Sarah Meiklejohn
Kazue Sako (Eds.)

LNCS 10957

Financial Cryptography and Data Security

22nd International Conference, FC 2018
Nieuwpoort, Curaçao, February 26 – March 2, 2018
Revised Selected Papers

Springer

Lecture Notes in Computer Science 10957

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

More information about this series at http://www.springer.com/series/7410

Sarah Meiklejohn · Kazue Sako (Eds.)

Financial Cryptography and Data Security

22nd International Conference, FC 2018
Nieuwpoort, Curaçao, February 26 – March 2, 2018
Revised Selected Papers

Springer

Editors
Sarah Meiklejohn
Department of Computer Science
University College London
London, UK

Kazue Sako
Security Research Laboratories
NEC
Kawasaki, Japan

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-58386-9 ISBN 978-3-662-58387-6 (eBook)
https://doi.org/10.1007/978-3-662-58387-6

Library of Congress Control Number: 2018961171

LNCS Sublibrary: SL4 – Security and Cryptology

This Springer imprint is published by the registered company Springer-Verlag GmbH, DE
part of Springer Nature
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

Preface

The 22nd International Conference on Financial Cryptography and Data Security, FC 2018, was held from February 26 to March 2, 2018, at the Santa Barbara Beach Resort in Curaçao.

We received 110 papers by the submission deadline for the conference, which was September 22, 2017. Of these, 29 were accepted – two as short papers and 27 as full papers – resulting in an acceptance rate of 26.4%. The present proceedings volume contains revised versions of all the papers presented at the conference.

The conference started with an invited talk by Yassir Nawaz, Executive Director, JP Morgan Chase & Co, titled "Blockchain and Cryptography at JP Morgan Chase."

The Program Committee consisted of 46 members spanning both industry and academia and covering all facets of financial cryptography. The review process took place over a period of roughly two months and was double-blind. Each paper received at least three reviews; certain papers, including submissions by Program Committee members, received additional reviews. The Program Committee used the HotCRP system to organize the paper reviewing. The merits of each paper were discussed thoroughly and intensely in the online platform as we converged to the final decisions. In the end, a number of worthy papers still had to be rejected owing to the limited number of slots in the conference program. The Program Committee made a substantial effort in improving the quality of accepted papers in the post-notification stage: Seven of the papers were conditionally accepted; each one was assigned a shepherd from the Program Committee, who guided the authors in the preparation of the conference version.

A number of grateful acknowledgments are due. First and foremost, we would like to thank the authors of all submissions for contributing their work for peer review by the Program Committee. Their support of FC 2018 was the most important factor for the success of the conference. Second, we would like to thank the members of the Program Committee for investing a significant amount of their time in the review and discussion of the submitted papers. In addition to the Program Committee, 68 external reviewers were invited to contribute to the review process and we also thank them for their efforts. In total, 360 reviews were submitted, 3.27 on average per submission. We would like to thank Nicolas Christin for administrating the reviewing system.

The conference also featured a poster session. We are grateful to the presenters of the posters for submitting their work and presenting it at the conference.

The general chair of the conference was Rafael Hirschfeld. We would like to especially thank him for his continued and tireless efforts to make FC a success over the years. A special thanks also goes to the board of directors of the International Financial Cryptography Association and the Financial Cryptography Steering Committee for their support and guidance.

Finally, we would like to thank all our sponsors this year, whose generous support was crucial in making the conference a success. In particular our platinum sponsors

Binary District, Blockchain Institute, Ethereum Foundation, and Zcash, our gold sponsors Kadena, Maker, and Protocol Labs, our silver Sponsors BANEX, Blockstream, Chia, *Journal of Cybersecurity*, Mosaic, and TrueBit, and our sponsor in kind WorldPay.

September 2018

Sarah Meiklejohn
Kazue Sako

Organization

Program Committee

Shashank Agrawal	Visa Research, USA
Ross Anderson	Cambridge University, UK
Elli Androulaki	IBM Research Zurich, Switzerland
Diego F. Aranha	University of Campinas, Brazil
Frederik Armknecht	University of Mannheim, Germany
Alex Biryukov	University of Luxembourg, Luxembourg
Jeremiah Blocki	Purdue University, USA
Rainer Böhme	Universität Innsbruck, Austria
Christian Cachin	IBM Research Zurich, Switzerland
Srdjan Capkun	ETH Zurich, Switzerland
Nicolas Christin	Carnegie Mellon University, USA
Jeremy Clark	Concordia University, Canada
Craig Costello	Microsoft Research, USA
Jean Paul Degabriele	TU Darmstadt, Germany
Maria Dubovitskaya	IBM Research Zurich, Switzerland
Ittay Eyal	Technion, Israel
Dario Fiore	IMDEA Software Institute, Spain
Bryan Ford	EPFL, Switzerland
Christina Garman	Purdue University, USA
Arthur Gervais	Imperial College London, UK
Jens Groth	University College London, UK
Feng Hao	Newcastle University, UK
Ryan Henry	Indiana University, USA
Nick Hopper	University of Minnesota, USA
Kévin Huguenin	University of Lausanne, Switzerland
Marc Joye	NXP Semiconductors, USA
Ghassan Karame	NEC Laboratories Europe, Germany
Stefan Katzenbeisser	TU Darmstadt, Germany
Aggelos Kiayias	University of Edinburgh & IOHK, UK
Markulf Kohlweiss	Microsoft Research, UK
Catherine Meadows	Naval Research Laboratory, USA
Andrew Miller	University of Illinois at Urbana-Champaign, USA
Tyler Moore	The University of Tulsa, USA
Steven Murdoch	University College London, UK
Satoshi Obana	Hosei University, Japan
Olya Ohrimenko	Microsoft Research, UK
Charalampos Papamanthou	University of Maryland, USA
Bart Preneel	KU Leuven, Belgium

Elizabeth Quaglia	Royal Holloway University of London, UK
Reihaneh Safavi-Naini	University of Calgary, Canada
Koutarou Suzuki	NTT Secure Platform Laboratories, Japan
Nicholas Weaver	ICSI & UC Berkeley, USA
Eric Wustrow	University of Colorado Boulder, USA
Zhenfeng Zhang	Institute of Software, Chinese Academy of Sciences, China
Aviv Zohar	The Hebrew University, Israel

Additional Reviewers

Abadi, Aydin
Abramova, Svetlana
Arapinis, Myrto
Avizheh, Sepideh
Bercea, Ioana
Böhl, Florian
Bootle, Jonathan
Bünz, Benedikt
Cerulli, Andrea
Chabanne, Hervé
Chepurnov, Alex
Christodorescu, Mihai
De Caro, Angelo
Demertzis, Ioannis
Dmitrienko, Alexandra
Elkhiyaoui, Kaoutar
Faonio, Antonio
Feher, Daniel
Fraser, Ashley
Fröwis, Michael
Gambs, Sébastien
Georgiou, Marios
Großschädl, Johann
Gutmann, Andreas
Harasser, Patrick
Hicks, Alexander
Hisil, Huseyin
Humbert, Mathias
Islam, Morshed
Janson, Christian
Jäschke, Angela
Karati, Sabyasachi
Karvelas, Nikolaos
Kikuchi, Ryo
Lacharité, Marie-Sarah
Litos, Orfeas
Lueks, Wouter
Machuletz, Dominique
Maller, Mary
Marson, Giorgia Azzurra
Massolino, Pedro Maat
McCorry, Patrick
Meyer, Maxime
Mikhalev, Vasily
Moosavi, Mahsa
Müller, Christian
Nguenewou, Patrick Towa
Ren, Ling
Salajegheh, Negin
Salehi, Fariborz
Schliep, Michael
Scholl, Peter
Sharifian, Setareh
Shumow, Dan
Sibut-Pinote, Thomas
Singelée, Dave
Sinha, Rohit
Sompolinsky, Yonatan
Stebila, Douglas
Tikhomirov, Sergei
Tramer, Florian
Udovenko, Aleksei
Vauclair, Marc
Vukolić, Marko
Wu, David
Yasunaga, Kenji
Zaverucha, Greg
Zhang, Yupeng

Contents

Privacy and Data Processing

Attacks

Applied Cryptography

Anonymity in Distributed Systems

Privacy

"Major Key Alert!" Anomalous Keys in Tor Relays

George Kadianakis[1], Claudia V. Roberts[2], Laura M. Roberts[2], and Philipp Winter[2(✉)]

[1] The Tor Project, Cambridge, USA
[2] Princeton University, Princeton, USA
pwinter@cs.princeton.edu

Abstract. In its more than ten years of existence, the Tor network has seen hundreds of thousands of relays come and go. Each relay maintains several RSA keys, amounting to millions of keys, all archived by The Tor Project. In this paper, we analyze 3.7 million RSA public keys of Tor relays. We (*i*) check if any relays share prime factors or moduli, (*ii*) identify relays that use non-standard exponents, (*iii*) characterize malicious relays that we discovered in the first two steps, and (*iv*) develop a tool that can determine what onion services fell prey to said malicious relays. Our experiments revealed that ten relays shared moduli and 3,557 relays—almost all part of a research project—shared prime factors, allowing adversaries to reconstruct private keys. We further discovered 122 relays that used non-standard RSA exponents, presumably in an attempt to attack onion services. By simulating how onion services are positioned in Tor's distributed hash table, we identified four onion services that were targeted by these malicious relays. Our work provides both The Tor Project and onion service operators with tools to identify misconfigured and malicious Tor relays to stop attacks before they pose a threat to Tor users.

Keywords: Tor · RSA · Cryptography · Factorization · Onion service

1 Introduction

Having seen hundreds of thousands of relays come and go over the last decade, the Tor network is the largest volunteer-run anonymity network. To implement onion routing, all the relays maintain several RSA key pairs, the most important of which are a medium-term key that rotates occasionally and a long-term key that ideally never changes. Most relays run The Tor Project's reference C implementation on dedicated Linux systems, but some run third-party implementations or operate on constrained systems such as Raspberry Pis which raises the question of whether these machines managed to generate safe keys

All four authors contributed substantially and share first authorship. The names are ordered alphabetically.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 3–19, 2018.
https://doi.org/10.1007/978-3-662-58387-6_1

upon bootstrapping. Past work has investigated the safety of keys in TLS and SSH servers [15], in nation-wide databases [3], as well as in POP3S, IMAPS, and SMTPS servers [14]. In this work, we study the Tor network and pay particular attention to Tor-specific aspects such as onion services.

Relays with weak cryptographic keys can pose a significant threat to Tor users. The exact impact depends on the type of key that is vulnerable. In the best case, an attacker only manages to compromise the TLS layer that protects Tor cells, which are also encrypted. In the worst case, an attacker compromises a relay's long-term "identity key," allowing her to impersonate the relay. To protect Tor users, we need methods to find relays with vulnerable keys and remove them from the network before adversaries can exploit them.

Drawing on a publicly-archived dataset of 3.7 million RSA public keys [30], we set out to analyze these keys for weaknesses and anomalies: we looked for shared prime factors, shared moduli, and non-standard RSA exponents. To our surprise, we found more than 3,000 keys with shared prime factors, most belonging to a 2013 research project [4]. Ten relays in our dataset shared a modulus, suggesting manual interference with the key generation process. Finally, we discovered 122 relays whose RSA exponent differed from Tor's hard-coded exponent. Most of these relays were meant to manipulate Tor's distributed hash table (DHT) in an attempt to attack onion services as we discuss in Sect. 5.4. To learn more, we implemented a tool—itos[1]—that simulates how onion services are placed on the DHT, revealing four onion services that were targeted by some of these malicious relays. Onion service operators can use our tool to monitor their services' security; e.g., a newspaper can make sure that its SecureDrop deployment—which uses onion services—is safe [11].

The entities responsible for the incidents we uncovered are as diverse as the incidents themselves: researchers, developers, and actual adversaries were all involved in generating key anomalies. By looking for information that relays had in common, such as similar nicknames, IP address blocks, uptimes, and port numbers, we were able to group the relays we discovered into clusters that were likely operated by the same entities, shedding light on the anatomy of real-world attacks against Tor.

We publish all our source code and data, allowing third parties such as The Tor Project to continuously check the keys of new relays and alert developers if any of these keys are vulnerable or non-standard.[2] Tor developers can then take early action and remove these relays from the network before adversaries get the chance to take advantage of them. In summary, we make the following three contributions:

- We analyze a dataset consisting of 3.7 million RSA public keys for weak and non-standard keys, revealing thousands of affected keys.
- We characterize the relays we discovered, show that many were likely operated by a single entity, and uncover four onion services that were likely targeted.

[1] The name is an acronym for "identifying targeted onion services.".
[2] Our project page is available online at https://nymity.ch/anomalous-tor-keys/.

– Given a set of malicious Tor relays that served as "hidden service directories," we develop and implement a method that can reveal what onion services these relays were targeting.

The rest of this paper details our project. In Sect. 2, we provide background information, followed by Sect. 3 where we discuss related work. In Sect. 4, we describe our method, and Sect. 5 presents our results. We discuss our work in Sect. 6 and conclude in Sect. 7.

2 Background

We now provide brief background on the RSA cryptosystem, how the Tor network employs RSA, and how onion services are implemented in the Tor network.

2.1 The RSA Cryptosystem

The RSA public key cryptosystem uses key pairs consisting of a public encryption key and a privately held decryption key [27]. The encryption key, or "RSA public key," is comprised of a pair of positive integers: an exponent e and a modulus N. The modulus N is the product of two large, random prime numbers p and q. The corresponding decryption key, or "RSA private key," is comprised of the positive integer pair d and N, where $N = pq$ and $d = e^{-1} \bmod (p-1)(q-1)$. The decryption exponent d is efficient to compute if e and the factorization of N are known.

The security of RSA rests upon the difficulty of factorizing N into its prime factors p and q. While factorizing N is impractical given sufficiently large prime factors, the greatest common divisor (GCD) of *two moduli* can be computed in mere microseconds. Consider two distinct RSA moduli $N_1 = pq_1$ and $N_2 = pq_2$ that share the prime factor p. An attacker could quickly and easily compute the GCD of N_1 and N_2, which will be p, and then divide the moduli by p to determine q_1 and q_2, thus compromising the private key of both key pairs. Therefore, it is crucial that both p and q are determined using a strong random number generator with a unique seed.

Even though the naive GCD algorithm is very efficient, our dataset consists of more than 3.7 million keys and naively computing the GCD of every pair would take more than three years of computation (assuming 15 µs per pair). Instead, we use the fast pairwise GCD algorithm by Bernstein [2] which can perform the computation at hand in just a few minutes.

2.2 The Tor Network

The Tor network is among the most popular tools for digital privacy and anonymity. As of December 2017, the Tor network consists of around 7,000 volunteer-run relays [31]. Each hour, information about all relays[3] is summarized

[3] This information includes IP addresses, ports, version numbers, and cryptographic information, just to name a few.

in the *network consensus*, which is used by clients to bootstrap a connection to the Tor network. The network consensus is produced by eight geographically-distributed *directory authorities*, machines run by individuals trusted by The Tor Project. For each relay in the consensus, there is a pointer to its *descriptor*, which contains additional, relay-specific information such as cryptographic keys.

Each of the ∼7,000 relays maintains RSA, Curve25519, and Ed25519 key pairs to authenticate and protect client traffic [9, Sect. 1.1]. In this work, we analyze the RSA keys. We leave the analysis of the other key types for future work. Each Tor relay has the following three 1024-bit RSA keys:

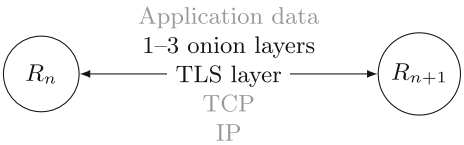


Fig. 1. The protocol stack between two Tor relays R_n and R_n+1. The lowest encryption layer is a TLS connection that contains one (between the middle and exit relay) to three (between the client and guard relay) onion layers. The onion layers protect the application data that the client is sending.

Identity key. Relays have a long-term identity key that they use only to sign documents and certificates. Relays are frequently referred to by their fingerprints, which is a hash over their identity key. The compromise of an identity key would allow an attacker to impersonate a relay by publishing spoofed descriptors signed by the compromised identity key.

Onion key. Relays use medium-term onion keys to decrypt cells when circuits are created. The onion key is only used in the Tor Authentication Protocol that is now superseded by the ntor handshake [13]. A compromised onion key allows the attacker to read the content of cells until the key pair rotates, which happens after 28 days [32, Sect. 3.4.1]. However, the onion key layer is protected by a TLS layer (see Fig. 1) that an attacker may have to find a way around.

Connection key. The short-term connection keys protect the connection between relays using TLS and are rotated at least once a day [9, Sect. 1.1]. The TLS connection provides defense in depth as shown in Fig. 1. If compromised, an attacker is able to see the encrypted cells that are exchanged between Tor relays.

In our work we consider the identity keys and onion keys that each relay has because the Tor Project has been archiving the public part of the identity and onion keys for more than ten years, allowing us to draw on a rich dataset [30]. The Tor Project does not archive the connection keys because they have short-term use and are not found in the network consensus or relay descriptors.

2.3 Onion Services

In addition to client anonymity, the Tor network allows operators to set up anonymous servers, typically called "onion services."[4] The so-called "hidden service directories," or "HSDirs," are a subset of all Tor relays and comprise a distributed hash table (DHT) that stores the information necessary for a client to connect to an onion service. These HSDirs are a particularly attractive target to adversaries because they get to learn about onion services that are set up in the Tor network. An onion service's position in the DHT is governed by the following equations:

$$\begin{aligned} \textit{secret-id-part} &= \textit{SHA}-1(\textit{time-period} \mid \\ &\qquad \textit{descriptor-cookie} \mid \\ &\qquad \textit{replica}) \\ \textit{descriptor-id} &= \textit{SHA}-1(\textit{permanent-id} \mid \\ &\qquad \textit{secret-id-part}) \end{aligned} \tag{1}$$

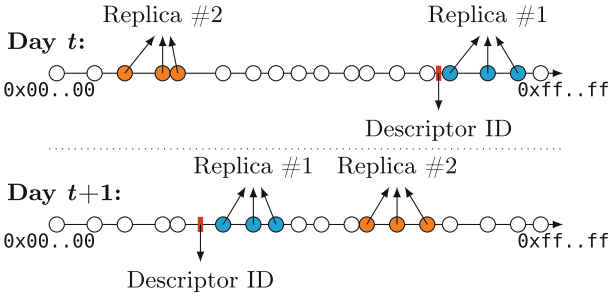


Fig. 2. Each day, an onion service places its descriptor ID at a pseudorandom location in Tor's "hash ring," which consists of all HSDir relays (illustrated as circles).

Secret-id-part depends on three variables: *time-period* represents the number of days since the Unix epoch; *descriptor-cookie* is typically unused and hence empty; and *replica* is set to both the values 0 and 1, resulting in two hashes for *secret-id-part*. The concatenation of both *permanent-id* (the onion service's hashed public key) and *secret-id-part* is hashed, resulting in *descriptor-id*, which determines the position in the DHT. When arranging all HSDirs by their fingerprint in ascending order, the three immediate HSDir neighbors in the positive direction constitute the first replica while the second replica is at another, pseudorandom location, as shown in Fig. 2. The onion service's descriptor ID and hence, its two replicas, changes every day when *time-period* increments.

[4] The term "hidden services" was used in the past but was discontinued, in part because onion services provide more than just "hiding" a web site.

3 Related Work

In 2012, Lenstra et al. [20] and Heninger et al. [15] independently analyzed a large set of RSA public keys used for TLS, SSH, and PGP. Both groups discovered that many keys shared prime factors, allowing an attacker to efficiently compute the corresponding private keys. The researchers showed that the root cause was weak randomness at the time of key generation: Many Internet-connected devices lack entropy sources, resulting in predictable keys.

One year later, Bernstein et al. [3] showed similar flaws in Taiwan's national "Citizen Digital Certificate" database. Among more than two million 1024-bit RSA keys, the authors discovered 184 vulnerable keys, 103 of which shared prime factors. The authors could break the remaining 81 keys by applying a Coppersmith-type partial-key-recovery attack [6,7].

Valenta et al. [36] optimized popular implementations for integer factorization, allowing them to factor 512-bit RSA public keys on Amazon EC2 in under four hours for only $75. The authors then moved on to survey the RSA key sizes that are used in popular protocols such as HTTPS, DNSSEC, and SSH, discovering numerous keys of only 512 bits.

Most recently, in 2016, Hastings et al. [14] revisited the problem of weak keys and investigated how many such keys were still on the Internet four years after the initial studies. The authors found that many vendors and device owners never patched their vulnerable devices. Surprisingly, the number of vulnerable devices has actually *increased* since 2012.

4 Method

In this section, we discuss how we drew on a publicly-available dataset (Sect. 4.1) and used Heninger and Halderman's fastgcd [16] tool to analyze the public keys that we extracted from this dataset (Sect. 4.2).

4.1 Data Collection

The Tor Project archives data about Tor relays on its CollecTor platform [30], allowing researchers to learn what relays were online at any point in the past. Drawing on this data source, we compiled a set of RSA keys by downloading all server descriptors from December 2005 to December 2016 and extracting the identity and onion keys with the Stem Python library [19]. Table 1 provides an overview of the resulting dataset—approximately 200 GB of unzipped data. Our 3.7 million public keys span eleven years and were created on one million IP addresses.

Our dataset also contains the keys of Tor's directory authorities. The authorities' keys are particularly sensitive: If an attacker were to compromise more than half of these keys, she could create a malicious network consensus—which could consist of attacker-controlled relays only—that would then be used by Tor clients. Therefore these keys are paramount to the security of the Tor network.

Table 1. An overview of our RSA public key dataset.

First key published	2005-12
Last key published	2016-12
Number of relays (by IP address)	1,083,805
Number of onion keys	3,174,859
Number of identity keys	588,945
Total number of public keys	3,763,804

4.2 Finding Vulnerable Keys

To detect weak, potentially factorable keys in the Tor network, we used Heninger and Halderman's fastgcd [16] tool which takes as input a set of moduli from public keys and then computes the pair-wise greatest common divisor of these moduli. Fastgcd's C implementation is based on a quasilinear-time algorithm for factoring a set of integers into their co-primes. We used the PyCrypto library [21] to turn Tor's PKCS#1-padded, PEM-encoded keys into fastgcd's expected format, which is hex-encoded moduli. Running fastgcd over our dataset took less than 20 min on a machine with dual, eight-core 2.8 GHz Intel Xeon E5 2680 v2 processors with 256 GB of RAM.

Fastgcd benefits from having access to a pool of moduli that is as large as possible because it allows the algorithm to draw on a larger factor base to use on each key [15]. To that end, we reached out to Heninger's group at the University of Pennsylvania, and they graciously augmented their 129 million key dataset with our 3.6 million keys and subsequently searched for shared factors. The number of weak keys did not go up, but this experiment gave us more confidence that we had not missed weak keys.

5 Results

We present our results in four parts, starting with shared prime factors (Sect. 5.1), followed by shared moduli (Sect. 5.2), unusual exponents (Sect. 5.3), and finally, targeted onion services (Sect. 5.4).

5.1 Shared Prime Factors

Among all 588,945 identity keys, fastgcd found that 3,557 (0.6%) moduli share prime factors. We believe that 3,555 of these keys were all controlled by a single research group, and upon contacting the authors of the Security & Privacy 2013 paper entitled "Trawling for Tor hidden services" [4], we received confirmation that these relays were indeed run by their research group. The authors informed us that the weak keys were caused by a shortcoming in their key generation tool. The issue stemmed from the fact that their tool first generated thousands of prime numbers and then computed multiple moduli using combinations of

those prime numbers in a greedy fashion without ensuring that the same primes were not reused. Because of the following shared properties, we are confident that all relays were operated by the researchers:

1. All relays were online either between November 11, 2012 and November 16, 2012 or between January 14, 2013 and February 6, 2013, suggesting two separate experiments. We verified this by checking how long the relays stayed in the Tor network consensus. The Tor consensus is updated hourly and documents which relays are available at a particular time. This data is archived by The Tor Project and is made publicly available on the CollecTor platform [30].
2. All relays exhibited a predictable port assignment scheme. In particular, we observed ports {7003, 7007, ..., 7043, 7047} and {8003, 8007, ..., 8043, 8047}.
3. Except for two machines that were located in Russia and Luxembourg, all machines were hosted in Amazon's EC2 address space. All machines except the one located in Luxembourg used Tor version 0.2.2.37.
4. All physical machines had multiple fingerprints. 1,321 of these 3,557 relays were previously characterized by Winter et al. [38, Sect. 5.1].

The remaining two keys belonged to a relay named "DesasterBlaster," whose origins we could not determine. Its router descriptor indicates that the relay has been hosted on a MIPS machine which might suggest an embedded device with a weak random number generator:

```
router DesasterBlaster 62.226.55.122 9001 0 0
platform Tor 0.2.2.13-alpha on Linux mips
```

To further investigate, we checked whether the relay "DesasterBlaster" shares prime factors with any other relays. It appears that the relay has rotated multiple identity keys, and it only shares prime factors with its own keys. Unfortunately the relay did not have any contact information configured which is why we could not get in touch with its operator.

5.2 Shared Moduli

In addition to finding shared prime factors, we discovered relays that share a *modulus*, giving them the ability to calculate each other's private keys. With p, q, and each other's es in hand, the two parties can compute each other's decryption exponent d, at which point both parties now know the private decryption keys.

Table 2 shows these ten relays with shared moduli clustered into four groups. The table shows the relays' truncated, four-byte fingerprint, IP addresses, and RSA exponents. Note that the Tor client hard-codes the RSA exponent to 65,537 [9, Sect. 0.3], a recommended value that is resistant to attacks against low public exponents [5, Sect. 4]. Any value other than 65,537 indicates non-standard key generation. All IP addresses were hosted by OVH, a popular French hosting provider, and some of the IP addresses hosted two relays, as our color coding indicates. Finally, each group shared a four- or five-digit prefix in their fingerprints.

We believe that a single attacker controlled all these relays with the intention to manipulate the distributed hash table that powers onion services [4]—the shared fingerprint prefix is an indication. Because the modulus is identical, we suspect that the attackers iterated over the relays' RSA exponents to come up with the shared prefix. The Tor Project informed us that it discovered and blocked these relays in August 2014 when they first came online.

Table 2. Four groups of relays that have a shared modulus. All relays further share a fingerprint prefix in groups of two or three, presumably to manipulate Tor's distributed hash table.

Short fingerprint	IP address	Exponent
838A296A	188.165.164.163	1,854,629
838A305F	188.165.26.13	718,645
838A71E2	178.32.143.175	220,955
2249EB42	188.165.26.13	4,510,659
2249EC78	178.32.143.175	1,074,365
E1EFA388	188.165.3.63	18,177
E1EF8985	188.165.138.181	546,019
E1EF9EB8	5.39.122.66	73,389
410BA17E	188.165.138.181	1,979,465
410BB962	5.39.122.66	341,785

5.3 Unusual Exponents

Having accidentally found a number of relays with non-standard exponents in Sect. 5.2, we checked if our dataset featured more relays with exponents other than 65,537. Non-standard exponents may indicate that a relay was after a specific fingerprint in order to position itself in Tor's hash ring. To obtain a fingerprint with a given prefix, an adversary repeatedly has to modify any of the underlying key material p, q, or e until they result in the desired prefix. Repeated modification of e is significantly more efficient than modifying p or q because it is costly to verify if a large number is prime. Leveraging this method, the tool Scallion [29] generates vanity onion service domains by iterating over the service's public exponent.

Among all of our 3.7 million keys, 122 possessed an exponent other than 65,537. One relay had both non-standard identity *and* onion key exponents while all remaining relays only had non-standard identity key exponents. Ten of these relays further had a shared modulus, which we discuss in Sect. 5.2. Assuming that these relays positioned themselves in the hash ring to attack an onion service, we wanted to find out what onion services they targeted. One can identify the victims by first compiling a comprehensive list of onion services and then

determining each service's position in the hash ring at the time the malicious HSDirs were online.

5.4 Identifying Targeted Onion Services

We obtained a list of onion services by augmenting the list of the Ahmia search engine [25] with services that we discovered via Google searches and by contacting researchers who have done similar work [23]. We ended up with a list of 17,198 onion services that were online at some point in time. Next, we developed a tool that takes as input our list of onion services and the malicious HSDirs we discovered.[5] The tool then calculates all descriptors these onion services ever generated and checks if any HSDir shared five or more hex digits in its fingerprint prefix with the onion service's descriptor. We chose the threshold of five manually because it is unlikely to happen by chance yet easy to create a five-digit collision.

It is difficult to identify all targeted onion services because (*i*) our list of onion services does not tell us when a service was online, (*ii*) an HSDir could be responsible for an onion service simply by chance rather than on purpose, resulting in a false positive, and (*iii*) our list of onion services is not exhaustive, so we are bound to miss victims. Nevertheless our tool identified four onion services (see Table 3) for which we have strong evidence that they were purposely targeted. While HSDirs are frequently in the vicinity of an onion service's descriptor by accident, the probability of being in its vicinity for several days in a row or cover both replicas by chance is negligible. Table 4 shows all partial collisions in detail. Because none of these four services seem to have been intended for private use, we are comfortable publishing them.

Table 3. The four onion services that were most likely targeted at some point. The second column indicates if only one or both replicas were attacked while the third column shows the duration of the attack.

Onion service	Replicas	Attack duration
`22u75kqyl666joi2.onion`	2	Two consecutive days
`n3q7l52nfpm77vnf.onion`	2	Six non-consecutive days
`silkroadvb5piz3r.onion`	1	Nine mostly consecutive days
`thehub7gqe43miyc.onion`	2	One day

`22u75kqyl666joi2.onion`. The service appears to be offline today, so we were unable to see for ourselves what it hosted. According to cached index pages we found online, the onion service used to host a technology-focused forum in Chinese. A set of relays targeted the onion service on both August 14 and 15, 2015 by providing nine out of the total of twelve responsible HSDirs.

[5] Both the tool and our list of onion services are available online at https://nymity.ch/anomalous-tor-keys/.

`n3q7l52nfpm77vnf.onion`. As of February 2017, the service is still online, hosting the "Marxists Internet Archive," an online archive of literature.[6] A set of relays targeted the onion service from November 27 to December 4, 2016. The malicious HSDirs acted inconsistently, occasionally targeting only one replica.

`silkroadvb5piz3r.onion`. The onion service used to host the Silk Road marketplace, whose predominant use was a market for narcotics. The service was targeted by a set of relays from May 21 to June 3, 2013. The HSDirs were part of a measurement experiment that resulted in a blog post [26].

`thehub7gqe43miyc.onion`. The onion service used to host a discussion forum, "The Hub," focused on darknet markets. A set of relays targeted both of The Hub's replicas from August 22, 2015.

Our data cannot provide insight into what the HSDirs did once they controlled the replicas of the onion services they targeted. The HSDirs could have counted the number of client requests, refused to serve the onion service's descriptor to take it offline, or correlate client requests with guard relay traffic in order to deanonymize onion service visitors as it was done by the CMU/SEI researchers in 2014 [8]. Since these attacks were short-lived we find it unlikely that they were meant to take offline the respective onion services.

6 Discussion

We now turn to the technical and ethical implications of our research, propose possible future work, and explain how the next generation of onion services will thwart DHT manipulation attacks.

6.1 Implications of Anomalous Tor Keys

Implications for the Network. As touched on earlier in Sect. 2.2, the main use of the identity key in Tor is to sign the relay's descriptor, which includes various information about the relay, e.g., its IP address and contact information. Relays publish their public identity keys in their descriptor. The network consensus acts as the public key infrastructure of Tor. Signed by the directory authorities whose public keys are hard-coded in Tor's source code, the network consensus points to the descriptors of each Tor relay that is currently online. If an attacker were to break the identity key of a relay (as we demonstrated), she could start signing descriptors in the relay's name and publishing them. The adversary could publish whatever information she wanted in the descriptor, e.g. her own IP address, keys, etc., in order to fool Tor clients. In other words, weak keys allow adversaries to obtain the affected relay's reputation which matters because Tor clients make routing decisions based on this reputation.

[6] The onion service seems to be identical to the website https://www.marxists.org (visited on 2017-05-09).

The Tor protocol's use of forward secrecy mitigates the potential harm of weak keys. Recall that a relay's long-lived identity keys are only used to sign data, so forward secrecy does not apply here. Onion keys, however, are used for decryption and encryption and are rotated by default every 28 days [32, Sect. 3.4.1]. An attacker who manages to compromise a weak onion key is still faced with the underlying TLS layer, shown in Fig. 1, which provides defense in depth. The Tor specification requires the keys for the TLS layer to be rotated at least once a day [9, Sect. 1.1], making it difficult to get any use out of compromised onion keys.

Implications for Tor Users. To understand how Tor users are affected by weak keys we need to distinguish between *targeting* and *hoovering* adversaries.[7] The goal of targeting adversaries is to focus an attack on a small number of users among the large set of all Tor users. Generally speaking, weak keys can be problematic in a targeted setting if they allow an attacker to gain access to a Tor relay she would otherwise be unable to control. This can be the case if the attacker learned the targeted user's guard relay, and the guard happens to have weak keys. However, judging by our experimental results, the probability of an attacker's knowing a targeted user's guard relay *and* said guard relay's having vulnerable keys is very low.

Hoovering adversaries are opportunistic by definition and seek to deanonymize as many Tor users as possible. Recall that Tor clients use a long-lived guard relay as their first hop and two randomly chosen relays for the next two hops.[8] A single compromised relay is not necessarily harmful to users but weak keys can be a problem if a user happens to have a guard relay with weak keys *and* selects an exit relay that also has weak keys, allowing the hoovering adversary to deanonymize the circuit. Again, considering the low prevalence of weak keys and the ease with which The Tor Project could identify and block relays with weak keys, hoovering adversaries pose no significant threat.

6.2 Preventing Non-standard Exponents

Recall that the Tor reference implementation hard-codes its public RSA exponent to 65,537 [9, Sect. 0.3]. The Tor Project could prevent non-standard exponents by having the directory authorities reject relays whose descriptors have an RSA exponent other than 65,537, thus slowing down the search for fingerprint prefixes. Adversaries would then have to iterate over the primes p or q instead of the exponent, rendering the search computationally more expensive because of the cost of primality tests. Given that we discovered only 122 unusual exponents in over ten years of data, we believe that rejecting non-standard exponents is a viable defense in depth.

[7] We here use Jaggard and Syverson's nomenclature of an adversary that either targets specific Tor users (targeting) or hoovers up all available data to deanonymize as many users as possible (hoovering) [17].

[8] We refer to these relays as randomly chosen for simplicity, but the path selection algorithm is more complicated.

6.3 Analyzing Onion Service Public Keys

Future work should shed light on the public keys of onion services. Onion services have an incentive to modify their fingerprints to make them both recognizable and easier to remember. Facebook, for example, was lucky to obtain the easy-to-remember onion domain facebookcorewwwi.onion [24]. The tool Scallion assists onion service operators in creating such vanity domains [29]. The implications of vanity domains on usability and security are still poorly understood [37]. Unlike the public keys of relays, onion service keys are not archived, so a study would have to begin with actively fetching onion service keys.

6.4 Investigating the Vulnerability of Tor Relays to Attacks on Diffie-Hellman

Recent work has demonstrated how Diffie-Hellman key exchange is vulnerable to attack [1,10,35]. Because Tor uses Diffie-Hellman, we decided to investigate how it might be affected by those findings. The Tor specification states that the implementation uses the Second Oakley Group for Diffie-Hellman, where the prime number is 1024 bits long [9, Sect. 0.3]. To gather evidence of this usage, we performed an nmap scan on Tor relays using the ssl-dh-params script [12], which confirmed the Tor specification. The use of a 1024-bit prime is concerning because Adrian et al. [1] stated that "1024-bit discrete log may be within reach for state-level actors," and thus, they suggest a move to 2048 bits. The authors also mention that developers should move away from using 1024-bit RSA, as well, which Tor uses.

6.5 *In Vivo* Tor Research

Caution must be taken when conducting research using the live Tor network. Section 5.1 showed how a small mistake in key generation led to many vulnerable Tor relays. To keep its users safe, The Tor Project has recently launched a research safety board whose aim is to assist researchers in safely conducting Tor measurement studies [33]. This may entail running experiments in private Tor networks that are controlled by the researchers or using network simulators such as Shadow [18].

As for our own work, we were in close contact with Tor developers throughout our research effort and shared preliminary results as we progressed. Once we wrote up our findings in a technical report, we brought it to the broader Tor community's attention by sending an email to the tor-dev mailing list [28]. On top of that, we adopted open science practices and wrote both our code and paper in the open, allowing interested parties to follow our progress easily.

6.6 The Effect of Next-Generation Onion Services

As of December 2017, The Tor Project is testing the implementation of next-generation onion services [22]. In addition to stronger cryptographic primitives,

the design fixes the issue of predicting an onion service's location in the hash ring by incorporating a random element. This element is produced by having the directory authorities agree on a random number once a day [34]. The random number is embedded in the consensus document and used by clients to fetch an onion service's descriptor. Attackers will no longer be able to attack onion services by positioning HSDirs in the hash ring; while they have several hours to compute a key pair that positions their HSDirs next to the onion service's descriptor (which is entirely feasible), it takes at least 96 hours to get the HSDir flag from the directory authorities [32, Sect. 3.4.2], so attackers cannot get the flag in time. We expect this design change to disincentivize attackers from manipulating their keys to attack onion services.

7 Conclusion

Inspired by recent research that studied weak keys in deployed systems, we set out to investigate if the Tor network suffers from similar issues. To that end, we drew on a public archive containing cryptographic keys of Tor relays dating back to 2005, which we subsequently analyzed for weak RSA keys. We discovered that (*i*) ten relays shared an RSA modulus, (*ii*) 3,557 relays shared prime factors, and (*iii*) 122 relays used non-standard RSA exponents.

Having uncovered these anomalies, we then proceeded to characterize the affected relays, tracing back the issues to mostly harmless experiments run by academic researchers and hobbyists, but also to attackers that targeted Tor's distributed hash table which powers onion services. To learn more, we implemented a tool that can determine what onion services were attacked by a given set of malicious Tor relays, revealing four onion services that fell prey to these attacks.

The practical value of our work is twofold. First, our uncovering and characterizing of Tor relays with anomalous keys provides an anatomy of real-world attacks that The Tor Project can draw upon to improve its monitoring infrastructure for malicious Tor relays. Second, our work provides The Tor Project with tools to verify the RSA keys of freshly set up relays, making the network safer for its users. In addition, onion service operators can use our code to monitor their services and get notified if Tor relays make an effort to deanonymize their onion service. We believe that this is particularly useful for sensitive deployments such as SecureDrop instances.

Acknowledgements. We want to thank Nadia Heninger and Josh Fried for augmenting their database with our moduli and attempting to find factors in them. We also want to thank Ralf-Philipp Weinmann, Ivan Pustogarov, Alex Biryukov from the Trawling research team and Donncha O'Cearbhaill from The Tor Project for providing us with additional information that helped us in our analysis of the weak keys. Finally, we want to thank Edward W. Felten for providing valuable feedback on an earlier version of our paper. This research was supported by the Center for Information Technology Policy at Princeton University and the National Science Foundation Awards CNS-1540066, CNS-1602399, CNS-1111539, CNS-1314637, CNS-1520552, and CNS-1640548.

A Potentially Targeted Onion Services

Table 4. The details of the attacks on four onion services. The second column shows the fingerprints of the HSDirs that were participating in the attack. The third column shows the affected onion service descriptors, followed by the date of the attack in the last column.

Onion service	Truncated HSDir fingerprint	Truncated onion service descriptor	Date
22u75kqyl666joi2	325CAC0B7FA8CD77E39D	325CAC08B0A3180B590E	2015-08-14
	325CAC0AB1AAD27493B9	325CAC08B0A3180B590E	2015-08-14
	325CAC0A43B2121B81CD	325CAC08B0A3180B590E	2015-08-14
	FA256741ED22FD96AF5A	FA256740740356704AB8	2015-08-14
	FA256743ACFCA9B7C85D	FA256740740356704AB8	2015-08-14
	E5E778326AF0FF0A634A	E5E7783245096FB554A1	2015-08-15
	A5C59B3D0FFBDE88405E	A5C59B3CD34802FC4AC3	2015-08-15
	A5C59B3FCCD2FA8FAD42	A5C59B3CD34802FC4AC3	2015-08-15
	A5C59B3FD5625A0D85D1	A5C59B3CD34802FC4AC3	2015-08-15
n3q7l52nfpm77vnf	A0E83AA191220B240EC0	A0E83AA115098CA7FE9B	2016-11-27
	A0E83AA28382135DC839	A0E83AA115098CA7FE9B	2016-11-27
	EBF154DA21B49101ED5B	EBF154D809425D3E923E	2016-11-28
	EBF154D8BB6EECCC2921	EBF154D809425D3E923E	2016-11-28
	EBF154D9E2B10A2420E0	EBF154D809425D3E923E	2016-11-28
	6761D2BE758FA0D76822	6761D2BCF40FF34274F3	2016-11-29
	59E415D78921BFF88168	59E415D5075157CAADB7	2016-11-29
	26597E62875C498AC139	26597E6048BF7CC9D593	2016-11-30
	26597E61DDFEE78F336D	26597E6048BF7CC9D593	2016-11-30
	7CDB224FE64F2A50CC50	7CDB224DC51432C037C5	2016-11-30
	2D148D3EBF9D2B9D8CCB	2D148D3CB6C5FC4DCA14	2016-12-01
	2E25D8469331FEAE933D	2E25D842BF5DDA936BA2	2016-12-05
	2E25D847E579AED1B0EC	2E25D842BF5DDA936BA2	2016-12-05
	2E25D8454C96E20CF153	2E25D842BF5DDA936BA2	2016-12-05
	2E25D846564DCBE43CD2	2E25D842BF5DDA936BA2	2016-12-05
	2E25D8447518DA93B4FF	2E25D842BF5DDA936BA2	2016-12-05
	264EA12B47CBCC8043C5	264EA12410F7D9CD6E54	2016-12-05
	264EA1284855A596D5D6	264EA12410F7D9CD6E54	2016-12-05
	264EA12B4C46672E002C	264EA12410F7D9CD6E54	2016-12-05
silkroadvb5piz3r	BC89A92F53581C4F6169	BC89A889D3DF7F0027A5	2013-05-21
	712CA45AF4055E7AC69A	712CA3DEF4EB21C76A95	2013-05-22
	DE15299D7EE5882F0BEF	DE1529316F5172B35B8E	2013-05-23
	FF0BF54FBEEE7A003CE6	FF0BF49076AA63C97FA2	2013-05-24
	E9F25C4899F9DC81E48E	E9F25BBA0D4501FAE18B	2013-05-28
	B81B43C015DE42D05208	B81B43637F22592ECC80	2013-05-29
	59529817C6E797D78311	5952979BD9FEECE847E7	2013-05-31
	BCB332864640653892D4	BCB33236E0AD461DF585	2013-06-02
	51FC178DFF3D0B869760	51FC172F0062B623A39D	2013-06-03
thehub7gqe43miyc	F6961286D361F825A9AD	F6961286C2FEEA8DEDEB	2015-08-22
	F6961286C453F6A6381D	F6961286C2FEEA8DEDEB	2015-08-22
	F6961286D826D7D1C0F9	F6961286C2FEEA8DEDEB	2015-08-22
	816FEE16200BE1719D00	816FEE15D26F41A72039	2015-08-22

References

1. Adrian, D., et al.: Imperfect forward secrecy: how Diffie-Hellman fails in practice. In: CCS. ACM (2015). https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf. Accessed 22 Sept 2017
2. Bernstein, D.J.: How to find smooth parts of integers (2004). https://cr.yp.to/factorization/smoothparts-20040510.pdf. Accessed 9 May 2017
3. Bernstein, D.J., et al.: Factoring RSA keys from certified smart cards: coppersmith in the wild. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 341–360. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_18, https://smartfacts.cr.yp.to/smartfacts-20130916.pdf
4. Biryukov, A., Pustogarov, I., Weinmann, R.P.: Trawling for Tor hidden services: detection, measurement, deanonymization. In: Security and Privacy. IEEE (2013). http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf. Accessed 9 May 2017
5. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Not. Am. Math. Soc. **46**(2) (1999). http://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf. Accessed 9 May 2017
6. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_16
7. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. Cryptol. **10**(4), 233–260 (1997). https://www.di.ens.fr/~fouque/ens-rennes/coppersmith.pdf. Accessed 9 May 2017
8. Dingledine, R.: Tor security advisory: "relay early" traffic confirmation attack, July 2014. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack/. Accessed 9 May 2017
9. Dingledine, R., Mathewson, N.: Tor protocol specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. Accessed 9 May 2017
10. Dorey, K., Chang-Fong, N., Essex, A.: Indiscreet logs: Diffie-Hellman backdoors in TLS. In: NDSS. Internet Society (2017). https://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/ndss2017_04A-2_Dorey_paper.pdf. Accessed 19 Sept 2017
11. Freedom of the Press Foundation: SecureDrop. https://securedrop.org. Accessed 19 Sept 2017
12. Gajek, J.: ssl-dh-params. https://nmap.org/nsedoc/scripts/ssl-dh-params.html. Accessed 22 Sept 2017
13. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Des. Codes Cryptogr. **67**(2), 245–269 (2013). Accessed 9 May 2017
14. Hastings, M., Fried, J., Heninger, N.: Weak keys remain widespread in network devices. In: IMC. ACM (2016). https://www.cis.upenn.edu/~nadiah/papers/weak-keys/weak-keys.pdf. Accessed 9 May 2017
15. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: detection of widespread weak keys in network devices. In: USENIX Security. USENIX (2012). https://factorable.net/weakkeys12.extended.pdf. Accessed 9 May 2017
16. Heninger, N., Halderman, J.A.: Fastgcd. https://factorable.net/fastgcd-1.0.tar.gz. Accessed 9 May 2017
17. Jaggard, A.D., Syverson, P.: Oft target. In: HotPETs (2017). https://petsymposium.org/2017/papers/hotpets/oft-target-1707.pdf

18. Jansen, R., Hopper, N.: Shadow: running Tor in a box for accurate and efficient experimentation. In: NDSS. Internet Society (2012). http://www.robgjansen.com/publications/shadow-ndss2012.pdf. Accessed 9 May 2017
19. Johnson, D.: Stem docs. https://stem.torproject.org. Accessed 9 May 2017
20. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_37
21. Litzenberger, D.: PyCrypto - the Python cryptography toolkit. https://www.dlitz.net/software/pycrypto/. Accessed 9 May 2017
22. Mathewson, N.: Next-generation hidden services in Tor. https://gitweb.torproject.org/torspec.git/tree/proposals/224-rend-spec-ng.txt. Accessed 1 Aug 2017
23. Matic, S., Kotzias, P., Caballero, J.: CARONTE: detecting location leaks for deanonymizing Tor hidden services. In: CCS. ACM (2015). https://software.imdea.org/~juanca/papers/caronte_ccs15.pdf. Accessed 9 May 2017
24. Muffett, A.: Facebook brute forcing hidden services, October 2014. https://lists.torproject.org/pipermail/tor-talk/2014-October/035413.html. Accessed 9 May 2017
25. Nurmi, J.: Ahmia - search Tor hidden services. https://ahmia.fi/onions/. Accessed 9 May 2017
26. O'Cearbhaill, D.: Trawling Tor hidden service - mapping the DHT (2013). https://donncha.is/2013/05/trawling-tor-hidden-services/. Accessed 9 May 2017
27. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://people.csail.mit.edu/rivest/Rsapaper.pdf. Accessed 9 May 2017
28. Roberts, L.M.: Anomalous keys in Tor relays. Technical report, April 2017. https://lists.torproject.org/pipermail/tor-dev/2017-April/012161.html. Accessed 2 Aug 2017
29. Swanson, E.: Scallion - GPU-based onion hash generator. https://github.com/lachesis/scallion. Accessed 9 May 2017
30. The Tor Project: CollecTor. https://collector.torproject.org. Accessed 9 May 2017
31. The Tor Project: Servers - Tor metrics. https://metrics.torproject.org/networksize.html. Accessed 9 May 2017
32. The Tor Project: Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt. Accessed 2 Aug 2017
33. The Tor Project: Tor research safety board. https://research.torproject.org/safetyboard.html. Accessed 9 May 2017
34. The Tor Project: Tor shared random subsystem specification. https://gitweb.torproject.org/torspec.git/tree/srv-spec.txt. Accessed 2 Aug 2017
35. Valenta, L., et al.: Measuring small subgroup attacks against Diffie-Hellman. In: NDSS. Internet Society (2017). https://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/ndss2017_04A-1_Valenta_paper_0.pdf. Accessed 19 Sept 2017
36. Valenta, L., Cohney, S., Liao, A., Fried, J., Bodduluri, S., Heninger, N.: Factoring as a service. In: Financial Cryptography. ACM (2016). https://eprint.iacr.org/2015/1000.pdf. Accessed 9 May 2017
37. Winter, P.: Are vanity onion domains a good idea?, October 2015. https://moderncrypto.org/mail-archive/messaging/2015/001928.html. Accessed 9 May 2017
38. Winter, P., Ensafi, R., Loesing, K., Feamster, N.: Identifying and characterizing Sybils in the Tor network. In: USENIX Security. USENIX (2016). https://nymity.ch/sybilhunting/pdf/sybilhunting-sec16.pdf. Accessed 9 May 2017

On Purpose and by Necessity: Compliance Under the GDPR

David Basin[1], Søren Debois[2(✉)], and Thomas Hildebrandt[2]

[1] ETH Zürich, Zürich, Switzerland
basin@inf.ethz.ch
[2] IT University of Copenhagen, Copenhagen, Denmark
{debois,hilde}@itu.dk

Abstract. The European General Data Protection Regulation (GDPR) gives primacy to *purpose*: Data may be collected and stored only when (i) end-users have consented, often explicitly, to the purposes for which that data is collected, and (ii) the collected data is actually necessary for achieving these purposes. This development in data protection regulations begets the question: how do we audit a computer system's adherence to a purpose?

We propose an approach that identifies a purpose with a business process, and show how formal models of interprocess communication can be used to audit or even derive privacy policies. Based on this insight, we propose a methodology for auditing GDPR compliance. Moreover, we show how given a simple interprocess dataflow model, aspects of GDPR compliance can be determined algorithmically.

1 Introduction

Data protection is now heavily anchored in national and international law. The prime example of this is the European General Data Protection Regulation (the GDPR) [9], which strengthens previous data protection directives to give individuals more rights on how their personal data is processed. A central principle of data protection in general, and the GDPR in particular, is that organisations collecting and processing personal data must be explicit about how the data will be used and *the data is actually used for the purposes for which it was collected.*

Contrast this situation with standard access control, which regulates who may carry out which operations in a system. With few exceptions (e.g., history-based access control or access decisions incorporating environmental attributes) access is independent of context: if Alice has the right to access Bob's bank account balance, then she can do this for *any* purpose. This includes both intended purposes, such as serving as his customer relations manager, or unintended ones, such as selling information on his financial status to credit agencies.

Authors listed alphabetically. This work is supported in part by Innovation Fund Denmark, grant 7050-00034A, project "Effective, co-created & compliant adaptive case management for knowledge workers" (EcoKnow).

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 20–37, 2018.
https://doi.org/10.1007/978-3-662-58387-6_2

Modern data protection calls for something more: *access control relative to a purpose*.

The key difficulty then is that mainstream programming technologies do not leave us any obvious representation of purpose, much less one that can be reasonably related to sites of data collection or data use. There is, however, one area of computer science where a notion of purpose takes a center stage, and where formal models abound: The study of Business Process Management, in particular Business Process Modelling. Here, the operations (both IT and human) of a business are modelled in one of a variety of formal languages including Statecharts [11], UML [21,22], BPMN [2], GSM [13], CMMN [23], DECLARE [1,25], or DCR graphs [12,19]. Such a model will include details about both data collection and data use. Crucially, most definitions of "business process" *also* either implicitly or explicitly include the purpose of the process, although sometimes expressed in terms of a product to be manufactured or a service to be rendered.

We propose exploiting the formal notion of a business process model as a bridge between a system implementation and the GDPR. In doing so, we exploit that a business process model by its very nature embodies a particular purpose, while at the same time it specifies at what points data is collected and used. For instance, an online-shop will have an order-fulfilment process where a customer's address is used to ship a product. Our proposal conflates a formal model of that *process* with the *purpose* of order-fulfilment; the model then describes both data collection and data use.

This idea poses a challenge to formal business process models. Under the GDPR, we must account for the data transferred between processes: data collected for one purpose and used for another. For example, a mailing address might be collected in a customer registration process that is subsequently used in an order-fulfilment process. Typical process models do not give detailed accounts of such inter-process interactions. We posit that an interprocess dataflow model is *necessary* to audit GDPR compliance.

We show that formal models of interprocess communication enable the algorithmic verification of parts of GDPR compliance. However the GDPR in certain cases goes *beyond* what we can automatically verify. For example, it can be difficult to determine whether a text message is an advertisement or a notification about an upcoming delivery. In these cases, the underlying business processes themselves must be augmented with human actions, for example explicit manager approval of the text message. Our approach therefore supports automated compliance checking complimented by human actions when necessary.

In summary, we make the following contributions:

1. We show how a mechanism for relating *purpose* to implementation artefacts is necessary to demonstrate compliance with the GDPR (Sect. 4.1).
2. We put forward the idea of *identifying* a business process and a purpose (Sect. 4.2).
3. We identify inter-process communication as key to GDPR compliance (Sect. 4.3).

4. We propose a *methodology* for auditing GDPR compliance by decomposing the audit into verifying the compliance of an implementation to a process model, of a process model to a privacy policy, and of these latter two to the GDPR itself (Sect. 4.4).
5. We show how a formal process model allows us to verify compliance of certain aspects of the GDPR algorithmically (Definitions 5.1 and 5.4). In particular, we can *generate* compliant privacy policies.
6. Finally, we illustrate through examples that GDPR compliance cannot be fully achieved by algorithmic means, and that process models can fill the gap here by specifying needed human actions (Sect. 5.4).

2 The General Data Protection Regulation

The GDPR [9] was passed on April 14, 2016 and will come into force May 25, 2018. It embodies a major departure from current practices. It requires not only that data is only collected after obtaining consent from the user, but also that data is collected and used *only for specific purposes*, and *must be deleted when those purpose are no longer applicable*. The GDPR spells out these requirements in its notions of *purpose limitation* and *data minimisation*, its treatment of *consent*, and the *right to be forgotten*.

Purpose limitation [9, Article 5, Sect. 1(b)]:

> "*[Personal data shall be] collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes; [...]*"

Data minimisation [9, Article 5, Sect. 1(c)]:

> "*[Personal data shall be] adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed [...]*"

Consent (and its connection to purpose) [9, Recital (32), emphasis ours]:

> *"Consent should be given by a clear affirmative act establishing a freely given, specific, informed and unambiguous indication of the data subject's agreement to the processing of personal data relating to him or her [...]* ***Consent should cover all processing activities carried out for the same purpose or purposes. When the processing has multiple purposes, consent should be given for all of them."***

Right to be forgotten [9, Article 17, Sect. 1]:

> *"[...] the controller shall have the obligation to erase personal data without undue delay where one of the following grounds applies:*
> *(a) the personal data are no longer necessary in relation to the purposes for which they were collected or otherwise processed;*
> *(b) the data subject withdraws consent [...]"*

We remark that the GDPR mandates access control [9, Article 25, Sect. 1]:

> "*The controller shall implement appropriate technical and organisational measures for ensuring that, by default, only personal data which are necessary for each specific purpose of the processing are processed. [...] personal data are not made accessible without the individual's intervention to an indefinite number of natural persons.*"

Finally, the GDPR has teeth. It imposes two levels of fines, depending on which parts of the GDPR are violated. The highest level imposes fines of up to 20 million EUR or 4% of the organisation's world-wide turnover, whichever is higher [9, Article 83, Sect. 5]. Processing without consent is on the list of high-level infringements [9, Article 83 Sect. 5(a)].

For the purposes of the present paper, we shall emphasize the above concepts of purpose limitation, data minimisation, consent, and the right to be forgotten. However, the GDPR confers on citizens (data subjects) a remarkable range of additional rights, such as rights of "access" and "data portability" [9, Article 15 & 20] and the right to be notified of data breaches [9, Article 33].

In summary, for data collection to be GDPR compliant, the data must:

1. be collected for a purpose,
2. to which the user has consented, and
3. be necessary to achieve that purpose;
4. moreover the collected data must be deleted when it is no longer necessary for any purpose.

Contrast to Extant Privacy Policies. Privacy policies are statements about how an organization collects, processes, and more generally manages, the personal data of its customers or other individuals. An informal survey of existing policies (including Facebook [8], Google [10], and IBM [14]) shows that their essence effectively consists of two types of declarations:

- **The kinds of data collected**, e.g., credentials, cookies, purchases, etc.
- **How collected data is used**, e.g., to process orders, personalize offerings and advertisements, etc.

These statements may be augmented with additional information, such as how *non*-personally identifiable information may be used, which usages one may opt out of, security measures taken when storing and processing data, and the like.

From the above, we conclude that current best-practice is to formulate *coarse grained* privacy policies. Their essence amounts to two sets, a set DC of the kinds of data collected and a set DU of data usages. In some cases (e.g., Google), a relation (a subset of $DC \times DU$) is given, where it is indicated how particular data items are used, e.g., "we use information collected from cookies to improve your user experience." However, in most cases (e.g., Facebook, IBM) the description of DC is non-specific with respect to which data is involved in which usages, e.g., "The information you provide may be used for marketing purposes."

The GDPR requires more. For example, Recital 39 specifies that the purposes that data will be used must be transparently laid out in a privacy policy. In particular, it should be clear that the personal data should be "adequate, relevant, and limited to what is necessary for the purposes for which they are processed." Indeed, "personal data should be processed only if the purpose of the processing could not reasonably be fulfilled by other means." This requires *fine grained* privacy policies that clearly elucidate purposes and the associated data required. Our thesis is that business process models provide the right basis for this elucidation, both supporting the creation of fine-grained natural language privacy policies (e.g., informing data owners) and supporting audit and compliance (e.g., informing technical specialists).

3 Running Example

We provide an example from on-line retailing that we subsequently use to illustrate our methodology to audit compliance with the GDPR. An on-line retailer has customers who order goods using the retailer's homepage, pay with their credit cards, and expect to subsequently receive their orders by post. The retailer may engage in marketing, targeted or otherwise, using channels such as web-advertisements and e-mail.

We will focus just on the core processes of such a retailer, emphasizing what data is collected and used. These core processes are:

Register Customer: A prospective customer signs up with an on-line retailer. As part of this process, the customer provides his e-mail, his mailing addresses, and his credit card information.

Purchase: A registered customer selects a product on the retailer's homepage, pays using the recorded credit card number, and the retailer subsequently sends the product and invoice.

Mass Marketing: A customer's e-mail or physical address is used to send otherwise un-targeted advertisement.

Targeted Marketing: A customer's e-mail or physical address and past purchase history is used to send individually targeted advertisements.

In the following, we write processes in sans-serif, e.g., Mass Marketing, and descriptions of data classes in brackets, e.g., ⟨credit card number⟩.

In Fig. 1 we show example models of these processes, written in the BPMN [2] notation. In brief, the diagram comprises four pools, one for each process; inside each pool is a number of activities in boxes, some human, some automated. Activities with white envelopes take incoming messages, typically user input; black envelopes produce outgoing messages, typically output to the user. Activities marked with a person icon are undertaken by humans. The sequencing of activities is indicated with solid arrows between them. Activities may both produce and consume data stored in databases, indicated by dashed arrows. Note that the databases allow data to be shared between processes.

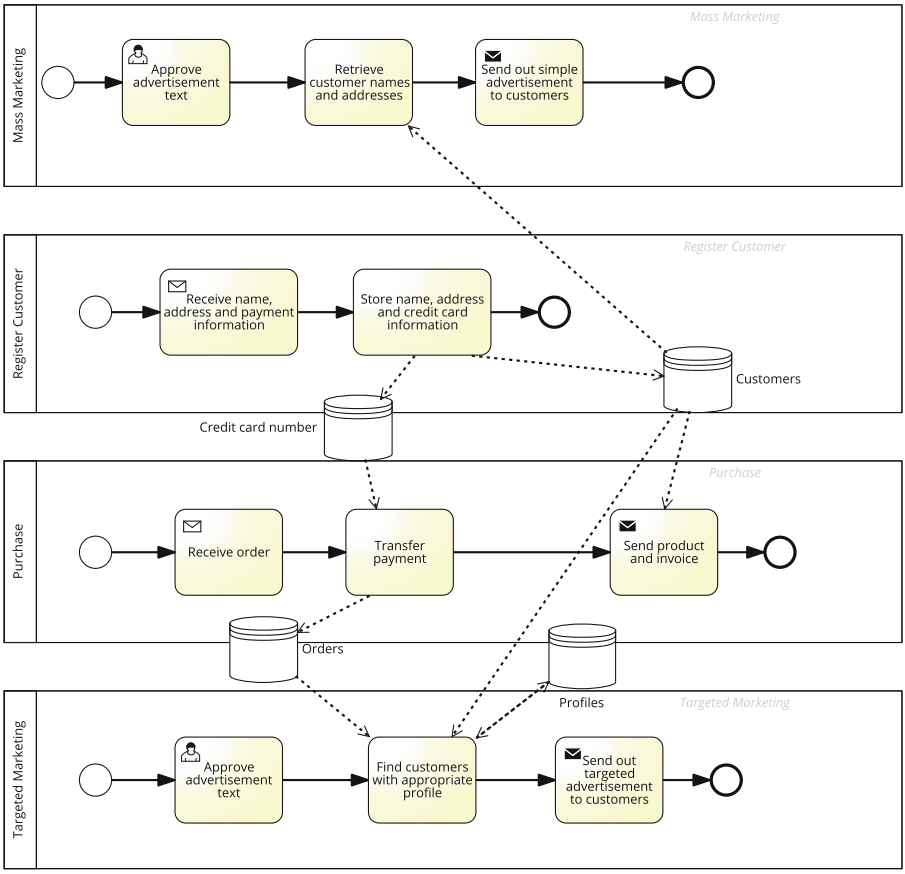


Fig. 1. BPMN model of the core on-line retailer processes.

4 Purposes

The GDPR's emphasis on *purpose* has interesting and subtle ramifications for both the design and audit of computer systems. In this section, we develop the idea that business process models encode purposes, and that this encoding can be used to analyse compliance with the GDPR.

4.1 Purpose and Compliance

As with any regulation that applies to computer systems, we are faced with two key questions:

- How do we *build* a computer system in a manner guaranteeing compliance?
- How do we *analyse* or *audit* a computer system for compliance?

Reviewing the conditions (1)–(4) from Sect. 2, which are required for a data-collection to be compliant with the GDPR, we see an immediate problem: The notion of compliance revolves around the notion of purpose, but purpose tends *not* to have an explicit representation in contemporary computer system implementations. Whereas notions like "user authentication" or "order management" usually have an *explicit* representation as lines of code and tables in databases, we seldom see implementation artefacts representing "the purpose" of a particular dataset. But to answer the two questions above, we must not only be able to identify the points at which data is collected (which is presumably easy), we must *also* associate that data with a purpose.

For example, suppose our on-line retailer collects a user's e-mail address during registration. There are legitimate uses for such information: it may be used as a user-id, to send invoices, or for password resets; alternatively, it might also be used for illegitimate purposes such as unsolicited marketing. By examining the system's code, we will readily discover where data may be collected and processed. However, it is impossible to determine from the code alone whether data being collected at a particular point is personal data or not, for what purpose data is being processed, and if it is really necessary for that purpose.

Even if we can technically determine every place in the implementation that accesses this e-mail address, we still may not be able to determine the *purpose* for that access. For example, the Mass Marketing process of our e-shop could enable staff to send arbitrary messages to *every* registered customer. Obviously, we cannot statically determine what the *purpose* of these messages might be: A staff member might send important information about deliveries ("Due to strikes at our logistics partner, all deliveries will be delayed."); marketing messages ("You have ordered recently from us. How about also buying an electric cat food dispenser?"); or even political propaganda ("Vote for me for president!").

Finally, we must delete data when it has served its purposes. But it is difficult to know when this is the case, especially in large computer systems where the same data may be used in multiple subsystems, for multiple purposes.

4.2 Business Processes as Purposes

We propose using the business processes [5] that the computer system in question supports to identify purposes and to classify the types of data collected. The key insight is that business processes *explicitly represent one or more purposes*. Here is a standard definition of a business process [5, pp. 5–7] (emphasis ours):

> "*a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. [...]* ***Processes are the structure by which an organization does what is necessary to produce value for its customers*** [...]"

The emphasized sentence highlights that a business process comprises both a purpose—the specifics of "to produce value for its customers"—as well as concrete steps—the specifics of "the structure by which." In practice, the purpose of a process will be most clearly represented by its title or perhaps a brief natural language statement. Through the description of "how work is done," a process description also gives us the necessary information about what data is collected, and where it is used. Moreover, when purposes are processes, we can determine when a purpose is served, e.g., when the corresponding process has terminated.

In general, it is reasonable to associate a "process" with a "purpose", e.g., the purchase process/purpose, the mass marketing process/purpose, the customer satisfaction evaluation process/purpose, or the warranty process/purpose.

4.3 Inter-process Communication

In practice, a company may collect data about customers in one process and use that data in another. In our example, the Customer registration process collects ⟨credit card number⟩ and customer information (name, email, and physical address) that we simply refer to as ⟨customer⟩, but it does not itself use these. They are instead used by the Purchase, Targeted marketing, and Mass marketing processes. This disconnect mirrors a challenge faced by many companies: whereas the individual processes within the company are usually well-understood by the staff undertaking them, including the interfaces to other processes, the global picture of *all* processes in the company is rarely well-understood. But the GDPR requires such a global understanding: data collected in one process may migrate to other processes, and end-user consent is required for *all* involved processes.[1]

We propose that for contemporary process models to be truly useful for GDPR analysis, we must interpret *collections* of processes as data-flow graphs. We therefore introduce the following simple model of process collections.

Definition 4.1 (process collection). *A* process collection PC *is a tuple* $PC = (P, D, DU, DC)$ *comprising:*

1. *a set* P *of processes,*
2. *a set of data classes* D*,*
3. *a relation* $DC \subseteq D \times P$ *specifying what data is* collected *by which processes, and*
4. *a relation* $DU \subseteq D \times P$ *specifying what data is* used *by which processes.*

Note that the set D of data classes is not a set of data values per se, but rather a set of the possible kinds of data, e.g., addresses, credit card numbers, etc.

Example 4.2. We construct a process collection modelling the example from Sect. 3 by lifting the informal description of the example to the process collection QC given formally in Fig. 2 and represented visually in Fig. 3. In the

[1] Notice that without the anchor of processes-as-purposes, this problem is hardly solvable in practice. For example, what are the purposes the user consents to for the hundreds of computer systems running at a large corporation?

$$
\begin{aligned}
P &= \{\mathsf{Purchase}, \mathsf{Register\ Customer}, \mathsf{Targeted\ Marketing}, \mathsf{Mass\ Marketing}\} \\
D &= \{\langle\text{customer}\rangle, \langle\text{credit card number}\rangle, \langle\text{order}\rangle, \langle\text{profile}\rangle\} \\
DU &= [\mathsf{Purchase} \mapsto \{\langle\text{customer}\rangle, \langle\text{credit card number}\rangle\}; \\
&\quad\ \mathsf{Register\ Customer} \mapsto \emptyset; \\
&\quad\ \mathsf{Targeted\ Marketing} \mapsto \{\langle\text{customer}\rangle, \langle\text{order}\rangle, \langle\text{profile}\rangle\}; \\
&\quad\ \mathsf{Mass\ Marketing} \mapsto \{\langle\text{customer}\rangle\}] \\
DC &= [\mathsf{Purchase} \mapsto \{\langle\text{order}\rangle\}; \\
&\quad\ \mathsf{Register\ Customer} \mapsto \{\langle\text{customer}\rangle, \langle\text{credit card number}\rangle\}; \\
&\quad\ \mathsf{Targeted\ Marketing} \mapsto \{\langle\text{profile}\rangle\}; \\
&\quad\ \mathsf{Mass\ Marketing} \mapsto \emptyset]
\end{aligned}
$$

Fig. 2. The use of personal data in an online retailer: process collection $QC = (P, D, DU, DC)$ corresponding to the example of Sect. 3. To conserve space, the relations DU and DC have been represented as maps.

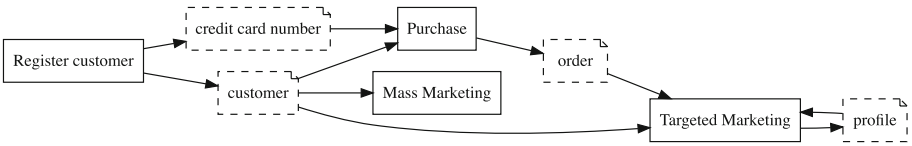


Fig. 3. Graphical representation of the process collection QC of Fig. 2.

latter figure, the processes of the retailer are rendered as square boxes labelled Purchase, Register Customer, Targeted Marketing, and Mass Marketing. Data is written as dashed, dog-eared boxes, e.g., ⟨customer⟩, ⟨credit card number⟩, and ⟨order⟩. Data use and collection is indicated by arrows between process boxes and data boxes:

1. An arrow from a process to data indicates that the process collects and stores the given data, e.g., the Register Customer process records the new customer's contact information (name and address) in the data class ⟨customer⟩.
2. An arrow from data to a process indicates that the process uses the given data, e.g., the Purchase process uses the customer's contact information in ⟨customer⟩ and the payment information in ⟨credit card number⟩.

For example, the Targeted Marketing process uses the order data ⟨order⟩ and produces the personal ⟨profile⟩ of the customer. The Mass Marketing process similarly uses the customer data, but it does not use the orders.

In this example, we derived the process collection from the informal description of the on-line retailer. In general, process collections can be extracted (even automatically) from formal process models such as the BPMN diagrams of Fig. 1.

Example 4.3. Consider the BPMN models in Fig. 1. The set P of processes is the set of labels of lanes. The set of data classes D is the set of labels of database access lines. The relations DC and DU are given by dashed lines from processes to databases (DC) and databases to processes (DU).

We can use the data production and usage relations to derive which user consents are needed. For example, the on-line retailer must acquire consent to use the customer data for future purchases and mass marketing. If we know when processes can no longer be started, we can also use the relations to infer when data must be deleted or made non-personal, for example by anonymising it. We shall pursue this idea further in Sect. 5.3.

4.4 A Methodology for Auditing for the GDPR

We now propose a methodology for auditing for the GDPR. Our methodology has the following inputs. First, at the lowest level, we require an *implementation*, say, written in Java, of the system under consideration. Second, we require *a collection of process models* describing the system's behaviour , from which we can produce a formal process collection (Definition 4.1). Finally, we require a user-facing *privacy policy*. Recall from Sect. 2 (transparency, consent) that this is required by the GDPR.

To establish GDPR compliance, we must show the following:

1. The implementation conforms to the process collection. That is, the implementation implements the processes described in the process collection.
2. The process collection conforms to the privacy policy. That is, the processes described actually treat data in the manner described by the privacy policy.
3. The process collection conforms to the GDPR. That is, the processes described follow the GDPR, for example they delete data as appropriate.
4. The privacy policy conforms to the GDPR. That is, the privacy policy does not make statements outside the GDPR, such as "we collect your personal information and use it for undisclosed purposes."

We illustrate the required conformance relations in Fig. 4.

Following this methodology ensures that the purpose limitation is upheld because the implementation collects and uses data as specified by the process collection (1); the process collection uses data as specified in the privacy policy (2); and both the privacy policy and process collection conform to the GDPR (3,4).

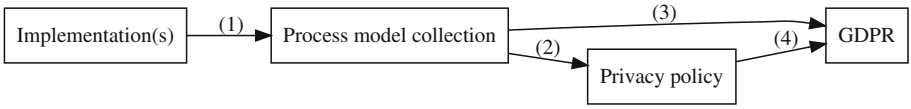


Fig. 4. Conformance requirements. An arrow $A \longrightarrow RHDB$ means "A conforms to B".

The difficulty of these steps depend on the exact nature of the implementation. For example, for Step (1), if the implementation has a collection of BPMN models as its specification, the compliance of that collection to an implementation might be reasonably assumed or spot-checked by an auditor. If the implementation is based on a BPMN process engine or a Statechart interpreter [11], the process might be automated. Alternatively, a process collection might be obtained from an informal requirements specification (indicating processes) and a dataflow analysis [16] of a mainstream programming-language implementation (establishing collection and use).

In the latter case, where an informal requirements specification is the starting point, Step (1) involves establishing that a process collection is a dataflow model of a program. This problem is in general undecidable, so this step entails approximating dataflow. But what kind of approximation do we need? An under-approximation would leave DU smaller than it really is: Some pair (d, p) is not in DU even though the system uses data d for the purpose p, in violation of GDPR consent requirements. Conversely, an over-approximation would leave DU larger than it really is: Some pair (d, p) is in DU even though the system does not use data d for the purpose p, in violation of the purpose limitation.

These observations point to a curious problem at the intersection between computer science and law: if static analysis cannot determine whether data will or will not be used, is there a violation of the purpose limitation? Here, we take the pragmatic solution that the inclusion of a data usage in a process collection means the *possible* use of that data, and leave for the human audit to verify that this usage may indeed happen in the implemented process.

Finally, we note that if we omitted the process-model middleman, some other means would be required to relate purposes to implementation artefacts.

5 Algorithmic Verification of Compliance

We demonstrate in this section how parts of our methodology for demonstrating GDPR compliance can be supported or even achieved algorithmically.

5.1 Consent Statements

We saw in Sect. 2 that the GDPR requires consent to the collection of data for specific purposes. We also saw how current privacy policies do not support this, primarily by failing to distinguish between different kinds of data and the purposes they are used for. For example, because Mass Marketing uses ⟨customer⟩, a privacy policy compliant with the GDPR must include words to the effect "we use your customer contact information (name and address) for mass marketing," indicating for what purpose the customers' contact information is used.

We now show how conformance of a process collection to a privacy policy (Part 2 of Fig. 4) can be decided algorithmically. Recall that the DU component of a process collection PC comprises a set of pairs (d, p), where d is a class of data and p is a process using that data. Assuming that PC adequately models

the underlying processes, that is, DU comprises exactly the data used by the processes, we can automatically generate the corresponding privacy policy.

Definition 5.1 (Data purpose). *Let $PC = (P, D, DU, DC)$ be a process collection. A* data purpose *DP is a relation $DP \subseteq D \times P$. For a data purpose DP, the* privacy policy $\mathsf{pp}(DP)$ *is the set of statements*

"We use d for p,"

for each $(d, p) \in DP$.

That is, a data purpose associates data with processes. Note that the above definition may associate the same data with *multiple* uses, e.g., users would typically be presented with a consent statement like "we collect d_1, d_2 for purpose p_1 and we collect d_1, d_3, d_4 for purpose p_2."

d	p
⟨customer⟩	Purchase
⟨credit card number⟩	Purchase
⟨customer⟩	Mass Marketing
⟨profile⟩	Targeted Marketing
⟨order⟩	Targeted Marketing
⟨customer⟩	Targeted Marketing

Fig. 5. Privacy policy $\mathsf{pp}(DU)$ for the process collection $QC = (P, D, DU, DC)$ of Fig. 2.

Example 5.2. Let $QC = (P, D, DU, DC)$ be the process model of Fig. 2. Then DU comprises the pairs in Fig. 5. Using Definition 5.1, and allowing meaning-preserving natural language transformations, the Targeted Marketing privacy policy reads: "We collect your customer information (name, address), order history, and profile, and use them to send you targeted advertising."

The notion of data purpose (Definition 5.1) naturally orders the possible data purposes by set inclusion.

Lemma 5.3. *Let D be a universe of data and P a collection of processes. Then the possible data purposes form a lattice under the subset-relation, i.e., $DP \sqsubseteq DP'$ iff $DP \subseteq DP'$.*

Proof. Immediate from Definition 5.1

The lattice ordering provides a means of formalising privacy policies where users give consent to some, but not all, purposes supported by the system. For a process collection $PC = (P, D, DU, DC)$, DU is the maximal data purpose;

asking anything more is in effect asking users for permission to data that the underlying processes do not actually use, violating purpose limitation.

Returning to the present setting where users must consent to all purposes, we note that DU is in fact also the *least* admissible data purpose: If users consent to strictly less than DU, there must be a pair $(d, p) \in DU$ the user did not consent to. Hence the system violates the GDPR requirements on obtaining consent.

Besides the outright generation of privacy policies, we can also use this observation to check an existing privacy policy for correctness: Simply extract from the policy the appropriate set of pairs $\{(d_1, p_1), (d_2, p_2), (d_3, p_2)\}$ and compare it with DU.

5.2 Data Minimisation

In the last section, we generated data purpose statements algorithmically from process models. However, these statements are GDPR compliant only if they mention all the data *used* by the process. As we saw in Sect. 2, the GDPR also requires that all of the data collected is *necessary* for the stated purpose.

We caution that necessity is a slippery concept. For example, one may ask whether an online merchant really *needs* my credit card number given that sending an invoice might satisfy the same purpose of collecting payment. We shall leave such fine distinctions for the auditors.

We can however determine algorithmically some classes of *un*necessary data: we can check whether data that has been collected is in fact also used. If not, that data is clearly unnecessary, violating data minimisation. This information will help a human auditor quickly judge the conformance arrow (3) in Fig. 4.

Definition 5.4 (Used data). *Let $PC = (P, D, DU, DC)$ be a process collection and let $d \in D$ be data for PC. We say that d is* used *iff for some $p \in P$ we have $(d, p) \in DU$.*

In other words, d is "used" if some process uses it according DU.

Example 5.5. Returning to the process collection QC of Figs. 2 and 3, it is straightforward to verify that no collected data is unused. However, if we did not have the Targeted Marketing process, then ⟨profile⟩ would not be present at all and ⟨order⟩ would not be "used" in the sense of Definition 5.4. Consequently, either the order data could not be legally stored in an order data base or the process collection is incomplete.

We remark that the data used is computable in time polynomial in the size of $\mathcal{L}(PC)$ under reasonable assumptions about representation:

Proposition 5.6. *Let $PC = (P, D, DU, DC)$ be a process such that P and DU are finite, and assume that P and DU are represented as sequences of their elements. Then computing whether any d of D is used is possible in time polynomial in the sizes of P and DU.*

Proof. Let $d \in D$ be given. Observe that d is used iff for some $(d', p) \in DU$ we have $d' = d$. Given a pair (d', p), we can determine in time $\mathcal{O}(|d|)$ whether $d' = d$. We can then compute whether d is used by iterating over the elements of DU in time $\mathcal{O}(|d| * |DU|)$.

5.3 Deletion

We saw in Sect. 2 that the GDPR, via the right to be forgotten, requires that data must be deleted on request, provided that the purposes for which consent has been given no longer apply. Since we have identified purposes and processes, this would be when either (i) no currently running process uses the data, and (ii) no process that may be started in the future uses the data.

For practical purposes, determining the set of (non-)applicable processes is often straightforward. In many web-services, the set of applicable processes is all or nothing: all the services are offered until the user deletes his account, at which point no services are offered.

Example 5.7. In our running example of an on-line retailer, we assume that consent is given before the customer inputs personal data, i.e. in the first activity in the Register customer process given in Fig. 1. After a customer is registered, purchases and marketing may be started indefinitely. Implicit in our process collection model is that once the user deletes his account then no more processes can be started (equivalently, no more purposes can be activated) and the user's data must now be deleted.

5.4 Human Verification of Compliance

We have seen in the preceding subsections that some aspects of GDPR compliance can be verified algorithmically. However, in Sect. 4.1 we explained that other aspects cannot: We cannot distinguish algorithmically between, e.g., marketing messages and political propaganda. To enforce the purpose limitation in such cases it may be necessary to add *human* enforcement activities.

We saw an example already in the BPMN processes given in Fig. 1: the Mass Marketing process includes a human activity "Approve advertisement text", whereby an authorised staff member confirms that the proposed advertisement text is in fact an advertisement.

This ability to model both automated and human activities is unique to business process models. This makes them particularly well-suited for the analysis of GDPR compliance: A model of only the computer systems cannot account for the necessary human activities.

6 Related Work

Purpose-based access control [3,4] proposes an access control mechanism for databases where each data item has an associated intended purpose. To access

the item, a user must state his access purpose. Both kinds of purposes are arranged in hierarchies, and a notion of compatibility of purpose is defined. The access control mechanism itself is essentially role-based access control (RBAC). Similar ideas form the basis of a formal language for specifying purpose-based access control policies in [30] and for deriving formal invariants and proof obligations from a formal specification of such policies. The idea of matching a stated with an intended purpose is also pursued in [24]. Finally, similar to our work, [26] proposes identifying purposes and business processes. The authors use knowledge of the current task within a process for access control decisions. In contrast to all of these works, our focus is not on designing access control mechanisms, but rather audit and compliance.

Privacy-aware role based access control [20] proposes extending RBAC with a hierarchical notion of purpose to model privacy policies, emphasizing conflict detection in the resulting formal models. Similarly, *Purpose-Aware Role-Based Access Control* [18] extends RBAC with an explicit notion of purpose, in part to alleviate technical difficulties in expressing privacy policies in RBAC. In [7], the authors propose an access control mechanism supporting conditional purpose, allowed purpose, and prohibited purpose, each of which is defined through dynamic roles, with the actual intended purpose computed dynamically. The paper emphasizes balancing privacy concerns against data mining opportunities. In [17], the authors use information-flow labels to specify and enforce purpose-based access control policies. They argue that information-flow diagrams are well-suited to express and reason about purpose-based privacy policies. This is very much akin to the use of process collections in the present paper.

All the above works associate data with purpose, an idea we have taken up in our notion of process collections. However, the previous works proceed to give methods for access control under various circumstances. In contrast, our work is concerned with the questions: what are the appropriate purposes in the first place, and is (data) access required for these purposes? Moreover, these other methods are invariably *automated*, whereas we have emphasized that the GDPR also requires human activities to ensure compliance (see Sect. 5.4).

Finally, [15] gives a semantic model of purposes themselves, stipulating that "the purpose of an action is determined by its situation among other interrelated actions." The authors model actions and their relations in an *action graph*, and develop a modal logic and model-checking algorithm for verifying purpose-based policies. This work is akin to the present one in that it addresses the question "how do we find purposes?" However, the present paper does not attempt a semantic analysis of purpose, and leverages instead the observation that practitioners have already defined purposes via business process modelling.

7 Discussion and Conclusion

We investigated the GDPR and we showed how a mechanism for relating *purpose* to implementation artefacts is necessary to demonstrate compliance. To remedy the problem that purpose is usually not represented explicitly in implementations

of computer systems, we put forward the idea of *identifying* a business process and a purpose. We proposed a methodology where GDPR compliance is decomposed into the compliance of an implementation to an interprocess dataflow model, of the dataflow model to a privacy policy, and of these latter two models to the GDPR. We demonstrated that given a model of interprocess dataflow, we may verify compliance of certain aspects of the GDPR algorithmically. In particular, we can generate privacy policies automatically from the model and detect violations of data minimisation. Finally, we explained why GDPR compliance cannot be entirely automated and the role of humans in enforcement.

Discussion. We return now to the question of data deletion. Recall from Sect. 5.3 that data should be deleted once the purposes for which it is used can no longer apply, that is, when the corresponding processes can no longer be started. Providing such a fine-grained account of deletion requires modelling when processes start using a *process lifecycle model*. We provide an illustrative example here, leaving the full development of this idea to future work.

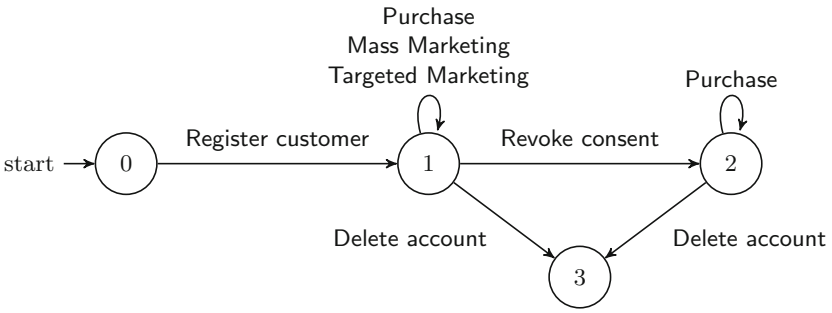


Fig. 6. Lifecycle model for the on-line retailer processes.

Our example is the model in Fig. 6, which models the lifecycle of the processes in our running example. We have added processes here for deleting an account and revoking consent. This model is a finite state machine where states distinguish what processes can be started and transitions are processes started. Some processes, such as Purchase, do not change the current state. Other processes lead to state changes, such as the new processes Revoke consent and Delete account. In a given state, the set of processes that may yet start is the set of reachable transitions. For example, in state 0, it is all the processes; in state 1 it is all but Register customer; in state 2 it is only Purchase and Delete account; and in state 3, we may not start any processes. From this information, we can compute what data we must delete. For example, in the transition from state 1 to 2 we lose the ability to start Targeted marketing, which is the only purpose for storing the ⟨profile⟩ data. It follows that immediately after this transition, we must delete that data.

Future Work. Another important area for future work concerns data transfers to third parties. The GDPR has precise rules about who may transfer data to other parties, when these transfers can occur, and under what circumstances other parties can or must delete, produce, or store data. Naturally, this opens up questions about audits and compliance similar to the ones addressed in this paper. Moreover, enforcement in this setting is closely related to research on distributed usage control [27,28] and on executable process models [6,29]. Other relevant future work includes how to distinguish between personally identifiable information and other information, and handling systems that allow users to consent to some, but not all, purposes.

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: towards a truly declarative service flow language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006). https://doi.org/10.1007/11841197_1
2. BPMN Technical Committee: Business process model and notation (BPMN). Technical Report formal/2011-01-03, Object Management Group. Version 2.0, January 2011
3. Byun, J.-W., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, pp. 102–110. ACM (2005)
4. Byun, J.-W., Li, N.: Purpose based access control for privacy protection in relational database systems. VLDB J. **17**(4), 603–619 (2008)
5. Davenport, T.H.: Process Innovation: Reengineering Work Through Information Technology. Harvard Business Press, Boston (1993)
6. Debois, S., Hildebrandt, T., Slaats, T.: Concurrency and asynchrony in declarative workflows. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 72–89. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_5
7. Enamul Kabir, Md., Wang, H., Bertino, E.: A conditional purpose-based access control model with dynamic roles. Expert Syst. Appl. **38**(3), 1482–1489 (2011)
8. Facebook Data Policy. https://www.facebook.com/policy.php. Accessed 9 Aug 2017
9. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Off. J. Eur. Union, **L119**, 1–88 (2016)
10. Google Privacy Policy. https://www.google.com/policies/privacy/. Accessed 9 Aug 2017
11. Harel, D., Politi, M.: Modeling Reactive Systems with Statecharts: The Statemate Approach, 1st edn. McGraw-Hill Inc., New York (1998)
12. Hildebrandt, T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Post-Proceedings of PLACES 2010, EPTCS, vol. 69, pp. 59–73 (2010)

13. Hull, R., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_1
14. IBM Privacy Policy. https://www.ibm.com/privacy/us/en/. Accessed 9 Aug 2017
15. Jafari, M., Fong, P.W.L., Safavi-Naini, R., Barker, K., Sheppard, N.P.: Towards defining semantic foundations for purpose-based privacy policies. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy, CODASPY 2011, pp. 213–224. ACM, New York (2011)
16. Knoop, J., Rüthing, O., Steffen, B.: Towards a tool kit for the automatic generation of interprocedural data flow analyses. J. Prog. Lang. **4**(4), 211–246 (1996)
17. Kumar, N.V.N., Shyamasundar, R.K.: Realizing purpose-based privacy policies succinctly via information-flow labels. In: 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, pp. 753–760, December 2014
18. Masoumzadeh, A., Joshi, J.B.D.: PuRBAC: purpose-aware role-based access control. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1104–1121. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_12
19. Mukkamala, R.R.: A formal model for declarative workflows: dynamic condition response graphs. Ph.D. thesis, IT University of Copenhagen (2012)
20. Ni, Q., et al.: Privacy-aware role-based access control. ACM Trans. Inf. Syst. Secur. **13**(3), 24:1–24:31 (2010)
21. Object Management Group: Unified modeling language: superstructure. Technical report formal/05-07-04, Object Management Group, Version 2.0 (2005)
22. Object Management Group: Unified modeling language: infrastructure. Technical report formal/05-07-05, Object Management Group, Version 2.0, March 2006
23. Object Management Group: Case management model and notation. Technical report formal/2014-05-05, Object Management Group, Version 1.0, May 2014
24. Peng, H., Gu, J., Ye, X.: Dynamic purpose-based access control. In: 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 695–700, December 2008
25. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, pp. 287–300. IEEE (2007)
26. Petković, M., Prandi, D., Zannone, N.: Purpose control: did you process the data for the intended purpose? In: Jonker, W., Petković, M. (eds.) SDM 2011. LNCS, vol. 6933, pp. 145–168. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23556-6_10
27. Pretschner, A., Hilty, M., Basin, D.: Distributed usage control. Commun. ACM **49**(9), 39–44 (2006)
28. Pretschner, A., Hilty, M., Basin, D., Schaefer, C., Walter, T.: Mechanisms for usage control. In: ASIACCS 2008: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, pp. 240–244. ACM (2008)
29. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 329–347. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_19
30. Yang, N., Barringer, H., Zhang, N.: A purpose-based access control model. In: Third International Symposium on Information Assurance and Security, pp. 143–148, August 2007

MP3: A More Efficient Private Presence Protocol

Rahul Parhi, Michael Schliep, and Nicholas Hopper(✉)

University of Minnesota, Minneapolis, MN, USA
{parhi003,schli116,hoppernj}@umn.edu

Abstract. This paper presents a novel private presence protocol, referred to as MP3, where the service provider does not have any knowledge of the social graph of its users. In prior work, a private presence protocol, referred to as DP5, was presented. However, the size of the presence database in this protocol grows rapidly as the number of users increases; this limits its scalability and increases its cost. In the proposed protocol, the size of the presence database is reduced significantly, enabling significantly cheaper registration and lookup compared to that of DP5. MP3 requires about two-thirds the bandwidth of DP5 for $N = 200\,000$ users and about one-half the bandwidth of DP5 for $N = 1\,000\,000$ users. Furthermore, these savings grow asymptotically with the number of users. Additionally, the client-facing latency in MP3 is an order of magnitude less than that of DP5. We provide an evaluation of the performance and scalability of both protocols.

1 Introduction

Due to the recent threat and concern of government surveillance and collection of user data [1], an increasing number of services have emerged with the goal of protecting users' privacy from the provider of the service. A common approach is end-to-end secure messaging, which is currently employed in services such as Apple's iMessage, Open Whisper Systems' Signal Protocol, and Facebook's Messenger [2–4]. Secure messaging hides the content of the conversation from the provider by using strong cryptographic techniques, but the metadata of the conversation is still known to the service provider.

A critical part of secure messaging is presence, i.e., knowing when a friend is online. Although secure messaging providers do not have access to the plaintext conversation, they still have access to the set of friends of every user. This means that these services know the social graph of their users as well as the presence status of every user at any given time. To have a truly private communication platform, the metadata of the communication must also be protected.

An existing solution to this problem is DP5—the Daghstul Privacy Preserving Presence Protocol P—proposed by Borisov, Danezis, and Goldberg [5]. DP5 is the first private presence protocol to leak no information about the social graph to third parties and limit the information retained by the service itself.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 38–57, 2018.
https://doi.org/10.1007/978-3-662-58387-6_3

DP5 allows its users to see the online status of their friends in a completely private manner. It utilizes Private Information Retrieval (PIR) [6] for querying the service for a given user's buddy's presence.

The major weakness of DP5 is its *lack of* scalability for a large number of users. The presence database of DP5 grows much more rapidly than the number of users of the service. This results in a very expensive service for even a small number of users. To overcome this bottleneck, we propose MP3—the Minnesota Private Presence Protocol. MP3 maintains the same security goals as DP5. This is elaborated in Sects. 2.3 and 3.7. Compared to DP5, MP3 assumes that revoking and unrevoking friends is uncommon and thus we are able to significantly reduce the size of the presence database by using a *dynamic broadcast encryption scheme* [7]. As a result, MP3 is significantly cheaper to run for even a relatively modest user base. Additionally, these savings increase as the number of users increase. The *key contribution* of this paper lies in significantly reducing the size of the presence database compared to DP5; this allows cheaper registration and lookup queries in the context of the bandwidth required. The client-facing latency of MP3 is also an order of magnitude less than that of DP5 due to the smaller presence database.

This paper is organized as follows. Section 2 describes the background, goals, and related work pertaining to the MP3 protocol. Section 3 presents a detailed description of the MP3 protocol. Section 4 analyzes the performance of MP3 and compares it to that of DP5.

2 Background

The primary functionality of a private presence protocol is to allow for the registration of one's online presence and to allow for the query of the presence status of one's buddies in a completely private manner.

2.1 DP5 Overview

Since our design shares many characteristics with DP5, we give a brief overview of the protocol here. DP5 uses Private Information Retrieval (PIR), in which a database is distributed to several servers so that a user can query the servers to retrieve a specific record without revealing which record they retrieve. Given this functionality, a "trivial" private presence protocol would have each user A with n_A friends encrypt n_A *presence records* recording their status (and possibly other information, such as a contact address), with a shared key for each friend, and periodically upload this information to a presence database. When A's friend B wants to check on A's status, they would query the current database (using PIR) for the presence records encrypted under the symmetric key A shares with B. To hide information about the social graph, each user would need to pad the number of presence records uploaded per period to some maximum value denoted by N_{fmax}. This protocol results in a nearly quadratic-size database in the number of users. Moreover, the server-side computational costs of PIR scale

linearly in the size of the database (and the bandwidth costs also increase, though sub-linearly).

To combat this inefficiency, DP5 splits the presence service into two asymmetric services. The primary, short-term, service is used to register and query presence of users with the precision of short windows (on the order of minutes). The secondary service is the long-term service, which is used to provide metadata for querying during the short-term. The long-term service also provides friend registration and revocation with the precision of long windows (on the order of days). As in the "trivial" protocol, it is assumed that users share a unique secret with each of their friends. Additionally, in order to not leak information about how many friends a given user has, DP5 defines a limit of N_{fmax} as the maximum number of friends a user may have.

In each period of the long-term service, a user (let's say Alice) of DP5 will upload her presence to the registration mechanism of DP5. This is referred to as Alice's long-term presence record. This record is actually *several* records, one for each of Alice's friends, padded with random records upto N_{fmax}. If, N_{fmax} is on the order of the number of users of the service, then the long-term presence database scales quadratically with the number of users, which in turn increases the amount of bandwidth and CPU this service requires. These long-term records contain information used by her friends to identify her during the short-term period. Then, during the short-term period, Alice uploads a single record to the short-term presence database in each short-term period she is online. Additionally, a *single* record containing a signature is uploaded to an auditable *signature database* during every short-term period. Thus, the short-term service, which is queried more often, grows only linearly with the number of online users.

To improve the DP5 protocol, we address the issue of scaling in the long-term service of DP5 by reducing the number of records Alice uploads in each long-term period to only a relatively constant number of records, yielding performance closer to that of the short-term service. We leave the short-term service of DP5 unchanged as it is already quite cheap.

2.2 Threat Model

We make standard assumptions about the users and adversaries of MP3. They are real world adversaries with common capabilities.

- We assume that honest users' end systems are secure and not compromised. We also assume that honest servers can maintain secrecy and integrity. Our design maintains forward security and does not require servers to store any long-term secrets.
- We allow the adversary to be an observer or a dishonest user of the system, and we assume they have not made any recent breakthrough in computational cryptographic assumptions, and assume that they cannot distinguish between different ciphertexts. More detailed assumptions are described in the protocol description in Sect. 3.

- Our security properties are under the covert model, i.e., adversaries will not act dishonestly if it would cause them to be detected and identified.
- Our protocol maintains availability against malicious parties. This is further detailed in Appendix B.

2.3 Security Goals

Here, we describe the goals required for a private presence service. Our goals are the same as the goals of DP5.

Privacy of Presence and Auxiliary Data. The presence status of a user and their auxiliary data should be available to only that user's explicit friends.

Integrity of Authentication and Auxiliary Data. The friends of Alice should not accept the presence and auxiliary data unless it was submitted by Alice.

Unlinkability of Sessions. It should be infeasible for a user to be linked between multiple uses of the service. The infrastructure and non-friend users should not be able to link the presence of another between epochs.

Privacy of the Social Graph. No information about the social graph of a user should be revealed to any other party of the service. More specifically, friends should not learn about other friends and the infrastructure should not learn any new information about a user.

Forward/Backward Secrecy of the Infrastructure. Any compromised keys stored in the infrastructure servers should not reveal past or future information that is secured with previous or future keys.

Auditability. All operations performed by the infrastructure should be auditable. A user should detect when their friend registration or presence registration has not been performed honestly by the service provider.

Support for Anonymous Channels. The protocol should not require any identifying information for operation. The use of an anonymous channel should only enhance the privacy of the system and the service will not compromise the anonymity of the user.

Indistinguishability of Revocation, Suspension, and Offline Status. A user is revoked if they are no longer able to query the presence status of the buddy that revoked them. A suspension is a temporary revocation, i.e., for some period of time, a user cannot query the presence status of the buddy that suspended them. This means a suspended buddy can be "unrevoked." Revocations and suspensions should not be distinguishable from being offline. For example, if Bob's buddy Alice appeared to be offline, Bob would not know if he was revoked

or suspended, or if Alice was genuinely offline. MP3 provides *plausibly deniable* revocation and suspension of buddies. Plausibly deniable revocation means the transcript does not prove a user has been revoked. However, if a user does not see their friend come online for an extended period of time they may begin to assume they have been revoked. This is discussed in further detail in Sect. 3.7.

2.4 Related Work

DP5 seems to have been the first and only previous work to address social graph privacy in the context of presence services. Similar, but different, related work includes Dissent [8] and Riposte [9] which offer anonymous micro-blogging services; these systems are similar to private presence in that posting a micro-blog implies the author was online. Dissent is based on a DC-net with a client-server architecture. Clients in Dissent must form a group to post anonymous messages for each other using distrusted servers. Dissent provides anonymity within a static group. Riposte utilizes a novel private database writing mechanism based on techniques of PIR. Both of these systems have high latency when dealing with large anonymity sets and are not concerned with social graph privacy.

Two other similar and relevant anonymous messaging/microblogging systems that build on PIR techniques include Riffle [10] and Pung [11]. These systems allow for a user to send a message to their friends without revealing the social graph of the users. These messages could be used to indicate presence. However, these two systems assume every user uploads a message during every epoch. This implies that all users must be present at all times which is unrealistic and negates the need for a private presence protocol.

3 The MP3 Protocol

3.1 Overview

In this section, we provide an overview of MP3. MP3 is composed of two databases, a long-term database and a short-term database. The short-term database contains the presence status and information of a user. A new short-term database is generated on the order of minutes (5 min), we refer to these as short-term epochs. The long-term database contains information for a user to identify their friends' short-term database entries. A new long-term database is generated less frequently (once every 24 h), these are referred to as long-term epochs. Alice uploads her presence at most once for every long-term and short-term epoch. When Bob wants to check if his friend Alice is online, he first queries the long-term database and retrieves Alice's entry. Then he computes her short-term identifier and queries the short-term database for her presence. Each long-term database entry of a user contains information for looking up the next long-term database entry of that user. Alice may be unable to upload every long-term epoch so MP3 keeps the most recent long-term databases corresponding to 30 days.

The database entries are simple ⟨key, value⟩ pairs where the key is a unique identifier for a user in that specific epoch. These databases are queried using a hash-based PIR protocol of retrieving ⟨key, value⟩ pairs. Since these databases are queried with PIR, the infrastructure does not learn any information about a user's queries. In our implementation users upload their database entries to a single registration server and use a distributed PIR protocol with N_{lookup} servers for database queries. Similar to DP5, when querying the long-term database the users must query all long-term databases to avoid revealing which database contained the entry they needed.

The short-term database contains a single entry (presence message) per online user. For Alice (a) we denote presence message as $m_a(i)$ during short-term epoch t_i. This presence message may contain information such as how to contact Alice or a public key. If $m_a(i)$ is not present in during t_i, Alice is assumed to not be online.

MP3 uses a Dynamic Broadcast Encryption (DBE) [7] scheme for long-term database entries. DBE allows Alice to create a single constant-sized ciphertext that can be decrypted by all of her friends. Before participating in MP3, Alice generates a single DBE encryption key (mk) and a decryption key (dk_{af}) for each friend (f). During long-term epoch T_j Alice creates a DBE ciphertext with her short-term identity information for all short-term epochs that occur during T_j.

The dynamic part of DBE allows Alice to revoke decryption keys so the revoked keys cannot decrypt future ciphertexts. We utilize this to allow Alice to revoke up to N_{rev} friends in each long-term epoch. To provide plausible deniability of revocation, Alice distributes new decryption keys to the revoked friends. If Alice wants to truly revoke the friend she gives them a new randomly generated decryption key. If they are revoked for deniability reasons she gives them a new correctly generated decryption key. For the rest of the paper DBE.Revoke is used to refer to DBE revocations and MP3.Revoke is used to refer to Alice actually revoking a friend.

To unrevoke a previously MP3.Revoked friend, Alice constructs a special long-term database entry using the revoked friend's random decryption key. This record DBE.Revokes the random key and issues a new valid decryption key. This special entry must maintain N_{rev} DBE revocations to be indistinguishable from a regular entry. MP3 allows N_{unrev} friends to be unrevoked during a single long-term epoch. This means Alice uploads $N_{\text{unrev}} + 1$ entries, each of N_{rev} size for a long-term epoch. One entry to distribute short-term lookup information to her friends and possibly revoke N_{rev} friends. And N_{unrev} entries that allow her to unrevoke friends.

Finally, to prevent forged records, we employ two signature schemes, one for long-term presence records and another for short-term presence records. Thus, all of Alice's friends can be confident that the record they received from MP3's lookup mechanism can only belong to Alice. We use Elliptic Curve Digital Signature Algorithm (ECDSA) [12] for long-term epochs and Boneh-Lynn-Shacham (BLS) [13] signature scheme for short-term epochs. Moreover, an auditable *signature database* is employed during each short-term epoch.

3.2 Cryptographic Primitives

It is assumed that everyone participating in MP3 shares a set of known cryptographic primitives. We assume three primes $p_{dbe}, p_{dsa}, p_{bls}$, for simplicity we omit the subscripts when it is clear which prime we are discussing. Let G_1 and G_2 be two cyclic groups of prime order p_{dbe} and let G_T be a cyclic group also of prime order p_{dbe}. Denote $\mathbb{Z}_p$ as the ring of integers modulo p and denote $\mathbb{Z}_p^\times$ as the set of units in $\mathbb{Z}_p$, also denote $g \leftarrow G$ as randomly selecting an element g from a set G. An efficiently computable asymmetric pairing defined by the map $e : G_1 \times G_2 \rightarrow G_T$ is known so that for generators $g_1 \in G_1$, $g_2 \in G_2$ and for all $u, v \in \mathbb{Z}_p$

$$e(g_1^u, g_2^v) = e(g_1, g_2)^{uv}$$

The decisional Diffie-Hellman problem and the decisional co-Diffie-Hellman problem are assumed hard for G_1 and G_2. It is also assumed that our pairing is non-degenerate.

MP3 utilizes the following:

- ECDSA [12] signature scheme with group G_{dsa} of prime order p_{dsa} with generator g_{dsa}.
- BLS signature scheme [13] with groups G_3, G_4, G_5 of prime order p_{bls} with g_3 generating G_3 and an efficiently computable asymmetric pairing defined by the map $e_{bls} : G_3 \times G_4 \rightarrow G_5$.
- PRF_K, a keyed pseudorandom function that maps a short-term epoch timestamp to keys for AEAD described below.
- H_1, an efficiently computable hash function that maps the concatenation of the byte representations of long-term epoch timestamps and elements of G_T to elements of $\mathbb{Z}_p^\times$.
- H_2, an efficiently computable hash function that maps long-term public key to shared identifiers.
- H_3, an efficiently computable hash function that maps elements of G_T to elements of $\mathbb{Z}_p^\times$.
- H_4, an efficiently computable hash function that maps elements of G_1 to keys in the pseudorandom function above.
- H_5, an efficiently computable hash function that maps short-term epoch timestamps to elements of G_4.
- H_6, an efficiently computable hash function that maps elements of G_T to shared identifiers.
- $\mathrm{AEAD}_K^{\mathrm{IV}}(m)$, an authenticated encryption function where m is the message, K is the key, and IV is the initialization vector.

3.3 Dynamic Broadcast Encryption

In our construction of MP3, long-term epochs share many characteristics with a broadcast encryption scheme. In the context of MP3's long-term epoch, we use a *dynamic broadcast encryption* scheme with constant-size ciphertexts and decryption keys.

The definition of a Dynamic Broadcast Encryption (DBE) [7] is slightly different from a conventional broadcast encryption scheme in that it involves *two* authoritative parties: a *group-manager* and a *broadcaster*. The job of the group manager is to grant new users access to the group. The job of the broadcaster is to encrypt and transport messages to this group of users. When a message is encrypted, some members of this group can be revoked temporarily (suspended) or permanently (revoked) from decrypting the broadcasted message. Formally, a DBE scheme with revocation is a tuple of probabilistic algorithms (Setup, Join, Encrypt, Decrypt, Revoke, Update). These algorithms rely on an asymmetric pairing as described in Sect. 3.2.

Our design relies on the DBE scheme proposed by Delerablée et al. [7]. In our use case, every user is both a group-manager and a broadcaster. We introduce two additional operations (ShiftMK, ShiftDK) to support MP3's plausibly deniable revocation and suspension. Our modifications are detailed in Appendix A. We provide the relevant components of DBE as it pertains to MP3 below:

- DBE.Setup() - generates a *manager key* as the tuple $mk \coloneqq (G, H, \gamma)$, where $G \leftarrow G_1$ and $H \leftarrow G_2$ are randomly selected generators and $\gamma \leftarrow \mathbb{Z}_p^\times$.
- DBE.Join($mk = (G, H, \gamma)$) - allows a user to friend someone by sharing a *decryption key* derived from their manager key as the tuple $dk \coloneqq (x, A, B)$, where $x \leftarrow \mathbb{Z}_p^\times$ is fresh, $A \coloneqq G^{\frac{x}{\gamma+x}}$, $B \coloneqq H^{\frac{1}{\gamma+x}}$. If x and γ happen to be inverses, resample x. In our construction of MP3, we add an additional component to the decryption key, κ, a shared random symmetric key for AEAD that is persistent even when (x, A, B) is reassigned. Thus, the decryption key derived is $dk \coloneqq (x, A, B, \kappa)$.
- DBE.Encrypt($mk = (G, H, \gamma)$) - generates a shared secret $K \coloneqq e(G, H)^w$ and two ciphertexts: $C_1 \coloneqq G^{w\gamma}$ and $C_2 \coloneqq H^w$, where $w \leftarrow \mathbb{Z}_p^\times$.
- DBE.Decrypt($dk = (x, A, B, \kappa), C_1, C_2$) - takes as input a decryption key and the ciphertexts and computes the shared secret $K = e(C_1, B) \cdot e(A, C_2)$. Notice that this is the same shared secret computed by the broadcast manager in DBE.Encrypt.
- DBE.Revoke($mk = (G, H, \gamma), x_r, B_r$) - revokes the user with decryption key that contains x_r and B_r from the group of the user with manager key mk by updating $H \coloneqq H^{\frac{1}{\gamma+x_r}}$. The user then advertises x_r and $H^{\frac{1}{\gamma+x_r}}$ to all their buddies.
- DBE.Update($dk = (x, A, B, \kappa), x_r, B_r$) - every non-revoked user with decryption key $dk = (x, A, B, \kappa)$ must update their decryption key via $B \coloneqq \left(\frac{B_r}{B}\right)^{\frac{1}{x-x_r}}$ to revoke the user who owns x_r and B_r. Note that the revoked buddy will not be able to update their B value due to not being able to compute $\frac{1}{x_r - x_r}$. It is important to notice that this revocation is explicit to the revoked buddy, but in our construction of MP3 we add some auxiliary functionality in the long-term epoch to make revocations plausibly deniable as described in Sect. 3.7.
- DBE.ShiftMK($mk = (G, H, \gamma), \lambda$) - updates $G \coloneqq G^\lambda$ and $H \coloneqq H^\lambda$.
- DBE.ShiftDK($dk = (x, A, B, \kappa), \lambda$) - updates $A \coloneqq A^\lambda$ and $B \coloneqq B^\lambda$.

3.4 Setup

Initialization. To participate in MP3, Alice generates a manager key

$$mk_a \coloneqq \text{DBE.}\textsc{Setup}()$$

She also generates a set of private-public base key pairs

$$(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}}) \quad \text{and} \quad (z_{a,\mathsf{init}}, q_{a,\mathsf{init}})$$

where $Y_{a,\mathsf{init}}$ is a randomly generated ECDSA private key and $P_{a,\mathsf{init}} = g_{dsa}^{Y_{a,\mathsf{init}}}$, and $z_{a,\mathsf{init}}$ is a BLS private key randomly selected from $\mathbb{Z}_p^\times$ and $q_{a,\mathsf{init}} = g_3^{z_{a,\mathsf{init}}}$.

Adding Buddies. For every friend f of Alice, she derives a decryption key

$$dk_{af} \coloneqq \text{DBE.}\textsc{Join}(mk_a)$$

and shares it along with her public base keys out-of-band. Assuming that the current long-term epoch is T_j, Alice also shares her most recent shared secrets K, from DBE.Encrypt, and R, her most recently generated random bit-string R, i.e., Alice shares[1] $(dk_{af}, P_{a,\mathsf{init}}, q_{a,\mathsf{init}}, K, R)$ with friend f out-of-band.

3.5 Long-Term Epoch

Registration. To register for epoch T_j, Alice must register during epoch T_{j-1}. Let $T_{j'}$ be the last long-term epoch in which Alice has registered. To begin, Alice computes a new long-term private-public key pair for T_j

$$h_j \coloneqq H_1(T_j \parallel K_{j'} \oplus R_{j'}), \quad Y_a(j) \coloneqq Y_{a,\mathsf{init}} \cdot h_j, \quad P_a(j) \coloneqq P_{a,\mathsf{init}}^{h_j}$$

as well as a new short-term private-public key pair for all short-term epochs during T_j

$$z_a(j) \coloneqq z_{a,\mathsf{init}} \cdot h_j, \quad q_a(j) \coloneqq q_{a,\mathsf{init}}^{h_j}$$

where $K_{j'}$ is Alice's shared secret from calling DBE.Encrypt in epoch $T_{j'}$ and $R_{j'}$ is a random bit-string generated in epoch $T_{j'}$. The $\oplus$ between $K_{j'}$ and $R_{j'}$ implicitly converts $K_{j'}$ to a bit-string of some length and the length of $R_{j'}$ is defined to be that length.

During a long-term registration, Alice has the opportunity to revoke or suspend, or unrevoke a certain number of her buddies. Denote the number of buddies she can revoke or suspend at each long-term epoch as N_{rev} and the number of buddies she can unrevoke at each long-term epoch as N_{unrev}.

Every long-term epoch registration, Alice will upload $1+N_{\text{unrev}}$ records to the long-term presence database. A single record is for all her buddies she wishes

[1] If this is the very first epoch, Alice must generate an initial $K \coloneqq K_{\mathsf{init}}$ and $R \coloneqq R_{\mathsf{init}}$ to share with her buddies to bootstrap the protocol.

to continue being buddies with or that she wishes to revoke or suspend. The remaining N_{unrev} records are for buddies she had previously suspended and wishes to "re-friend." Alice constructs the single record according to Algorithm 1 and she constructs the other N_{unrev} records according to Algorithm 2.

All long-term records are of the form

$$\overbrace{P}^{(a)} \| \underbrace{x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}}}_{(b)} \| \overbrace{E_1 \| \cdots \| E_{N_{\text{rev}}}}^{(c)} \| \underbrace{C_1 \| C_2}_{(d)} \| \overbrace{R}^{(e)} \| \underbrace{S}_{(f)}$$

where (a) is a long-term identifier, (b) contains the x and B values of all buddies to revoke or suspend, (c) contains new encrypted decryption keys for all buddies revoked in (b), (d) contains the ciphertext components from DBE, (e) is a random bit string, and (f) is the signature for the record that can be verified with (a). Further details of how (b) and (c) are used to allow *plausibly deniable* revocation and suspension are described in Sect. 3.7.

Upon receiving a record of the form

$$P \| x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}} \| E_1 \| \cdots \| E_{N_{\text{rev}}} \| C_1 \| C_2 \| R \| S$$

from Alice, the registration server verifies the signature S with P. If the signature is valid, it computes $\text{ID}_a(j) \coloneqq H_2(P)$ and stores the $\langle$key, value$\rangle$ pair

$$\langle \text{ID}_a(j), x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}} \| E_1 \| \cdots \| E_{N_{\text{rev}}} \| C_1 \| C_2 \| R \| S \rangle$$

Otherwise, if the signature is invalid, nothing is stored.

Lookup. To look up Alice's presence for epoch T_j, Bob first requests the metadata associated with the databases from each lookup server. Since all lookup servers have the same database, the metadata should be the same, but in the event that some of the servers are dishonest, he takes the majority of the received metadata. The metadata contains information about the number of buckets and size of the buckets. He then computes $P_a(j) \coloneqq P_{a,\mathsf{init}}^{H_1(T_j \| K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered, and subsequently computes $\text{ID}_a(j) = H_2(P_a(j))$. Finally, he builds a PIR request using the metadata for $\text{ID}_a(j)$ to retrieve a record of the form

Algorithm 1. Alice computing her long-term presence record

Input:
$mk_a = (G_a, H_a, \gamma_a)$, Alice's manager key
$(Y_a(j), P_a(j))$, Alice's private-public key pair for T_j
$\mathcal{R}$, the set of buddies' decryption keys to MP3.Revoke or suspend
$\mathcal{B}$, the set of buddies' decryption keys to continue being buddies
Output: Long-term presence record

function
  $(C_1, C_2, K) \coloneqq$ DBE.Encrypt(mk_a)
  Generate and store a random bit-string R_j
  $\lambda \coloneqq H_3(K)$
  DBE.ShiftMK(mk_a, λ)
  Store $K_j \coloneqq K^{\lambda^2}$
  $record \coloneqq P_a(j)$
  $n \coloneqq N_{\text{rev}} - |\mathcal{R}|$
  $\mathcal{B}_{\text{padded}} \coloneqq$ pad $\mathcal{B}$ with random decryption keys upto N_{fmax}
  $\mathcal{F} \coloneqq n$ decryption keys chosen uniformly from $\mathcal{B}_{\text{padded}}$
  for each $(x, A, B, \kappa) \in \mathcal{R} \cup \mathcal{F}$ **do**
    $record \coloneqq record \,\|\, x \,\|\, H^{\frac{1}{\gamma + x_r}}$
    DBE.Revoke(mk_a, x, B)
  end for
  for each $(x, A, B, \kappa) \in \mathcal{R}$ **do**
    $x' \leftarrow \mathbb{Z}_p^{\times}$, $A' \leftarrow G_1$, $B' \leftarrow G_2$ ▷ Here, x' is fresh
    ▷ Store the following in case we want to unrevoke this buddy
    Store $((x', A', B', \kappa), C_1, C_2, R_j)$ in a global set $\mathcal{U}$
    $E \coloneqq \text{AEAD}_{\kappa}^{j}(x' \,\|\, A' \,\|\, B')$
    $record \coloneqq record \,\|\, E$
  end for
  for each $(x, A, B, \kappa) \in \mathcal{F}$ **do**
    ▷ generate new and valid x, A, and B
    $(x', A', B', \_) \coloneqq$ DBE.Join(mk_a)
    $E \coloneqq \text{AEAD}_{\kappa}^{j}(x' \,\|\, A' \,\|\, B')$
    $record \coloneqq record \,\|\, E$
  end for
  Shuffle the ciphertexts E (the encrypted $x \,\|\, A \,\|\, B$) in *record*
  $record \coloneqq record \,\|\, C_1 \,\|\, C_2 \,\|\, R_j$
  $S \coloneqq \text{ECDSA-Sign}_{Y_a(j)}(record)$
  $record \coloneqq record \,\|\, S$
  return *record*
end function

$$x_1 \,\|\, B_1 \,\|\, \cdots \,\|\, x_{N_{\text{rev}}} \,\|\, B_{N_{\text{rev}}} \,\|\, E_1 \,\|\, \cdots \,\|\, E_{N_{\text{rev}}} \,\|\, C_1 \,\|\, C_2 \,\|\, R \,\|\, S$$

Bob then processes this long-term record according to Algorithm 3 and stores the returned K and R values for computing Alice's long- and short-term identifiers in the next long-term epochs.

Algorithm 2. Alice computing N_{unrev} presence records to unrevoke buddies

Input:
$\mathcal{U}$, the global set of buddies to unrevoke
T_j, the current long-term epoch
mk_a, Alice's manager key
$(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}})$, Alice's initial long-term epoch base keys
K_j stored from Algorithm 1
R_j stored from Algorithm 1
Output: list of N_{unrev} long-term presence records

1: **function**
2: $ret \coloneqq$ new list
3: **for each** $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}}) \in \mathcal{U}$ to unrevoke **do**
4: Remove $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}})$ from $\mathcal{U}$
5: $K_{\mathsf{revoked}} \coloneqq \text{DBE.}\textsc{Decrypt}((x, A, B, \kappa), C_1, C_2)$
6: $Y_{\mathsf{revoked}} \coloneqq Y_{a,\mathsf{init}} \cdot H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})$
7: $P_{\mathsf{revoked}} \coloneqq P_{a,\mathsf{init}}^{H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})}$
8: $R_{\mathsf{unrevoked}} \coloneqq K_j \oplus R_j \oplus K_{\mathsf{revoked}}$
9: $record \coloneqq P_{\mathsf{revoked}}$
10: ▷ We concatenate enough bytes so the record is the same length as that of Algorithm 1. We also make sure the byte encodings are of the correct type.
11: $record \coloneqq record \parallel \langle N_{\text{unrev}} - 1$ encrypted random triples $\overline{x} \leftarrow \mathbb{Z}_p^\times \parallel \overline{A} \leftarrow G_1 \parallel \overline{B} \leftarrow G_2 \rangle$
12: ▷ generate a fresh decryption key for the unrevoked buddy
13: $(x', A', B', \_) \coloneqq \text{DBE.}\textsc{Join}(mk_a)$
14: $record \coloneqq record \parallel \text{AEAD}_\kappa^j(x' \parallel A' \parallel B')$
15: Shuffle the record as in Algorithm 1 line 28
16: $record \coloneqq record \parallel C_1 \parallel C_2 \parallel R_{\mathsf{unrevoked}}$
17: $S_{\mathsf{unrevoked}} \coloneqq \text{ECDSA-}\textsc{Sign}_{Y_{\mathsf{revoked}}}(record)$
18: $record \coloneqq record \parallel S_{\mathsf{unrevoked}}$
19: store $record$ in ret
20: **end for**
21: **if** number of buddies unrevoked $< N_{\text{unrev}}$ **then**
22: ▷ these records must be of the correct encoding
23: store random records in ret so that its length is N_{unrev}
24: **end if**
25: **return** ret
26: **end function**

3.6 Short-Term Epoch

Registration. To register for epoch t_i, Alice must register during t_{i-1}. Assume that epoch t_i is during epoch T_j. Recall that Alice computed the private-public key pair $(z_a(j), q_a(j))$ during the long-term registration for T_j. To begin, Alice encrypts her presence message, $m_a(i)$ as follows:

$$k_a(i) \coloneqq \text{PRF}_{H_4(q_a(j))}(t_i) \qquad c_a(i) \coloneqq \text{AEAD}_{k_a(i)}^i(m_a(i))$$

She then computes the unforgeable signature:

$$s_a(i) \coloneqq H_5(t_i)^{z_a(j)}$$

Alice then uploads $c_a(i) \parallel s_a(i)$ to the short-term registration server.

Upon receiving Alice's record, the short-term registration server will compute $\text{id}_a(i) = H_6(e_{bls}(g_1, s_a(i)))$, and store $\langle \text{id}_a(i), c_a(i) \rangle$. Additionally, $\langle \text{id}_a(i), s_a(i) \rangle$ is stored in a short-term *signature database* to audit.

Algorithm 3. Bob processing the long-term record of Alice

Input:
$P_a(j)$, Alice's long-term key for T_j
$x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S$, long-term record
Output: K and R for the next long-term epoch

1: **function**
2: ▷ Only process the record if the signature is valid, otherwise the lookup server was malicious
3: **if** S is valid signature for the record with $P_a(j)$ **then**
4: *plausiblyRevoked* ≔ **false**
5: **for** $i = 1$ **to** N_{rev} **do**
6: **if** $x_{ab} \neq x_i$ **then**
7: DBE.Update(dk_{ab}, x_i, B_i)
8: **else**
9: *plausiblyRevoked* ≔ **true**
10: **end if**
11: **end for**
12: **if** *plausiblyRevoked* **then**
13: **for** $i \coloneqq 1$ **to** N_{rev} **do**
14: Try to decrypt E_i with κ_{ab}
15: **if** successfully decrypted E_i **then**
16: $(x', A', B') \coloneqq E_i$ decrypted with κ_{ab}
17: $x_{ab} \coloneqq x'$, $A_{ab} \coloneqq A'$, $B_{ab} \coloneqq B'$
18: **end if**
19: **end for**
20: ▷ Note that we updated dk_{ab} in line 17
21: $K \coloneqq$ DBE.Decrypt(dk_{ab}, C_1, C_2)
22: **return** K, R
23: **else**
24: $K \coloneqq$ DBE.Decrypt(dk_{ab}, C_1, C_2)
25: $\lambda \coloneqq H_3(K)$
26: DBE.ShiftDK(dk_{ab}, λ)
27: $K \coloneqq K^{\lambda^2}$
28: **return** K, R
29: **end if**
30: **end if**
31: **end function**

Lookup. To lookup Alice's presence for epoch t_i, Bob requests the metadata associated with the short-term databases in the same manner as in the long-term epoch. To begin, he first computes $q_a(j) \coloneqq q_{a,\mathsf{init}}^{H_1(T_j \| K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered. Then he computes $\text{id}_a(i) \coloneqq H_6(e_{bls}(q_a(j), H_5(t_i))$ which is equivalent to the registration server's computation of $H_6(e_{bls}(g_3, s_a(i)))$ by the properties of pairings. He then builds a PIR request for $\text{id}_a(i)$ to retrieve $c_a(i)$. Bob can be certain that $c_a(i)$ is from Alice due to the unforgeable signature $s_a(i)$ by auditing the signature database.

Bob can then compute $k_a(i) \coloneqq \mathrm{PRF}_{H_4(q_a(j))}(t_i)$ and decrypts $c_a(i)$ retrieving $m_a(i)$. The decryption being successful implies that Alice is online during epoch t_i. As in the long-term epoch, in order to not leak information of how many buddies Bob has, he must pad his lookup to N_{fmax} ids.

3.7 Details

Unrevoking and Unsuspending. For simplicity in this section we assume $N_{\text{rev}} = 1$. The long-term presence record of Alice that MP3.Revokes Bob takes the form

$$P \parallel x_{ab} \parallel B_{ab} \parallel E_{ab} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

That is x_{ab} and B_{ab} are Bob's x and B values and E_{ab} contains a triple of random (x, A, B). encrypted with κ_{ab}, i.e., x, A, and B were *not* generated from Alice's manager key. This means that for future epochs, Bob cannot compute the proper K (Algorithm 3 line 21), and thus can no longer query for Alice. Alice can unrevoke Bob by computing the incorrect K that he computes from when he was revoked (Algorithm 2 line 5) and use this to upload a record that Bob *can* query for. This record will allow Bob to compute the correct K and R values for Alice for future epochs, as well as providing Bob with fresh x, A, and B values that *are* generated from Alice's manager key (Algorithm 2 line 14). This is possible by storing the correct K value within the R value in this "unrevoking" record (Algorithm 2 line 8). This allows Bob to compute the proper identifier for Alice in the next long-term epoch. Thus, Bob can continue to query Alice's presence as before.

Alice's friends must process all of her long-term database entries so they must query all long-term databases. To allow users to be offline for extended periods of time MP3 stores the previous 30 days worth of long-term database. If a user does not come online for more than 30 days they must share new keys with all of their friends. Revocations in the database entries are plausible deniable but a user may be able to notice if a friend does not come online anymore indicating they may have been revoked. This problem is inherent to presence systems.

Plausible Deniability of Revocation and Suspension. For MP3.Revoke to be deniable a revoked user must not be able to determine they have been revoked. MP3 implements this by DBE.Revoking users that have not been MP3.Revoked and issues these users new valid DBE decryption keys. Where as friends which are MP3.Revoked are issued a new random decryption key. For a friend to determine if they have been MP3.Revoked they must be able to distinguish between valid and random decryption keys. We introduce DBE.ShiftMK and DBE.ShiftDK to make the decryption keys indistinguishable. Appendix A details a distinguisher if DBE.ShitfMK and DBE.ShiftDK are not used.

More formally, if a friend can distinguish a transcript where they have been MP3.Revoked from a transcript where they have been DBE.Revoked but not MP3.Revoked they can be used as a Decisional Diffie-Hellman (DDH)

distinguisher. That is, given (g, g^x, g^y, g^z) determine if $g^{xy} = g^z$. We assume all hash functions are modeled as random oracles.

We now quickly sketch the proof. If given a decryption key (x, A, B) and ciphertext $(C_1 = G^{w_0\gamma}, C_2 = H^{w_0})$ and ciphertext $(C_1' = G^{w_1\gamma\lambda}, C_2 = H^{w_1 \frac{1}{\gamma+x}\lambda})$ where w_0, γ, w_1, and λ are random, and given (x', A', B'), determine if (x', A', B') are valid decryption keys or random. Given a DDH challenger we can construct the above problem. Let the manager key $mk = (G = g, H = g^x, \gamma)$. Then $C_1 = g^{w_0\gamma}$, $C_2 = g^{xw_0}$. Define $\lambda = y$, thus $mk' = (G = g^y, H = g^{z\frac{1}{\gamma+x}}, \gamma)$ and $C_1' = g^{yw_1\gamma}$, and $C_2' = g^{z\frac{1}{\gamma+x}w_1}$. If a friend can determine they have been MP3.Revoked they can win the DDH game.

Complexity Comparisons. During each long-term epoch in DP5, $N \cdot N_{\text{fmax}}$ records are stored, where each record is a constant size. Thus, the registration bandwidth is $\Theta(N \cdot N_{\text{fmax}})$. During each long-term epoch in MP3, $N \cdot (N_{\text{unrev}} + 1)$ records are stored, where each record's length scales with N_{rev}. Thus, the registration bandwidth complexity is $\Theta(N \cdot N_{\text{unrev}} \cdot N_{\text{rev}})$. In reality, N_{unrev} and N_{rev} will be relatively constant[2] compared to N. This implies that (as functions of N and N_{fmax}) the registration bandwidth complexities for DP5 and MP3 are $\Theta(N \cdot N_{\text{fmax}})$ and $\Theta(N)$, respectively.

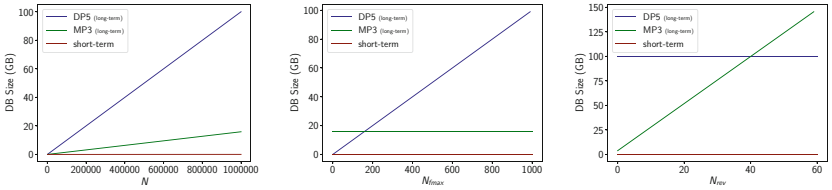
With the PIR protocol used in both DP5 and MP3, the bandwidth cost per query of a single record scales with the square root of the size of the database[3]. Also, recall that each user must query for N_{fmax} buddies to not reveal any information about their number of buddies. This implies that the bandwidth complexities for an entire long-term epoch (assuming all users query) for DP5 and MP3 are $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ and $\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$, respectively. Using the same approximations for N_{unrev} and N_{rev} as above, this results in lookup bandwidth complexities of $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ for DP5 and $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$ for MP3. Similar arguments can be made for the shared short-term epoch of DP5 and MP3 and are summarized in Table 1.

4 Experimental Results

Implementation. Our MP3 library is implemented in 350 lines of C and 4000 lines of C++. The core cryptography relies on OpenSSL for AES, SHA-256, and elliptic curve arithmetic and signatures; RELIC [14] for pairing-friendly curves; and Percy++ [15] for PIR. The groups G_1, G_2, and G_T are defined by the Optimal Ate pairing over a 256-bit Barreto-Naehrig curve. We use a 224-bit Elliptic Curve, specifically secp224r1, for ECDSA, though the choice was

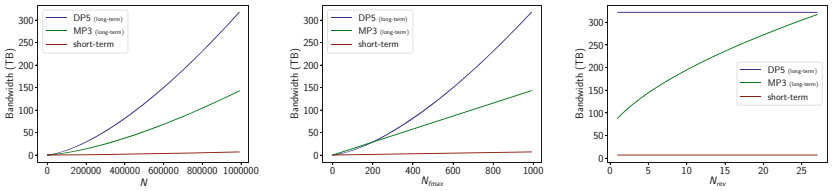
[2] Arguments for why this is a valid assumption are discussed in Sect. 4.

[3] As in the PIR protocol in DP5, we constructed $r = \lceil \sqrt{ns} \rceil$ buckets and we can upper bound the size of each bucket by $\left(\frac{n}{r} + \sqrt{\frac{n}{r}}\right) \cdot s \approx \sqrt{ns} + \sqrt[4]{ns^3}$ (here, s is the size of each record in bytes); since a query results in an entire bucket, this scales with the square root of the size of the database.



(a) $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (b) $N = 1\,000\,000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (c) $N = 1\,000\,000$, $N_{\text{fmax}} = 1000$, $N_{\text{unrev}} = 5$

Fig. 1. Presence database sizes



(a) $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (b) $N = 1000000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (c) $N = 1000000$, $N_{\text{fmax}} = 1000$, $N_{\text{unrev}} = 5$

Fig. 2. Lookup server bandwidth

arbitrary. AEAD is implemented using AES in Galois/Counter Mode (GCM). The PRFs and hash functions are implemented using SHA-256.

Evaluation. To evaluate the performance of MP3 vs. DP5, we simulated both protocols in a "worst-case" scenario with the number of users ranging from 1000 to 1 000 000 clients. To simulate the worst-case scenario, we had all clients perform registration and lookup for all epochs. All simulations were run on a machine with dual Intel Xeon E5-2630 v3 CPUs and 256GB of RAM. The number of PIR servers (N_{lookup}) was held fixed at 3 for all setups and both protocols. The equivalent of 1 year of execution were simulated in all setups. For both MP3 and DP5, the most expensive components are the lookup servers from both CPU and bandwidth perspectives.

Figure 1 compares the size of the presence databases for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. If we fix N_{rev} and N_{unrev} to small constants compared to N, it is obvious that the size of the long-term database of MP3 is significantly less than that of DP5. A smaller database implies cheaper lookup costs in the context of both bandwidth and CPU. Additionally, we can raise N_{rev} quite a bit and still maintain a smaller long-term database than that of DP5. It's also important to see how cheap the short-term database is relative to the long-term databases. Also note that the presence database sizes are proportional to the registration bandwidth.

Figure 3 compares the client-facing latency of the long-term epochs of MP3 and DP5 as well as the shared short-term epoch, for $N = 100\,000$ users. The latency of MP3's long-term epoch is smaller than that of DP5 due to the inherently smaller database. The short-term epoch's latency is even less as the short-term databases are even smaller.

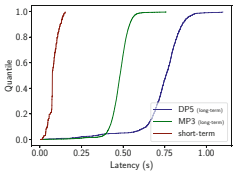


Fig. 3. Client-facing lookup latency excluding RTT. $N = 100\,000$, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$

Figure 2 compares the bandwidth of a single long-term lookup server for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. The same pattern occurs that we saw in Fig. 1 - for relatively constant N_{rev} and N_{unrev}, the bandwidth required is significantly less for MP3 than for DP5 and the bandwidth requirements of the short-term epoch are negligible compared to that of the long-term epochs.

Table 1. Bandwidth complexities comparing MP3 and DP5 as a function of N, N_{fmax}, N_{rev}, and N_{unrev}.

		Client		Server	
		registration	lookup	registration	lookup
long-term	DP5	$\Theta(N_{\text{fmax}})$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}^{3/2}\right)$	$\Theta(N \cdot N_{\text{fmax}})$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$
	MP3	$\Theta(N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{1/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N \cdot N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$
short-term	DP5 MP3	$\Theta(1)$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N)$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$

Discussion of Scalability and Cost Improvements. A primary bottleneck in DP5 is its lack of scaling with large number of users, specifically for long-term epochs. MP3 solves just that. The complexity for bandwidth usage of all operations are summarized in Table 1.

The bulk of the cost in running a service such as MP3 or DP5 comes from the bandwidth usage of the given protocol. N_{rev} and N_{unrev} are always less than or equal to N_{fmax} by definition. In reality, with a 24-h long-term epoch, setting N_{rev} and N_{unrev} to a small constant is very reasonable[4]; therefore, MP3 is significantly cheaper during long-term epochs, and thus overcomes the scalability bottleneck of DP5.

In Fig. 2a, with $N = 1\,000\,000$ users, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$, we can see that MP3 uses about half the bandwidth of that of DP5 and it's evident that the savings grow as the number of users increases.

5 Conclusion

The proposed protocol reduced the complexity and cost of the most expensive component of DP5, i.e., the long-term presence database. In reference to

[4] Social networks such as Twitter disallow bulk unfollowing [16], making our argument about setting N_{rev} and N_{unrev} to a small/constant value even stronger.

DP5, MP3 requires about half the bandwidth for $N = 1\,000\,000$ users, and this reduction only increases as the number of users increases. Therefore, MP3 is a more efficient private presence protocol than DP5. Future work will be directed towards formally proving the security properties of MP3.

Acknowledgments. This research was partially supported by the NSF under grant 1314637 and by an Undergraduate Research Opportunities Program (UROP) Award from the University of Minnesota.

A Modifications to Dynamic Broadcast Encryption

Recall the operations of DBE from Sect. 3.3. Our modifications to DBE [7] only add the DBE.ShiftMK and DBE.ShiftDK operations. These operations are required for plausibly deniable revocations and suspensions. Recall that in a long-term database record for Alice in which Bob is actually revoked, Bob's old decryption key (dk_{ab}) is revoked and he is issued a new, but invalid, decryption key (dk'_{ab}). Without DBE.ShiftMK and DBE.ShiftDK, Bob could use dk'_{ab} to invert the Update operation and detect whether he was revoked or not.

The inverse update function is:

- DBE.Update$^{-1}(dk' = (x', A', B', \kappa'), x_r, B_r)$ - takes as input a decryption key and a revocation values and computes a new decryption key $dk := (x', A', B, \kappa')$ where $B := \frac{B_r}{B'^{(x'-x_r)}}$ that *can* decrypt ciphertexts *created before* the revocation.

Assuming DBE.ShiftMK and DBE.ShiftDK are not in place, a revoked user Bob can use DBE.Update^{-1} to *detect* that he has been revoked by Alice. Given two long-term presence records of Alice, where the former has not revoked Bob and the latter has revoked Bob, Bob can apply the DBE.Update^{-1} function to his new decryption key and compute a decryption key for the former presence record.

Let Bob's decryption keys for the former presence record be dk_{ab} and let Bob's decryption key for the latter presence record (after calling DBE.Update^{-1}) be dk'_{ab}. Also let C_1, C_2 be the ciphertext components from the former presence record. To detect if he has been revoked, all he must do is check if DBE.Decrypt$(dk_{ab}, C_1, C_2) \neq$ DBE.Decrypt(dk'_{ab}, C_1, C_2). If the statement is true, then Bob has been revoked by Alice.

By introducing DBE.ShiftMK and DBE.ShiftDK we create a one-way operation to the revocation process of MP3; thus Bob *cannot* invert the DBE.ShiftMK and DBE.ShiftDK functions without the knowledge of the plaintext of (C_1, C_2) and therefore cannot detect whether or not he was revoked.

B Availability Against Malicious Parties

Some conventional approaches to ensure availability against malicious parties cannot be applied directly to privacy-preserving protocols, as they can leak information. This causes several challenges: keeping the databases small, ensuring

the registration server stores all uploaded presence records, ensuring the lookup servers store all presence records and do not modify them.

A malicious client could upload many presence records during a given epoch, causing a denial of service (DoS) for all other clients. If a malicious client were using an anonymous channel, authentication would compromise the anonymity of that client, defeating the purpose of MP3. To eliminate this, k-times anonymous authentication schemes have been proposed [17,18]. In these schemes, users are guaranteed anonymity up to k times; that is, if a user authenticates $k+1$ times, the identity of the user can be computed. Such private rate-limiting schemes can be used to limit the number of times a client registers during a given epoch without losing anonymity.

In the case that the registration server is dishonest and drops records, a user could "friend themself" to ensure that their presence records are being stored, by looking themself up during every epoch. Note that all presence records are indistinguishable, so the registration server can not target specific records for dropping.

Lastly, in the case that the lookup servers are dishonest and modify the database, Devet et al. propose a robust PIR scheme [19] that allows detection of malicious servers. This detection requires at least $t+2$ honest servers, where t is the number of servers needed to collude to be able to determine the data in a query. This robust PIR scheme is implemented in MP3.

References

1. Rushe, D.: Lavabit founder refused FBI order to hand over email encryption keys. The Guardian, October 2013
2. Apple: Approach to privacy. http://www.apple.com/privacy/approach-to-privacy/
3. Open Whisper Systems. https://whispersystems.org/
4. Marlinspike, M.: Facebook messenger deploys signal protocol for end to end encryption. https://whispersystems.org/blog/facebook-messenger
5. Borisov, N., Danezis, G., Goldberg, I.: DP5: a private presence service. Proc. Priv. Enhanc. Technol. **2015**(2), 4–24 (2015)
6. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: IEEE Symposium on Foundations of Computer Science (1995)
7. Delerablée, C., Paillier, P., Pointcheval, D.: Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In: Takagi, T., Okamoto, E., Okamoto, T., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 39–59. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73489-5_4
8. Wolinsky, D.I., Corrigan-Gibbs, H., Ford, B., Johnson, A.: Dissent in numbers: making strong anonymity scale. Presented as Part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2012), pp. 179–182 (2012)
9. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: an anonymous messaging system handling millions of users. In: 2015 IEEE Symposium on Security and Privacy, pp. 321–338. IEEE (2015)
10. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle. Proc. Priv. Enhanc. Technol. **2016**(2), 115–134 (2016)

11. Angel, S., Setty, S.T.: Unobservable communication over fully untrusted infrastructure. In: OSDI, pp. 551–569 (2016)
12. ANSI X9.62–1998: Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA). American National Standards Institute (ANSI), Washington, DC (1998)
13. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
14. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
15. Goldberg, I., Devet, C., Lueks, W., Yang, A., Hendry, P., Henry, R.: Percy++ project on sourceforge (2014). http://percy.sourceforge.net
16. Suspension - what's the daily/hourly unfollow limit for each user? What is the aggressive behaviour? https://twittercommunity.com/t/suspension-whats-the-daily-hourly-unfollow-limit-for-each-user-what-is-the-aggressive-behaviour/13971
17. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_28
18. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: Blacklistable anonymous credentials: blocking misbehaving users without TTPs. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 72–81. ACM (2007)
19. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. Presented as Part of the 21st USENIX Security Symposium (USENIX Security 12), pp. 269–283 (2012)

Cryptographic Integrity

Short Unique Signatures from RSA with a Tight Security Reduction (in the Random Oracle Model)

Hovav Shacham(✉)

The University of Texas at Austin, Austin, USA
hovav@cs.ucsd.edu

Abstract. A signature scheme is unique if for every public key and message there is only one signature that is accepted as valid by the verification algorithm. At Crypto 2017, Guo, Chen, Susilo, Lai, Yang, and Mu gave a unique signature scheme whose security proof incurred a security loss logarithmic in the number of hash oracle queries made by the adversary, bypassing an argument due to Bader, Jager, Li, and Schäge that the security loss must be at least linear in the number of signing oracle queries made by the adversary. Unfortunately, the number of elements in a Guo et al. signature is also logarithmic in the number of hash oracle queries made by the adversary.

We translate Guo et al.'s signatures into the integer factorization setting. Doing so allows us to bring to bear signature aggregation ideas due to Lysyanskaya, Micali, Reyzin, and Shacham. We obtain unique signatures that are short *and* have a tight security reduction from the RSA problem.

1 Introduction

A signature scheme is unique if for every public key and message there is only one signature accepted as valid by the verification algorithm. Unique signatures make useful building blocks in primitives such as verifiable random functions [16].

At Eurocrypt 2016, Bader et al. [1] used a meta-reduction technique to show that any security proof for a unique signature scheme given some static assumption must incur a q_S security loss, where q_S is the number of signing oracle queries made by the adversary. In other words, ϵ-intractability of the underlying hard problem will translate only into $q_S\epsilon$-unforgeability of the unique signature. By contrast, there exist signature schemes with just *two* valid signatures per message whose security reductions incur a security loss of 2 [8].

At Crypto 2017, Guo et al. [9] observed that the Bader et al. meta-reduction assumed that the unforgeability reduction would extract the information it needed to break the underlying hard problem from the adversary's forgery.

H. Shacham—This material is based upon work supported by the National Science Foundation under grant No. CNS-1410031.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 61–79, 2018.
https://doi.org/10.1007/978-3-662-58387-6_4

An unforgeability reduction that instead extracts the information it needs from the adversary's hash oracle queries (in the random oracle model) would fall outside the Bader et al. model and might be able to give better security guarantees.

Guo et al. presented a unique signature scheme with a proof of security assuming the computational Diffie-Hellman problem is intractable with a security loss of just $nq_H^{1/n}$, where q_H is the number of hash oracle queries made by the forger and n is a scheme parameter. Security loss is minimized by choosing $n \approx \ln q_H$. Unfortunately, signatures in Guo et al.'s scheme have length linear in n; Guo et al. describe the resulting signatures as "somewhat impractical even taking the loss factor into account," but "theoretically interesting."

Our Contributions. We show that the ideas of Guo et al. give rise not just to "theoretically interesting" unique signatures with tight reductions but also to practical ones. We give a unique signature scheme with the same security loss as that of Guo et al. — $nq_H^{1/n}$ — where signatures consist of just two group elements, compared to $n + 1$ for Guo et al.

Guo et al. signatures consist of $n + 1$ iterations of an underlying BLS signature. A BLS signature on a message M is of the form $H(M)^x$, where x is the secret exponent. It is straightforward to replace BLS with RSA full-domain hash signatures, of the form $H(M)^d \bmod N$, where d is a secret exponent.

Our key insight is that although BLS and RSA signatures have a similar signing operations but quite different verification operations. BLS verification transforms a signature in group G_1 into a value in group G_T by means of the pairing. It is intractable to go in the other direction, from G_T to G_1 [17]. RSA verification computes $\sigma^e \bmod N$, where e is the public exponent, producing a value that is still in $\mathbb{Z}/N\mathbb{Z}$ and therefore might be transformed into another signature to verify. This property was used by Lysyanskaya et al. [12] to build aggregate signatures from RSA. In an aggregate signature, a single, short signature takes the place of n signatures by n signers on n respective messages. A verifier given the aggregate signature σ, public keys $pk_1, \ldots, pk_n$, and messages $M_1, \ldots, M_n$ should accept only if the signing key corresponding to each pk_i was applied to M_i in the signing protocol.

The ideas of Lysyanskaya et al. do not immediately apply to give a short variant of Guo et al.'s signatures. The key difference is that in an aggregate signature scheme the verifier must still be sent the messages $M_1, \ldots, M_n$; in Guo et al.'s scheme, M_i includes $\sigma_1 \| \sigma_2 \| \cdots \| \sigma_{i-1}$; transmitting even just M_n along with the signature would negate any benefit to be had from signature aggregation.

We replace $\sigma_1 \| \sigma_2 \| \cdots \| \sigma_{i-1}$ in Guo et al.'s block messages with $G(M, 1, \sigma_1) \oplus G(M, 2, \sigma_2) \oplus \cdots \oplus G(M, i-1, \sigma_{i-1})$, essentially aggregating the block messages along with the block signatures. The Guo et al. security analysis no longer applies, and we replace it with a new security analysis that draws on both Guo et al. and Lysyanskaya et al.

The ith block signature in our scheme is of the form

$$\sigma_i = \left[\sigma_{i-1} + H(M, i, \mu_{i-1})\right]^d \bmod N$$

along with the block messages aggregated as as above. The verifier can compute μ_{i-1} as $\mu_i \oplus G(M, i, \sigma_i)$, then peel back σ_i to recover σ_{i-1} as $\sigma^e - H(M, i, \mu_{i-1}) \bmod N$. Verification consists of repeating this procedure n times starting from (σ_n, μ_n).

Our scheme resembles a construction for single-signer aggregate signatures with message recovery presented (without security proof) by Neven [14].

We show how our scheme can be instantiated from any family of certified trapdoor permutations, then explain how to obtain a suitable permutation from RSA. Our scheme shows that unique signatures built using the Guo et al. paradigm can be of more than just theoretical interest. At the 128-bit security level, and assuming $q_S = 2^{40}$ and $q_H = q_G = 2^{80}$, a signature in our scheme is under 4,000 bits, including 256 bits for the group E, whereas full-domain RSA signatures must be nearly 6,000 bits long to obtain $(128+40)$-bit factoring security and make up for their loose security reduction. (These estimates make use of the bit-length formula of Orman and Hoffman [15].)

2 Preliminaries

Trapdoor Permutations. Let D be a finite set. A permutation family Π over D specifies a randomized algorithm for generating (descriptions of) a permutation and its inverse, written $(s, t) \xleftarrow{R} \mathit{Generate}$; an evaluation algorithm $\mathit{Evaluate}(s, \cdot)$; and an inversion algorithm $\mathit{Invert}(t, \cdot)$. We require that, for all (s, t) output by *Generate*, $\mathit{Evaluate}(s, \cdot)$ be a permutation of D, and that $\mathit{Invert}\big(t, \mathit{Evaluate}(s, \cdot)\big)$ be the identity map.

A trapdoor permutation family is one way if it is hard to invert given just the forward permutation description s. Formally [12, Definition 2.1], a trapdoor permutation family is (t, ϵ)-one way if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in the game

$$\mathsf{Adv}\,\mathsf{Invert}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[x = \mathcal{A}\big(s, \mathit{Evaluate}(s, x)\big) : (s, t) \xleftarrow{R} \mathit{Generate}, x \xleftarrow{R} D\right],$$

where the probability is over the coin tosses of *Generate* and $\mathcal{A}$.

A trapdoor permutation is certified [3] if a permutation description s output by *Generate* can be efficiently recognized: if there is an algorithm *Certify* that returns 1 for every string in the set $S = \big\{s \bigm| (s, t) \xleftarrow{R} \mathit{Generate}\big\}$ and 0 for every string not in the set S.

Where it does not introduce ambiguity, we prefer a more compact notation: we write $(\pi, \pi^{-1}) \xleftarrow{R} \Pi$ in place of $(s, t) \xleftarrow{R} \mathit{Generate}$ and $\pi(\cdot)$ and $\pi^{-1}(\cdot)$ in place of $\mathit{Evaluate}(s, \cdot)$ and $\mathit{Invert}(t, \cdot)$ respectively.

Digital Signatures. A digital signature scheme consists of a randomized key generation algorithm that emits a (public) verification and a (private) signing key, written $(pk, sk) \xleftarrow{R} \mathit{KeyGen}$; a (possibly randomized) signing algorithm that takes a signing key and a message $M \in \{0,1\}^*$, and emits a signature σ, written

$\sigma \xleftarrow{R} \mathit{Sign}(sk, M)$; and a (usually not randomized) verification algorithm that takes a verification key, message, and claimed signature, and returns 0 or 1, written $1 \stackrel{?}{=} \mathit{Verify}(pk, M, \sigma)$. We require that for all $(pk, sk) \xleftarrow{R} \mathit{KeyGen}$, for all $M \in \{0,1\}^*$, and for all $\sigma \xleftarrow{R} \mathit{Sign}(sk, M)$ we have $\mathit{Verify}(pk, M, \sigma) = 1$.

A digital signature scheme is (t, q_S, ϵ) existentially unforgeable if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in the game

$$\mathsf{Adv}\,\mathsf{Forge}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l}\mathit{Verify}(pk, M^*, \sigma^*) = 1 : \\ (pk, sk) \xleftarrow{R} \mathit{KeyGen},\ (M^*, \sigma^*) \xleftarrow{R} \mathcal{A}^{\mathit{Sign}(sk,\cdot)}(pk)\end{array}\right],$$

where the probability is over the coin tosses of *KeyGen*, *Sign*, and $\mathcal{A}$, and where we require that $\mathcal{A}$ make no more than q_S queries to the signing oracle and (to exclude trivial forgeries) that $\mathcal{A}$ not have queried its signing oracle at M^*.

In the random oracle model, all parties have access to a hash oracle that returns an independently random result on every input. A digital signature scheme is (t, q_H, q_S, ϵ) if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in game $\mathsf{Adv}\,\mathsf{Forge}_{\mathcal{A}}$ above while making at most q_S signing oracle queries and at most q_H hash oracle queries. The definition generalizes naturally to multiple oracle hash functions.

A digital signature is unique if for every message $M \in \{0,1\}^*$ and every public key pk there is at most one signature σ such that $\mathit{Verify}(pk, M, \sigma) = 1$. This property must hold unconditionally, even for maliciously generated public keys that would not be emitted by *KeyGen*. In a unique signature scheme the signing and verification algorithms are not randomized.

Full-Domain Hash Signatures. Trapdoor permutations give rise to a simple, natural unique signature scheme. Let Π be a trapdoor permutation family over domain D, and let $H\colon \{0,1\}^* \to D$ be a hash function, modeled as a random oracle. The key generation algorithm *KeyGen* picks a trapdoor permutation $(s, t) \xleftarrow{R} \mathit{Generate}$ and sets $pk = s$ and $sk = t$. The signing algorithm $\mathit{Sign}(sk, M)$ parses sk as t and emits $\sigma = \mathit{Invert}\big(t, H(M)\big)$. The verification algorithm $\mathit{Verify}(pk, M, \sigma)$ parses pk as s and rejects if $\mathit{Certify}(s) \neq 1$; checks that σ is an element of D and rejects if not; and, finally, accepts if $\mathit{Evaluate}(s, \sigma) = H(M)$ and rejects otherwise.

In our compact notation, the signature on a message M is $\sigma = \pi^{-1}\big(H(M)\big)$; the verifier checks whether $\pi(\sigma) \stackrel{?}{=} H(M)$.

Certifying that s is valid ensures that $\mathit{Evaluate}(s, \cdot)$ is a permutation of D; there can be only one element of D whose image under that permutation is $H(M)$, which guarantees unique signatures.

Bellare and Rogaway showed that the full-domain hash signature scheme is existentially unforgeable in the random oracle model if the underlying trapdoor permutation is one-way [2]. Their reduction suffered a security loss of q_H. Coron gave an improved security analysis, with security loss q_S, assuming that the underlying trapdoor permutation family is homomorphic, as RSA is [6]. Coron later gave a "meta-reduction" that showed that any proof that full-domain hash

signatures are secure assuming that an underlying trapdoor permutation family is one way must incur a q_S security loss [7]. Even so, Kakvi and Kiltz proved that RSA full-domain hash signatures are secure under the phi-hiding assumption with a *tight* reduction [10]. Kakvi-Kiltz signatures have public exponent $e < N^{1/4}$. With e in this range, the RSA permutation is not certified, and the resulting signatures are not unique.

3 Unique Signatures with Tight Security Reduction

At Eurocrypt 2016, Bader et al. [1] extended Coron's meta-reduction technique to show that any security proof for a unique signature scheme assuming a static assumption like computational Diffie-Hellman or the trapdoor permutation one-wayness must incur a q_S security loss. Nevertheless, at Crypto 2017, Guo et al. [9] were able to present unique signatures secure under the computational Diffie-Hellman problem, with a tight security reduction. The Bader et al. metareduction assumes that the simulator extracts the information it uses to break the underlying problem from the adversary's forgery. In the Guo et al. signature scheme, the simulator instead extracts that information from the adversary's *hash queries*. A Guo et al. signature is built from $n+1$ blocks, where n is a system parameter. Each block consists of a BLS signature [4] on the message and the previous blocks. For the adversary to compute the ith block of a forgery in progress, it must first reveal blocks 1 through $i-1$ in a hash query; the simulator takes advantage of these hash queries to solve the underlying hard problem.

The security loss in the Guo et al. reduction is $nq_H^{1/n}$. Even $n = 2$ improves on the Bader et al. bound; with $q_H = 2^{80}$, setting $n = 55$ gives security loss less than 151.[1]

Guo et al. observe that their signature framework could be instantiated using a different underlying block signature. In this section, we translate the Guo et al. signatures to the trapdoor permutation setting, still with signature size linear in n. In the next section, we present our variant of Guo et al. signatures with $O(1)$ signature size.

Let Π be a trapdoor permutation family over domain D. Let $H\colon \{0,1\}^* \times \mathbb{N} \times D^* \to D$ be a hash function, modeled as a random oracle. Note that we can instantiate H using a hash function $H'\colon \{0,1\}^* \to D$ by means of an unambiguous encoding of $\{0,1\}^* \times \mathbb{N} \times D^*$ in $\{0,1\}^*$.

KeyGen. Pick $(s,t) \xleftarrow{R} \mathit{Generate}$. The public key is $pk = s$. The private key is $sk = t$.

Sign(sk, M)**.** Parse sk as t. For each i, $1 \le i \le n+1$, compute

$$h_i \leftarrow H\big(M, i, (\sigma_1, \sigma_2, \ldots, \sigma_{i-1})\big) \qquad \text{and} \qquad \sigma_i \leftarrow \mathit{Invert}(t, h_i).$$

The signature is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1})$.

[1] The Bitcoin network hash rate is estimated at 2^{79} hashes per day. See https://blockchain.info/charts/hash-rate, visited September 22, 2017.

Verify(pk, M, σ)**.** Parse pk as s and reject if $\mathit{Certify}(s) \neq 1$. Parse σ as $(\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1}) \in D^{n+1}$, and reject if parsing fails. For each i, $1 \leq i \leq n+1$, compute

$$h_i \leftarrow H\big(M, i, (\sigma_1, \sigma_2, \ldots, \sigma_{i-1})\big).$$

For each i, $i \leq i \leq n+1$, check that

$$\mathit{Evaluate}(s, \sigma_i) = h_i.$$

If any of these checks fails, reject; otherwise, accept.

The proof that the Guo et al. signature scheme is unforgeable [9, Section 5] can be easily adapted to show that the variant above is unforgeable as well. The simulator $\mathcal{B}$ is given the public description π of a trapdoor permutation, and an element y^* of D. Its goal is to find $x^* \in D$ such that $\pi(x^*) = y^*$. It sets π as the challenge signing key and interacts with the forger $\mathcal{A}$ as specified by Guo et al. For hash oracle queries other than the one in which it will embed the challenge, the simulator chooses $x \xleftarrow{R} D$ and returns $\pi(x)$ as the hash; this allows it to compute the corresponding block signature x, should it later need to answer a signing oracle query on the same message. For the hash oracle query where it chooses to embed the challenge, $\mathcal{A}$ simply responds with y^*. A subsequent hash oracle query from $\mathcal{B}$ that contains the next block signature on the same message will reveal x^* to $\mathcal{B}$. The proof and analysis are otherwise unchanged.

4 Short Unique Signatures with Tight Security Reduction

We now explain how to compress the signatures of Guo et al. Our scheme is inspired by the sequential aggregate signature of scheme of Lysyanskaya et al. [12].

Let D be a group with operation $+$, identity 0_D, and inverse operation $-$. Let Π be a trapdoor permutation family over domain D; we do not require any homomorphic interaction between Π and $+$.

Let λ be a positive integer (whose value depends on the security parameter, as discussed below), and let E be the set $\{0,1\}^\lambda$ together with the bitwise exclusive or operation $\oplus$ and identity 0_E.

Let $H\colon \{0,1\}^* \times \mathbb{N} \times E \to D$ and $G\colon \{0,1\}^* \times N \times D \to E$ be hash functions, modeled as random oracles. As before, we can instantiate H and G from hash functions with domain $\{0,1\}^*$ using appropriate unambiguous encodings.

KeyGen. Pick $(s,t) \xleftarrow{R} \mathit{Generate}$. The public key is $pk = s$. The private key is $sk = t$.

Sign(sk, M)**.** Parse sk as t. Set

$$\sigma_0 \leftarrow 0_D \qquad \text{and} \qquad \mu_0 \leftarrow 0_E.$$

For each i, $1 \leq i \leq n$, compute

$$\sigma_i \leftarrow \mathit{Invert}\big(t, \sigma_{i-1} + H(M, i, \mu_{i-1})\big) \tag{1}$$

and

$$\mu_i \leftarrow \mu_{i-1} \oplus G(M, i, \sigma_i). \tag{2}$$

The signature is $\sigma = (\sigma_n, \mu_n)$.

Verify(pk, M, σ)**.** Parse pk as s and reject if $\mathit{Certify}(s) \neq 1$. Parse σ as $(\sigma_n, \mu_n) \in D \times E$, and reject if parsing fails. For i from n down to 1, compute

$$\mu_{i-1} \leftarrow \mu_i \oplus G(M, i, \sigma_i) \tag{3}$$

and

$$\sigma_{i-1} \leftarrow \mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}). \tag{4}$$

Check that

$$\sigma_0 = 0_D \qquad \text{and} \qquad \mu_0 = 0_E.$$

If either of these checks fails, reject; otherwise, accept.

4.1 Proof of Correctness

Let $\sigma = (\sigma_n, \mu_n)$ be a signature on message M under keys $pk = s$ and $sk = t$. The signer computed partial values $\sigma_0, \sigma_1, \ldots, \sigma_n$ and $\mu_0, \mu_1, \ldots, \mu_n$ according to (1) and (2). The verifier will compute partial values $\sigma'_n, \sigma'_{n-1}, \ldots, \sigma'_0$ and $\mu'_n, \mu'_{n-1}, \ldots, \mu'_0$ according to (3) and (4).

We know that $\sigma'_n = \sigma_n$ and $\mu'_n = \mu_n$, because the verifier is verifying the output of the signing algorithm. Suppose that for some i we have $\sigma'_i = \sigma_i$ and $\mu'_i = \mu_i$. Then

$$\mu'_{i-1} = \mu'_i \oplus G(M, i, \sigma'_i) = \mu_i \oplus G(M, i, \sigma_i) = \mu_{i-1}$$

and

$$\begin{aligned}
\sigma'_{i-1} &= \mathit{Evaluate}(s, \sigma'_i) - H(M, i, \mu'_{i-1}) = \mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}) \\
&= \mathit{Evaluate}\Big(s, \mathit{Invert}\big(t, \sigma_{i-1} + H(M, i, \mu_{i-1})\big)\Big) - H(M, i, \mu_{i-1}) \\
&= \sigma_{i-1} + H(M, i, \mu_{i-1}) - H(M, i, \mu_{i-1}) = \sigma_{i-1}.
\end{aligned}$$

By induction, then, $\sigma'_0 = \sigma_0 = 0_D$ and $\mu'_0 = \mu_0 = 0_E$, so the verification checks will succeed.

4.2 Proof of Uniqueness

Suppose for the sake of contradiction that $\sigma \neq \sigma'$ are two valid signatures on a message M under public key pk. If pk cannot be parsed as s or $Certify(s) \neq 1$, the verification algorithm will reject both σ and σ'. Accordingly, $Evaluate(s, \cdot)$ must be a permutation of D, which means that, for any $u, v \in D$, if $Evaluate(s, u) = Evaluate(s, v)$ then $u = v$.

Parse σ as $(\sigma_n, \mu_n) \in D \times E$ and σ' as $(\sigma'_n, \mu'_n) \in D \times E$. If either signature fails to parse it will be rejected by the verification algorithm. Applied to σ, the verification algorithm will compute partial values $\sigma_n, \sigma_{n-1}, \ldots, \sigma_0$ and $\mu_n, \mu_{n-1}, \ldots, \mu_0$ according to (3) and (4). Applied to σ', the verification algorithm will compute partial values $\sigma'_n, \sigma'_{n-1}, \ldots, \sigma'_0$ and $\mu'_n, \mu'_{n-1}, \ldots, \mu'_0$ according to (3) and (4). All of the σ_i and σ'_i values must be elements of D because σ_n and σ'_n are. All of the μ_i and μ'_i values must be elements of E because μ_n and μ'_n are.

Since both σ and σ' verify, we know that $\sigma_0 = 0_D$, $\mu_0 = 0_E$, $\sigma'_0 = 0_D$, and $\mu'_0 = 0_E$. Stated another way, we know that $\sigma_0 = \sigma'_0$ and $\mu_0 = \mu'_0$. Now suppose that for some i we have $\sigma_{i-1} = \sigma'_{i-1}$ and $\mu_{i-1} = \mu'_{i-1}$. Then, substituting (4) twice in $\sigma_{i-1} = \sigma'_{i-1}$ we obtain

$$Evaluate(s, \sigma_i) - H(M, i, \mu_{i-1}) = Evaluate(s, \sigma'_i) - H(M, i, \mu'_{i-1}),$$

and, substituting $\mu'_{i-1} = \mu_{i-1}$ on the right hand side, we obtain

$$Evaluate(s, \sigma_i) - H(M, i, \mu_{i-1}) = Evaluate(s, \sigma'_i) - H(M, i, \mu_{i-1}),$$

and we can obtain

$$Evaluate(s, \sigma_i) = Evaluate(s, \sigma'_i)$$

by adding $H(M, i, \mu_{i-1})$ to both sides. Because $Evaluate(s, \cdot)$ is a permutation, we conclude that $\sigma_i = \sigma'_i$. Now, substituting (3) twice into $\mu_{i-1} = \mu'_{i-1}$, we have

$$\mu_i \oplus G(M, i, \sigma_i) = \mu'_i \oplus G(M, i, \sigma'_i)$$

and, since $\sigma_i = \sigma'_i$, we conclude that $G(M, i, \sigma_i) = G(M, i, \sigma'_i)$ and therefore $\mu_i = \mu'_i$. By induction, $\sigma_n = \sigma'_n$ and $\mu_n = \mu'_n$, contradicting the assumption that $\sigma \neq \sigma'$.

4.3 Proof of Unforgeability

Suppose, for the sake of contradiction, that there exists some algorithm $\mathcal{A}$ that forges signatures with non-negligible probability. We will show that we can use $\mathcal{A}$ to break the one-wayness of the underlying trapdoor permutation family Π.

Description of the Simulator. We describe algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of the trapdoor permutation family Π.

Environment Setup. Algorithm $\mathcal{B}$ is given a permutation key s^* and a value $y^* \in D$. Its goal is to compute $x^* \in D$ such that $\mathit{Evaluate}(s^*, x^*) = y^*$. Algorithm $\mathcal{B}$ picks an integer c^* uniformly at random from the range $[1, n]$, and then an integer k^* uniformly at random from the range $\left[1, (q_H + 1)^{(n+1-c^*)/n}\right]$. (Note that the range from which k^* is chosen depends on c^*.) Algorithm $\mathcal{B}$ initializes a global variable x^* to $\perp$ and a global counter k to 0.

Algorithm $\mathcal{B}$ sets $pk \leftarrow s^*$; it does not know the corresponding sk. It then runs $\mathcal{A}$ with input pk.

The Tables Recording Hash Oracle Queries and Responses. Algorithm $\mathcal{B}$ maintains a table to help it answer H oracle queries, which we call the H-table. The H-table starts out empty. Each row in the H-table has the following entries:

- $M \in \{0,1\}^*$, $i \in \mathbb{N}$, $\mu \in E$: these, together, are the inputs to the hash query.
- $z \in D$: the output from the hash query.
- *good* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track hash queries that could contribute to an eventual signature.
- *internal* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track H oracle queries that it initiated as part of signature generation, versus those initiated by $\mathcal{A}$.
- $x \in D \cup \{\perp\}$: a secret used by $\mathcal{B}$ as the basis for computing the answer z to some hash queries.

Algorithm $\mathcal{B}$ likewise maintains a table to help it answer G oracle queries, which we call the G-table. The G-table starts out empty. Each row in the G-table has the following entries:

- $M \in \{0,1\}^*$, $i \in \mathbb{N}$, $\sigma \in D$: these, together, are the inputs to the hash query.
- $z \in E$: the output from the hash query.
- *good* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track hash queries that could contribute to an eventual signature.
- *internal* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track G oracle queries that it initiated as part of signature generation, versus those initiated by $\mathcal{A}$.
- $\mu \in E \cup \{\perp\}$: the value algorithm $\mathcal{A}$ is expected to compute for μ_i, as the xor of μ_{i-1} and z, or $\perp$ if not known.

Answering an H oracle query. To answer an H oracle query on $(M, i, \mu) \in \{0,1\}^* \times \mathbb{N} \times E$, $\mathcal{B}$ responds as follows.

1. If there has already been an H oracle query for (M, i, μ), there will be an entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ in the H-table. Algorithm $\mathcal{B}$ responds with z. This keeps the oracle consistent if queried multiple times on the same input.
2. If $i < 1$ or $i > n$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{\text{R}} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $x \leftarrow \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.
3. If $i = 1$, $\mathcal{B}$ consults μ. If $\mu \neq 0_E$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{\text{R}} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$,

and $x = \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.

Otherwise, $\mu = 0$, and the query is relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ will decide whether to embed its challenge as the answer to this query, according to the following criteria. If this H oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{true}$; it will not embed its challenge as the answer to this query. Otherwise, $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{false}$. If $i \neq c^*$, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise, if $i = c^*$, $\mathcal{B}$ increments the global counter k. If the counter k now has a value different from k^*, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise all three of the following conditions hold: (1) the query was generated by $\mathcal{A}$; (2) i equals c^*; and (3) k, incremented, equals k^*. In this case $\mathcal{B}$ *will* embed its challenge as the answer to this query.

If $\mathcal{B}$ didn't chose to embed its challenge as the answer to this query, it selects $x \xleftarrow{\mathrm{R}} D$, computes $z \leftarrow \mathit{Evaluate}(s^*, x)$, and sets $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.

If $\mathcal{B}$ did chose to embed its challenge as the answer to this query, it sets $x \leftarrow \perp$, $z \leftarrow y^*$, and $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H table.

Before returning, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query to the G oracle at level i. Algorithm $\mathcal{B}$ examines all entries in the G-table matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'',\ \mu'')$. If $x \neq \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\sigma'' = x$. If $x = \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\mathit{Evaluate}(s^*, \sigma'') = y^*$. In either case there is at most one such entry in the G-table (in the latter case because $\mathit{Evaluate}(s^*, \cdot)$ is a permutation). If such an entry exists, it was inserted with $\mathit{good} = \texttt{false}$ but should have been inserted with $\mathit{good} = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.

If algorithm $\mathcal{B}$ was not forced to abort, it responds to the H-query with z.

4. Otherwise, we have $2 \leq i \leq n$. Algorithm $\mathcal{B}$ cannot immediately tell whether μ makes the query good — it must consult G oracle queries for the previous block, $i - 1$. Algorithm $\mathcal{B}$ searches the G-table for an entry matching $(M' = M,\ i' = i - 1,\ \sigma',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}',\ \mu' = \mu)$. There is at most one such entry in the G-table.

   If no matching entry is found, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{\mathrm{R}} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $x = \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.

   Otherwise, there is a matching entry $(M' = M,\ i' = i - 1,\ \sigma',\ z',\ \mathit{good}' = \texttt{true},\ \mu' = \mu)$. (For each M' and i' there can be at most one entry in the G-table with $\mathit{good}' = \texttt{true}$.) This means that the H query now being handled *is* relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ will decide whether to embed its challenge as the answer to this query according to the

following criteria. If this H oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $internal \leftarrow \texttt{true}$; it will not embed its challenge as the answer to this query. Otherwise, $\mathcal{B}$ sets $internal \leftarrow \texttt{false}$. If $i \neq c^*$, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise, if $i = c^*$, $\mathcal{B}$ increments the global counter k. If the counter k now has a value different from k^*, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise all three of the following conditions hold: (1) the query was generated by $\mathcal{A}$; (2) i equals c^*; and (3) k, incremented, equals k^*. In this case $\mathcal{B}$ *will* embed its challenge as the answer to this query.
If $\mathcal{B}$ didn't chose to embed its challenge as the answer to this query, it selects $x \xleftarrow{R} D$, computes $z \leftarrow Evaluate(s^*, x) - \sigma'$, and sets $good \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, good, internal, x)$ to the H table, and responds with z.
If $\mathcal{B}$ did chose to embed its challenge as the answer to this query, it sets $x \leftarrow \perp$, $z \leftarrow y^* - \sigma'$, and $good \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, good, internal, x)$ to the H table.
Before returning, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query to the G oracle at level i. Algorithm $\mathcal{B}$ examines all entries in the G-table matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ good'' = \texttt{false},\ internal'',\ \mu'')$. If $x \neq \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\sigma'' = x$. If $x = \perp$, algorithm $\mathcal{B}$ checks for such an entry with $Evaluate(s^*, \sigma'') = y^*$. In either case there is at most one such entry in the G-table (in the latter case because $Evaluate(s^*, \cdot)$ is a permutation). If such an entry exists, it was inserted with $good = \texttt{false}$ but should have been inserted with $good = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.
If algorithm $\mathcal{B}$ was not forced to abort, it responds to the H-query with z.

Answering a G oracle query. To answer a G oracle query on $(M, i, \sigma) \in \{0,1\}^* \times \mathbb{N} \times D$, $\mathcal{B}$ responds as follows.

1. If there has already been a G oracle query for (M, i, σ), there will be an entry $(M, i, \sigma, z, good, \mu)$ in the G-table. Algorithm $\mathcal{B}$ responds with z. This keeps the oracle consistent if queried multiple times on the same input.
2. If $i < 1$ or $i > n$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} D$ at random. It sets $good \leftarrow \texttt{false}$, $internal \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, good, internal, \mu)$ to the G-table, and responds with z.
3. Otherwise, we have $i \leq i \leq n$. Algorithm $\mathcal{B}$ searches its H-table for entries matching $(M' = M,\ i' = i,\ \mu',\ z',\ good' = \texttt{true},\ internal',\ x')$. There is at most one such entry in the H-table.
   If no matching entry is found, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random. It sets $good \leftarrow \texttt{false}$, $internal \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, good, internal, \mu)$ to the G-table, and responds with z.
   Otherwise, algorithm $\mathcal{B}$ has found a matching entry $(M' = M,\ i' = i,\ \mu',\ z',\ good' = \texttt{true},\ internal',\ x')$ in the H-table. The query now being handled is

relevant to an eventual signature if either (a) $x' \neq \perp$ and $\sigma = x'$ or (b) $x' = \perp$ and $\mathit{Evaluate}(s^*, \sigma) = y^*$. In the latter case, algorithm $\mathcal{B}$ has just learned the solution to the inversion problem it was posed. It stores the solution σ in its global variable x^*.

If the query now being handled is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random. It sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mathit{internal}, \mu)$ to the G-table, and responds with z.

Otherwise, the query now being handled *is* relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random and sets $\mathit{good} \leftarrow \texttt{true}$ and $\mu \leftarrow \mu' \oplus z$. If this G oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{true}$. Otherwise, $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{false}$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mu)$ to the G-table.

So long as $i \neq n$, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query at level $i+1$ before returning. Algorithm $\mathcal{B}$ searches its H-table for an entry matching $(M'' = M,\ i'' = i+1,\ \mu'' = \mu' \oplus z,\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'' = \texttt{false},\ x'')$. If such an entry exists, it was inserted with $\mathit{good} = \texttt{false}$ but should have been inserted with $\mathit{good} = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.

If algorithm $\mathcal{B}$ was not forced to abort, it responds to the hash query with z.

Answering a Signing Oracle Query. To answer a signature query on $M \in \{0,1\}^*$, $\mathcal{B}$ responds as follows.

Otherwise, for each i from 1 to n, algorithm $\mathcal{B}$ searches its H-table for entries of the form $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$. There will be at most one such entry in the H-table for each i. There will be some index I such that for all $i \leq I$ there is a matching entry in the table, and for all $i > I$ there is not. (It's possible that there are no matching entries in the H-table, in which case I is 0.)

For each i from 1 to n, algorithm $\mathcal{B}$ searches its G-table for entries of the form $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$. There will be at most one such entry in the G-table for each i. There will be some index J such that for all $i \leq J$ there is a matching entry in the table, and for all $i > J$ there is not. (It's possible that there are no matching entries in the G-table, in which case J is 0.) It must be the case that either $J = I - 1$ or $J = I$.

If $I = 0$, algorithm $\mathcal{B}$ sets $\sigma_I \leftarrow 0_D$ and $\mu_I \leftarrow 0_E$.

Otherwise, if $I > 0$ and $J = I$, let $(M' = M,\ i' = I,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ be the matching entry in the G-table for $i' = I$, and let $(M'' = M,\ i'' = J,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$. be the matching entry in the G-table for $i'' = J = I$. Algorithm $\mathcal{B}$ sets $\sigma_I \leftarrow \sigma''$ and $\mu_I \leftarrow \mu''$.

Otherwise, we have $I > 0$ and $J = I - 1$. Let $(M' = M,\ i' = I,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ be the matching entry in the G-table for $i' = I$. If $x' = \perp$, algorithm $\mathcal{B}$ does not know how to compute the

next block signature, and must abort. Otherwise, $x' \in D$, and $\mathcal{B}$ sets $\sigma_I \leftarrow x'$. It makes an internal G oracle query for $G(M, i, \sigma_I)$. This query ensures that there is an entry in the G-table of the form ($M'' = M$, $i'' = I$, σ'', z'', $good'' = \texttt{true}$, $internal'' = \texttt{true}$, μ''). Algorithm $\mathcal{B}$ sets $\mu_I \leftarrow \mu''$.

Now algorithm $\mathcal{B}$ repeats the following steps for each i from $I + 1$ to n. It makes an internal H oracle query for $H(M, i, \mu_{i-1})$. This hash query ensures that there is an entry in the H-table of the form ($M' = M$, $i' = i$, $\mu' = \mu_{i-1}$, z', $good' = \texttt{true}$, $internal' = \texttt{true}$, x'). Algorithm $\mathcal{B}$ sets $\sigma_i \leftarrow x'$. (Algorithm $\mathcal{B}$ has set μ_{i-1} to the value that will cause its hash oracle code to create an entry with $good' = \texttt{true}$ and $x' \neq \perp$.) Algorithm $\mathcal{B}$ makes an internal G oracle query for $G(M, i, \sigma_i)$. This query ensures that there is an entry in the G-table of the form ($M'' = M$, $i'' = i$, σ'', z'', $good'' = \texttt{true}$, $internal'' = \texttt{true}$, μ''). Algorithm $\mathcal{B}$ sets $\mu_i \leftarrow \mu''$.

When the loop has finished, $\mathcal{B}$ returns (σ_n, μ_n) as the answer to the signature query.

Handling the Claimed Forgery. Finally, algorithm $\mathcal{A}$ halts and emits a message $M^* \in \{0,1\}^*$ and a claimed signature forgery σ^* on M. Algorithm $\mathcal{A}$ must not have made a signing oracle query on message M^*, or the forgery would be trivial. Algorithm $\mathcal{B}$ attempts to parse σ as $(\sigma_n^*, \mu_n^*) \in D \times E$. If parsing succeeds, $\mathcal{B}$ attempts to verify the signature.

Algorithm $\mathcal{B}$ repeats the following steps for each i from n down to 1. Algorithm $\mathcal{B}$ checks that the G-table includes an entry of the form ($M'' = M^*$, $i'' = i$, $\sigma'' = \sigma_i^*$, z'', $good'' = \texttt{true}$, $internal'' = \texttt{false}$, μ''). If there is no such entry, $\mathcal{A}$ must not have queried its G oracle at $G(M^*, i, \sigma_i^*)$. Algorithm $\mathcal{B}$ declares failure and aborts. Otherwise, $G(M^*, i, \sigma_i^*) = z''$. Algorithm $\mathcal{B}$ sets $\mu_{i-1}^* \leftarrow \mu_i^* \oplus z''$. Algorithm $\mathcal{B}$ then checks that the H-table includes an entry of the form ($M' = M^*$, $i' = i$, $\mu' = \mu_{i-1}^*$, z', $good' = \texttt{true}$, $internal' = \texttt{false}$, x'). If there is no such entry, $\mathcal{A}$ must not have queried its H oracle at $H(M^*, i, \mu_{i-1}^*)$. Algorithm $\mathcal{B}$ declares failure and aborts. Otherwise, $H(M^*, i, \mu_{i-1}^*) = z'$. Algorithm $\mathcal{B}$ sets $\sigma_{i-1}^* \leftarrow \mathit{Evaluate}(s, \sigma_i^*) - z'$ and continues to the next loop iteration.

Assuming it completes the loop without aborting, $\mathcal{B}$ can tell whether the claimed forgery σ^* is valid by checking whether $\sigma_0^* = 0_D$ and $\mu_0^* = 0_E$.

Finally, $\mathcal{B}$ checks whether one of $\mathcal{A}$'s hash queries revealed the answer to the one-wayness challenge posed to $\mathcal{B}$ by examining its global variable x^*. If that variable still contains $\perp$, $\mathcal{B}$ declares failure. Otherwise $\mathcal{B}$ declares success: x^* is the value such that $\mathit{Evaluate}(s^*, x^*) = y^*$.

Analysis of the Simulator. We now analyze the performance of the simulator $\mathcal{B}$. Suppose that $\mathcal{A}$ makes q_S signing queries and q_H hash queries, and produces a valid forgery with probability ϵ. There are six reasons why $\mathcal{B}$ might fail to break the one-wayness of the trapdoor permutation family Π:

1. Algorithm $\mathcal{B}$ discovers, when handling a H oracle query at level i, that it failed to mark a G oracle query at level i as good.

2. Algorithm $\mathcal{B}$ discovers, when handling a G oracle query at level i, that it failed to mark a H oracle query at level $i+1$ as good.
3. Algorithm $\mathcal{B}$ discovers, when verifying $\mathcal{A}$'s claimed forgery, that $\mathcal{A}$ didn't query the H oracle at $H(M^*, i, \mu^*_{i-1})$ for some i.
4. Algorithm $\mathcal{B}$ discovers, when verifying $\mathcal{A}$'s claimed forgery, that $\mathcal{A}$ didn't query the G oracle at $G(M^*, i, \sigma^*_i)$ for some i.
5. Algorithm $\mathcal{A}$ makes a signature query that $\mathcal{B}$ cannot answer because it would require knowing the preimage of the challenge value y^*.
6. Algorithm $\mathcal{A}$ produces a valid forgery but never made a hash oracle query that revealed the preimage of the challenge value y^*.

We can bound the likelihood of reasons 1 through 4 in a straightforward way. To bound the likelihood of reasons 5 and 6, we will need to recall machinery by Guo et al. [9].

We start by bounding reason 1. Algorithm $\mathcal{B}$ is handling an H oracle query for $H(M, i, \mu)$. For each M and i, there is exactly one value of μ that will trigger a consistency check, with $\mathcal{B}$ examining its G-table for entries matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ good'' = \texttt{false},\ internal'',\ \mu'')$. There is exactly one value of σ'' in such an entry that would cause $\mathcal{B}$ to abort, and that this value depends on a value that isn't in $\mathcal{A}$'s view: either $\sigma'' = x$, with x chosen uniformly at random as part of the query handling, or $Evaluate(s^*, \sigma'') = y^*$, with y^* not previously revealed to $\mathcal{A}$.

Let $\mathcal{Q}_G(M, i)$ be the set of queries algorithm $\mathcal{A}$ makes to its G oracle of the form $G(M, i, \cdot)$, i.e., the set of queries that can add a matching entry to G-table. Then when algorithm $\mathcal{A}$ makes a query $H(M, i, \mu)$ with the one value of μ that triggers a consistency check, the probability that the consistency check will lead $\mathcal{B}$ to abort is at most $|\mathcal{Q}_G(M, i)|/|D|$, and the probability that $\mathcal{B}$ ever needs to abort for reason 1, regardless of how many H oracle queries $\mathcal{A}$ makes, is at most

$$\sum_{(M,i)} \frac{|\mathcal{Q}_G(M, i)|}{|D|} = \frac{1}{|D|} \sum_{(M,i)} |\mathcal{Q}_G(M, i)| \leq \frac{q_G}{|D|}.$$

(Crucially, for each M and i there is only one $H(M, i, \cdot)$ query that can trigger a consistency check, and so $\mathcal{Q}_G(M, i)$ is counted only once.)

We bound reason 2 similarly. Algorithm $\mathcal{B}$ is handling a G oracle query for $G(M, i, \sigma)$. For each M and i, there is exactly one value of σ that will trigger a consistency check, with $\mathcal{B}$ examining its H-table for entries matching $(M'' = M,\ i'' = i+1,\ \mu'',\ z'',\ good'' = \texttt{false},\ internal'' = \texttt{false},\ x'')$.

There is exactly one value of μ'' in such an entry that would cause $\mathcal{B}$ to abort, and that this value depends on a value, z, that is chosen uniformly at random as part of the query handling and isn't in $\mathcal{A}$'s view.

Let $\mathcal{Q}_H(M, i)$ be the set of queries algorithm $\mathcal{A}$ makes to its H oracle of the form $G(M, i, \cdot)$, i.e., the set of queries that can add a matching entry to G-table. Then when algorithm $\mathcal{A}$ makes a query $G(M, i, \sigma)$ with the one value of σ that triggers a consistency check, the probability that the consistency check will lead $\mathcal{B}$ to abort is at most $|\mathcal{Q}_G(M, i+1)|/|E|$, and the probability that $\mathcal{B}$ ever needs

to abort for reason 1, regardless of how many G oracle queries $\mathcal{A}$ makes, is at most

$$\sum_{(M,i)} \frac{|\mathcal{Q}_H(M,i+1)|}{|E|} = \frac{1}{|E|} \sum_{(M,i)} |\mathcal{Q}_H(M,i+1)| \leq \frac{q_H}{|E|}.$$

To bound reason 3, we observe that if $\mathcal{A}$ did not query its H oracle at $H(M^*, i, \mu^*_{i-1})$ for some i then the value of $H(M^*, i, \mu^*_{i-1})$ is uniformly random and independent of $\mathcal{A}$'s view, and therefore so is the correct value for σ^*_i. We have already shown (in Sect. 4.2) that only one signature on M^* will verify as valid); it follows that only one value of σ^*_i will be valid as part of a signature. The value chosen (implicitly) by algorithm $\mathcal{A}$ is correct with probability $1/|D|$.

Similarly, to bound reason 4, we observe that if $\mathcal{A}$ did not query its G oracle at $H(M^*, i, \sigma^*_i)$ for some i then the value of $G(M^*, i, \sigma^*_i)$ is uniformly random and independent of $\mathcal{A}$'s view, and therefore so is the correct value for μ^*_i. We have already shown (in Sect. 4.2) that only one signature on M^* will verify as valid); it follows that only one value of μ^*_i will be valid as part of a signature. The value chosen (implicitly) by algorithm $\mathcal{A}$ is correct with probability $1/|E|$.

If $\mathcal{A}$ produces a valid forgery in an ϵ fraction of runs when run in the unforgeability experiment, it will produce a valid forgery without inducing a reason-1 through 4 abort in at least an $\epsilon - (q_G + 1)/|D| - (q_H + 1)/|E|$ fraction of runs under $\mathcal{B}$.

Now suppose that $\mathcal{A}$, running under $\mathcal{B}$, produces a valid forgery without inducing a reason-1 through 4 abort. We review the H-table and G-table maintained by $\mathcal{B}$. For each i, $1 \leq i \leq n$, we define the set $\mathcal{M}_i$ as follows: a message $M \in \{0,1\}^*$ is included in $\mathcal{M}_i$ if there is an entry ($M' = M$, $i' = i$, μ', z', $good' = \texttt{true}$, $internal' = \texttt{false}$, x') in the H-table, for some μ', z', and x'. We further define the set $\mathcal{M}_{n+1}$ as follows: a message $M \in \{0,1\}^*$ is included in $\mathcal{M}_{n+1}$ if there is an entry ($M'' = M$, $i'' = n$, σ'', z'', $good'' = \texttt{true}$, $internal'' = \texttt{false}$, μ'') in the G-table, for some σ'', z'', and μ''.

Only an H oracle query made by $\mathcal{A}$ can cause a message M to be included in $\mathcal{M}_i$ for some $i \leq n$ (because $internal' = \texttt{false}$), and each H oracle query made by $\mathcal{A}$ can add only one entry to the H-table. We therefore know that $\sum_{i=1}^{n} |\mathcal{M}_i| \leq q_H$. Only a G oracle query made by $\mathcal{A}$ can cause a message M to be included in $\mathcal{M}_{n+1}$ (because $internal'' = \texttt{false}$), and each G oracle query made by $\mathcal{A}$ can add only one entry to the G-table.

In handling a G oracle query $G(M, i, \cdot)$ with $1 \leq i \leq n$, algorithm $\mathcal{B}$ will add an entry to the G-table with $good = \texttt{true}$ only if it finds a corresponding entry in the H-table with $M' = M$, $i' = i$, and $good' = \texttt{true}$. In handling an H oracle query $H(M, i, \cdot)$ with $2 \leq i \leq n$, algorithm $\mathcal{B}$ will add an entry to the H-table with $good = \texttt{true}$ only if it finds a corresponding entry in the G-table with $M' = M$, $i' = i - 1$, and $good' = \texttt{true}$. We therefore know that $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Finally, since $\mathcal{B}$ produced a valid forgery on some message M^* but did not induce a reason-2 abort, we know that $M^* \in \mathcal{M}_i$ for all i and, in particular, that $M^* \in \mathcal{M}_{n+1}$. It follows that $|\mathcal{M}_{n+1}| > 0$.

We now restate two lemmas from Guo et al. [9].

Lemma 1 (Range Lemma [9]). *Let q and n be positive integers, and let $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ be sets satisfying $|\mathcal{M}_1| < q$, $|\mathcal{M}_{n+1}| > 0$, and $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Then there exists an integer $i^* \in [1, n]$ such that*

$$|\mathcal{M}_{i^*}| < q^{(n+1-i^*)/n} \qquad \text{and} \qquad |\mathcal{M}_{i^*+1}| \geq q^{(n-i^*)/n}$$

Lemma 2 (Probability Lemma [9]). *Let q and n be positive integers, and let $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ be sets satisfying $|\mathcal{M}_1| < q$, $|\mathcal{M}_{n+1}| > 0$, and $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Fix some arbitrary ordering on the elements of each set $\mathcal{M}_i$. Then if an integer c^* is chosen uniformly at random from the set $[1, n]$ and an integer k^* is chosen uniformly at random from the set $\left[1, q^{(n+1-c^*)/n}\right]$, the probability that the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$ is at least $1/\left(nq^{1/n}\right)$.*

For the proofs of these lemmas, see Guo et al. [9, Section 4]. Note that the Probability Lemma follows from the Range Lemma: We have $c^* = i^*$ with probability $1/n$, and conditioned on $c^* = i^*$ the probability that a message from $\mathcal{M}_{c^*}$ with index $k \in \left[1, q^{(n+1-c^*)/n}\right]$ is also in $\mathcal{M}_{c^*+1}$ is at least

$$\frac{|\mathcal{M}_{c^*+1}|}{\left|[1, q^{(n+1-c^*)/n}]\right|} \geq \frac{q^{(n-c^*)/n}}{q^{(n+1-c^*)/n}} = \frac{1}{q^{1/n}}.$$

Setting $q = q_H + 1$, we have already shown that the sets $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ defined by the hash table maintained by $\mathcal{B}$ satisfy the preconditions of the Probability Lemma in the case that $\mathcal{A}$ produces a valid forgery without inducing a reason-1 through 4 abort.

When algorithm $\mathcal{B}$ starts running, it chooses c^* uniformly at random from the set $[1, n]$ and k^* uniformly at random from the set $\left[1, q_S^{(n+1-c^*)/n}\right]$. It embeds the challenge in the k^*th entry in $\mathcal{M}_{c^*}$, where the arbitrary ordering on the elements of the sets $\mathcal{M}_{c^*}$ is the order of $\mathcal{A}$'s H oracle queries that cause elements to be added to $\mathcal{M}_{c^*}$.

In handling a signing query on a message M, algorithm $\mathcal{B}$ will find in its H-table an entry of the form ($M' = M$, $i' = i$, μ', z', $good' = \texttt{true}$, $internal' = \texttt{false}$, x') for each i, $1 \leq i \leq I$, and will find in its G-table an entry of the form ($M'' = M$, $i'' = i$, σ'', z'', $good'' = \texttt{true}$, $internal'' = \texttt{false}$, μ'') for each i, $1 \leq i \leq J$, where $I < n$ and J equals either $I-1$ or I. In the case that $I = J = n$, algorithm $\mathcal{B}$ has all the information it needs to answer the signing query. In any other case, it will make internal H oracle and G oracle queries in handling the signing query. These internal queries will add entries to the H-table and G-table, respectively, with $good = \texttt{true}$ and $internal = \texttt{true}$, which will prevent the later addition of entries with $good = \texttt{true}$ and $internal = \texttt{false}$.

In the notation we have just introduced, any message M submitted by $\mathcal{A}$ to its signing oracle will end up a member of $\mathcal{M}_1$ through $\mathcal{M}_I$ and, if $I = J = n$, of $\mathcal{M}_{n+1}$. A signing query on message M will induce a reason-5 abort if $J = I - 1$ and the H-table entry with $M' = M$ and $i' = I$ has $x' = \perp$. This can happen only if $\mathcal{B}$ chose to embed its inversion challenge in the response to a level-I H oracle query for the message M, but $\mathcal{A}$ did not then make a corresponding

level-I G oracle query for the message M, revealing the solution to the inversion challenge. And, provided that $I < n$, if $\mathcal{A}$ did not make a level-I G oracle query for the message M it also did not make a level-$(I+1)$ H oracle query for the message M. But $\mathcal{B}$ embeds its inversion challenge in the k^*th entry in $\mathcal{M}_{c^*}$. Put another way: If the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$ then no signing oracle query will induce a reason-5 abort.

Finally, a reason 6 abort will occur if none of $\mathcal{A}$'s G oracle queries revealed to $\mathcal{B}$ the solution to the challenge it embedded in M, the k^*th message in $\mathcal{M}_{c^*}$. If $c^* = n$, then the presence of M in $\mathcal{M}_{n+1}$ guarantees that $\mathcal{A}$ made a level-n G oracle query for M, revealing the solution. If $c^* < n$, then the presence of M in $\mathcal{M}_{c^*+1}$ guarantees that $\mathcal{A}$ made a level-(c^*+1) H oracle query for M, which means it must also have made a level-c^* G oracle query for M, likewise revealing the solution. Put another way: If the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$, then $\mathcal{A}$ will not be forced into a reason-6 abort.

Under Guo et al.'s Probability Lemma, then, assuming that $\mathcal{A}$, running under $\mathcal{B}$, produces a valid forgery without inducing a reason-1 through 4 abort, no signing oracle query will induce a reason-5 abort and $\mathcal{B}$ will not be forced into a reason 6 abort with probability at least $1/\left(nq^{1/n}\right)$, where $q = q_H + 1$. Thus if $\mathcal{A}$ produces a valid forgery in an ϵ fraction of runs when run in the unforgeability experiment, $\mathcal{B}$ will use $\mathcal{A}$ to solve a one-wayness challenge in at least an $\left(\epsilon - (q_G+1)/|D| - (q_H+1)/|E|\right) \;/\; \left(n(q_H+1)^{1/n}\right)$ fraction of runs. Accounting for the running time incurred by $\mathcal{B}$ in answering $\mathcal{A}$'s oracle queries, we have proved the following theorem:

Theorem 1. *Let Π be a (t', ϵ')-one-way trapdoor permutation family over domain D. Then the short signature scheme on Π is $(t, q_H, q_G, q_S, \epsilon)$-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq n(q_H+1)^{1/n}\epsilon' + \frac{q_G+1}{|D|} + \frac{q_H+1}{|E|} \qquad \text{and} \qquad t \leq t' - O(q_H + q_G + nq_S).$$

5 Instantiation with RSA

Lysyanskaya et al. explain how to use the RSA function to create a certified trapdoor permutation [12]. As usual, the permutation description consists of a modulus $N = pq$, where p and q are two large primes, along with e, relatively prime to $\varphi(N) = (p-1)(q-1)$. The trapdoor is $d = e^{-1} \mod \varphi(N)$. The domain D is $\mathbb{Z}/N\mathbb{Z}$, and the permutation is evaluated as

$$\pi\colon x \mapsto x^e \mod N \qquad \text{and} \qquad \pi^{-1}\colon y \mapsto y^d \mod N.$$

Two challenges remain.

First, we need $\pi(\cdot)$ to be a permutation of all of $\mathbb{Z}/N\mathbb{Z}$, even if N and e were maliciously generated. Lysyanskaya et al. extend $\pi(\cdot)$ as

$$\pi(x) = \begin{cases} x^e \bmod N & \text{if } x \text{ is relatively prime to } N \\ x & \text{otherwise} \end{cases}$$

and extend $\pi^{-1}(\cdot)$ similarly. If we knew that e is relatively prime to $\varphi(N)$, we would be done. Unfortunately, this property is not easy to verify given just N and e — indeed, it is intractable to verify when $e < N^{1/4}$, assuming the phi-hiding assumption holds, a fact used by Kakvi and Kiltz to build RSA full-domain hash signatures with tight security reduction [10]. Lysyanskaya et al., following Micali, Ohta, and Reyzin [13] and Cachin, Micali, and Stadler [5], propose to require that e prime and larger than N, properties that are easy to check and that are sufficient to guarantee that the extended $\pi(\cdot)$ above is a permutation. The downside to such large e is that applying $\pi(\cdot)$ takes time linear in $\log e$, which is why $e = 65537$ is frequently chosen in other applications of RSA.

Kakvi, Kiltz, and May observed that for prime e in the range $N^{\frac{1}{4}+\epsilon} < e < N$ it is possible to use Coppersmith's method to certify that e is relatively prime to $\varphi(N)$ in time $O(\epsilon^{-8} \log^2 N)$ [11]. Where verifiers are expected to process many signatures for each signing key they encounter, using smaller e and Kakvi-Kiltz-May certification instead of $e > N$ should reduce overall running time.

Second, we must pick the group operation for the domain $D = \mathbb{Z}/N\mathbb{Z}$. It cannot be $\times$, because, e.g., p has no multiplicative inverse modulo $N = pq$. Instead, following Lysyanskaya et al., we pick $+$ as our group operation.

Finally, we must select a group E. Here the only requirement is that $(q_H + 1)/|E|$ be negligible. It is sufficient for E to consist of bit strings of length twice the security parameter. At the 128-bit security level, for example, elements of E can be 256 bits long.

References

1. Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_10
2. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D., Pyle, R., Ganesan, R., Sandhu, R., Ashby, V. (eds.) Proceedings of CCS 1993, pp. 62–73. ACM Press, November 1993
3. Bellare, M., Yung, M.: Certifying permutations: non-interactive zero-knowledge based on any trapdoor permutation. J. Cryptol. **9**(1), 149–166 (1996)
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. **17**(4), 297–319 (2004)
5. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_28
6. Coron, J.-S.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_14
7. Coron, J.-S.: Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_18
8. Goh, E.-J., Jarecki, S., Katz, J., Wang, N.: Efficient signature schemes with tight reductions to the diffie-hellman problems. J. Cryptol. **20**(4), 493–514 (2007)

9. Guo, F., Chen, R., Susilo, W., Lai, J., Yang, G., Mu, Y.: Optimal security reductions for unique signatures: bypassing impossibilities with a counterexample. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 517–547. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_18
10. Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 537–553. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_32
11. Kakvi, S.A., Kiltz, E., May, A.: Certifying RSA. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 404–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_25
12. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_5
13. Micali, S., Ohta, K., Reyzin, L.: Provable-subgroup signatures. Unpublished manuscript (1999)
14. Neven, G.: Efficient sequential aggregate signed data. IEEE Trans. Inf. Theory **57**(3), 1803–1815 (2011)
15. Orman, H., Hoffman, P.: Determining strengths for public keys used for exchanging symmetric keys. RFC 3766, April 2004
16. Papadopoulos, D., et al.: Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099 (2017). https://eprint.iacr.org/2017/099
17. Verheul, E.R.: Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. J. Cryptol. **17**(4), 277–296 (2004)

Weak-Unforgeable Tags for Secure Supply Chain Management

Marten van Dijk, Chenglu Jin, Hoda Maleki, Phuong Ha Nguyen(✉), and Reza Rahaeimehr

University of Connecticut, Storrs, USA
{marten.van_dijk,chenglu.jin,hoda.maleki,phuong_ha.nguyen, reza.rahaeimehr}@uconn.edu

Abstract. Given the value of imported counterfeit and pirated goods, the need for secure supply chain management is pertinent. Maleki et al. (HOST 2017) propose a new management scheme based on RFID tags (with 2–3K bits NVM) which, if compared to other schemes, is competitive on several performance and security metrics. Its main idea is to have each RFID tag stores its reader events in its own NVM while moving through the supply chain. In order to bind a tag's identity to each event such that an adversary is not able to impersonate the tag's identity on another duplicate tag, a function with a weak form of unforgeability is needed. In this paper, we formally define this security property, present three constructions (MULTIPLY-ADD, ADD-XOR, and S-Box-CBC) having this security property, and show how to bound the probability of successful impersonation in concrete parameter settings. Finally, we compare our constructions with the light-weight hash function PHOTON used by Maleki et al. in terms of security and circuit area needed. We conclude that our ADD-XOR and S-Box-CBC constructions have approximately $1/4 - 1/3$ of PHOTON's total circuit area (this also includes the control circuitry besides PHOTON) while maintaining an appropriate security level which takes care of economically motivated adversaries.

Keywords: Light-weight cryptography · Unforgeability
One-time hash function · Secure supply chain management

1 Introduction

According to a recent report (in 2016) by the OECD and the EU's Intellectual Property Office [1], the value of imported counterfeited and pirated goods is worth nearly half a trillion dollars a year, which is around 2.5% of global imports with many of the proceeds going to organized crime. Close to 5% of goods that are imported into the European Union are fakes. The report analyses about half a million customs seizures around the world during 2011–13 covering all kinds of physical counterfeit goods (that infringe trademarks, intellectual property

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 80–98, 2018.
https://doi.org/10.1007/978-3-662-58387-6_5

rights, or copyright) in order to obtain rigorous estimates of the scale of trade in counterfeit and pirated goods (online piracy is not included). These fake products appear everywhere – the most dangerous ones are auto parts that fail, drugs making people sick, medical instruments delivering false readings, etc.

It is of utmost importance to make supply chains secure in order to detect counterfeit product injection into the supply chain. To this purpose, Radio-Frequency Identification (RFID) tags are used as a low-cost wireless identification method: Each product is equipped with an RFID tag which has a unique identifier and which is initialized at the supply chain back-end server with corresponding product information. In addition the RFID tag is either initialized with digital keys used for future authentication or, if a Physical Unclonable Function (PUF) is embedded, then read-out 'challenge response pairs' from the PUF at the back-end server can later be used for authentication. A supply chain moves through different supply chain partners that interact with RFID tags using RFID readers. These interactions are collected/stored and are together analyzed at the back-end server in order to detect whether the product is genuine or fake before the product exits the supply chain.

1.1 NVM-Based Scheme

This paper focuses on the most recent state-of-the-art proposal by Maleki et al. [2] for secure supply chain management based on RFID tags. Previous schemes come in two kinds: A first kind [3–7] requires persistent online communication between readers of supply chain partners and the back-end server. This, however, is in practice not always possible; sometimes the online connection does get lost and even an hour communication disruption can delay product transportation leading to financial loss. In order to avoid the need for persistent online communication, a second kind of scheme has been proposed which requires each supply chain partner to implement a local database [8–11]. Local databases are used as temporal storage to keep track of reader events. The local databases are integrated into the back-end server at a suitable time when an online connection with the back-end server is available.

The disadvantage of using local databases is that they need a reliable infrastructure and they must be maintained and secured. This imposes extra costs to partners and makes the supply chain possibly less secure as these local databases become an accessible point of attack. The main contribution of [2] is a new method which does not require any persistent online communication and also does not require local databases. Their idea is to distribute the local databases into the RFID tags themselves by utilizing the 2–3K bit Non-Volatile Memory (NVM) present in current off-the-shelve RFID tags [12]. This memory is sufficient for storing all the reader events the RFID tag engages in. Only when exiting the supply chain, the RFID tag is read out and verified by the back-end server for which online communication is needed (a minimal requirement for any scheme). The NVM-based scheme of Maleki et al. [2] is presented in detail in Appendix A (with discussion of pros and cons).

1.2 Software Unclonable Functions

The main take-away of the NVM-based scheme is that in order to bind the identity of an RFID tag to a reader event, the tag consumes one of its secret keys k stored in its NVM in order to compute $F_k(x)$ where input x is received from the reader and cannot be distinguished from a random bit string. The tag overwrites k with $F_k(x)$ in its NVM – and in this way the tag authenticates its own reader events stored in its NVM. For completeness, the reader represents the reader event (which includes the identities of the reader and tag, and a time stamp) as a bit string, which the reader MACs using its own key and this results in x. The back-end server has in its database the key sequence of the tag (which includes k) and the reader key; this is sufficient to verify the binding of the event to the reader (through the MAC) and tag (through $F_k(x)$).

Maleki et al. [2] explain that it is sufficient to require that $F_k(.)$ is a collision resistant hash function – and they propose to use PHOTON 80/20/16, a light-weight hash function costing 865 GE (Gate Equivalent). The complete solution (including control logic etc.) costs 1428 GE. Juels and Weis [13] state (and confirmed by [12]) "A basic RFID tag may have a total of anywhere from 1000–10000 gates, with only 200–2000 budgeted specifically for security." Also every 1000 gates costs approximately one dollar cent per tag. It is therefore important to further reduce the gate equivalence of the complete solution. It turns out that collision resistance is not required and this allows the design of a much more light-weight $F_k(.)$, which is the problem statement of this paper.

In the NVM-based scheme, $F_k(.)$ is used to protect against an adversary who can only access the tag through its read and write interface in order to gather sufficient information to construct a 'tag-simulator' which can be programmed into a fake tag. The read and write interface is such that after initialization only the values that have replaced keys in NVM can be read out. This means that in order to learn about one specific key k, an adversary can engage in a reader-like interaction with the tag in order to replace k with $F_k(x')$ for some value x' of his choice. Next $F_k(x')$ can be read out and the pair $(x', F_k(x'))$ can be used to design a simulator for predicting $F_k(x)$ for random input bit vectors x. Modeling this (very) weak attacker leads to the definition of "software unclonable functions" in [2] of which they only give collision resistant hash functions as an instance. We notice that in the discussion above the adversary is only allowed to use an RFID tag's read and write interface according to its specifications. An adversary who can circumvent the interface circuitry by means of a physical attack is not considered.

1.3 Contributions and Organization

In Sect. 2 we take the definition of software unclonable functions and give an equivalent definition in terms of a "software unclonable response game". Next we discuss its relation to the standard crypto notion of unforgeability for MACs and we conclude that being software unclonable means "unforgeable for random inputs when given one chosen input-output pair" – hence, the title of our paper.

We enrich the definition of software unclonability by adding a security measure for worst-case scenarios in concrete parameter settings.

Sections 3, 4, and 5 provide very light-weight constructions. The first, called MULTIPLY-ADD, is based on a simple multiplication with addition over integers. The second, called ADD-XOR, combines xor with addition-with-carry over binary vectors. The third, called S-Box-CBC, uses the idea of Cipher Block Chaining (CBC) mode with a specially designed S-box. In Sect. 6 we compare the different solutions and we show ADD-XOR and S-Box-CBC lead to dramatic reductions in total circuit size for reasonable security.

2 Software Unclonable Functions

Software unclonable functions are defined as follows:

Definition 1. *[2] A keyed function $F_k(.)$ is called software unclonable if the probability of guessing the output of $F_k(x)$ for a randomly chosen input x with knowledge of* **one** *chosen input-output pair $(x', F_k(x'))$ (the adversary chooses x') is less than negligible (in the function's key size).*

We notice that Definition 1 requires resistance against adversaries with unbounded computation – the definition formulates software unclonability in terms of information theoretic security. This implies that a symmetric key encryption scheme may not satisfy software unclonability since one chosen plaintext ciphertext pair may reveal significant information about the underlying secret key in the information theoretical setting. For a polynomial time adversary a semantically secure symmetric key encryption scheme will be software unclonable. Therefore, we recast the above definition for adversaries with polynomial computation by using the software unclonable response game given in Algorithm 1, where

- λ is a security parameter and $Gen(1^\lambda)$ is a ppt algorithm which generates a random key k,
- $\mathcal{A}_0$ is a ppt adversarial algorithm which allows the adversary to generate exactly one chosen input value x'
- for which the adversary is allowed to learn the output/response of function $F_k(x')$,
- $\mathcal{A}_1$ is a ppt adversarial algorithm which takes just this one input-output pair $(x', F_k(x'))$ in order to produce a ppt simulator $\mathcal{S}$, and
- where $\mathcal{S}$ successfully predicts $F_k(x)$ for a random input x if it outputs $F_k(x)$.

This leads to the following definition which we will use in our analysis in next sections:

Definition 2. *A function $F_k(x)$ is software unclonable if for any ppt pair $(\mathcal{A}_0, \mathcal{A}_1)$, the probability that* SoftwareUnclRespGame*$(\mathcal{A}_0, \mathcal{A}_1)$ (see Algorithm 1 with security parameter λ) returns 1 is $negl(\lambda)$.*

Algorithm 1. Software unclonable response game

function SoftwareUnclRespGame($\mathcal{A}_0, \mathcal{A}_1$)
  $k \leftarrow \text{Gen}(1^\lambda)$;
  $x' \in \{0,1\}^\lambda \leftarrow \mathcal{A}_0(1^\lambda)$;
  $\mathcal{S} \leftarrow \mathcal{A}_1(x', F_k(x'))$; /* $\mathcal{S}$ is a ppt algorithm */
  Random $x \in \{0,1\}^\lambda$
  if $F_k(x) \leftarrow \mathcal{S}(x)$ **then** $b = 1$; **else** $b = 0$; **end if**
  Return b;
end function

Now, based on standard cryptography there are numerous cryptographic functions which are software unclonable. Given our motivation we are only interested in light-weight solutions in the sense that at most a couple 100 gates should suffice for implementation. This implies that the primitive (as far as the authors know) cannot be based on a computational hardness assumption. And this means that we need to prove information theoretical security for our constructions after all (in Definition 2 algorithms $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{S}$ do not need to be restricted to ppt). As a consequence, since the adversary can use one λ-bit vector equation representing an input-output pair for construction of a simulator, the key must have size at least $\lambda + O(\lambda)$ (as opposed to a symmetric key encryption scheme which can have a λ-bit secret key).

2.1 Unforgeability

A closer look at Definition 2 shows the relation of a software unclonable function to a Message Authentication Code (MAC): A MAC is a triple $(Gen, Sign, Ver)$ of ppt algorithms where $k \leftarrow Gen(1^\lambda)$ with security parameter λ, $t \leftarrow Sign(k, x)$ produces a tag t for input string x with key k, and Ver verifies whether a tag t fits input x with key k. A software unclonable function $F_k(x)$ plays the role of producing tags as in $Sign$. A first small difference is that $F_k(.)$ is a function and not an algorithm. This implies that verification of the tag is straightforward in that the corresponding Ver simply verifies whether tag $t = F_k(x)$ (and this directly implies the correctness property for a MAC).

The security of a MAC is defined as follows: A MAC is unforgeable if for all ppt algorithms $\mathcal{A}$,

$$Prob\left(\begin{array}{l} k \leftarrow Gen(1^\lambda), \\ (x,t) \leftarrow \mathcal{A}^{Sign(k,.)}(1^\lambda) \text{ where } \mathcal{A} \text{ does not query } Sign(k,x), \\ Ver(k,x,t) = \text{accept} \end{array}\right) < negl(\lambda),$$

where $\mathcal{A}^{Sign(k,.)}$ denotes that $\mathcal{A}$ has access to oracle $Sign(k,.)$.

The second difference with software unclonability is that the adversarial algorithm is split into $\mathcal{A}_0$, which selects an x' for querying oracle $Sign(k,.) = F_k(.)$, and $\mathcal{A}_1$ which, based on the output of the queried oracle, produces a simulator $\mathcal{S}$ whose goal is to produce a valid (verifiable) tag $Sign(k,x) = F_k(x)$ for

some random input x. This means that software unclonability does not allow the adversary to adaptively choose an x for which a tag is constructed, instead unforgeability is for tags of random inputs.

The third difference is that software unclonability does not allow the adversary to have a polynomial number of queries to the oracle, instead only one chosen input-tag pair can be used. We conclude that a software unclonable function produces tags as in a MAC with a much weaker unforgeability property: a tag corresponding to a random input is unforgeable given only one chosen input-tag pair. The motivation presented in the introduction has led to the name "software unclonable". From a crypto perspective, however, a better terminology would be "unforgeable for random inputs when given one chosen input-output pair" and call the primitive a "one-time MAC for authenticating random (not chosen) inputs."

In a one-time MAC [14] a key is used at most once and can be constructed using a universal hash function which is pairwise independent. An example lightweight pairwise independent hash function is defined by tag $t = k_0x + k_1 \bmod p$, where p is prime. In Sect. 3 we analyse the "MULTIPLY-ADD" function where a tag is computed as $k_0x + k_1 \bmod 2^\lambda$ and the "key" (k_0, k_1) is chosen at random (not necessarily odd).

2.2 Average vs. Worst-Case Analysis

Software unclonable functions are meant to be applied in RFID-based secure supply chain management. Rather than just proving asymptotic results in the form of the probability of a successful attack being negligible in λ, we want to know a concrete upper bound on this probability as a function of λ. This will allow us to suggest concrete parameter settings.

As we will explain below, Definition 2 talks about the average over 'queries $x' \leftarrow \mathcal{A}_0(1^\lambda)$ to oracle $F_k(.)$' of the probability of a successful prediction by the simulator computed by $\mathcal{S} \leftarrow \mathcal{A}_1(x', F_k(x'))$. In asymptotic terms, if this average is negligible, then it is not possible to have a significant worst-case probability γ_0 of selecting an oracle query which leads to a simulator which also has a significant probability $\geq \gamma_1$ of success, because $\gamma_0\gamma_1$ is at most average p which is negligible,

$$\gamma_0\gamma_1 \leq p. \tag{1}$$

In other words, either γ_0 or γ_1 must be negligible.

In this argument we did not specify the pair (γ_0, γ_1) and we note that there are many possibilities. In the concrete non-asymptotic setting, we want an achievable pair (γ_0, γ_1) for which both the probability γ_0 of having a 'lucky' query as well as the probability γ_1 of succesfull prediction by a simulator originating from 'normal' queries to be equally small: If we find such a pair, then we know that both the worst-case probability γ_0 is small as well as the probability γ_1 of success in the normal case is small. This is not captured by studying the concrete asymptotic behavior of the average as a function of λ.

For example, if $p = 2^{-\lambda}$, then the minimum α of $\max\{\gamma_0, \gamma_1\}$ over all possible/achievable pairs (γ_0, γ_1) could be realized by $\gamma_0 = \gamma_1 = 2^{-\lambda/2}$ which meets

(1) with equality (the argument is in essence the application of the birthday paradox to our problem setting). This leads to a very different concrete parameter setting compared to 'just' considering the average case.

So, we are still not entirely satisfied with Definition 2 when considering concrete parameter settings for the following reason: For $\mathcal{A}_1$ and $x' \in \{0,1\}^\lambda$, let

$$p[\mathcal{A}_1](x') = Prob_{x \leftarrow \{0,1\}^\lambda}(F_k(x) \leftarrow \mathcal{S}(x) | \mathcal{S} \leftarrow \mathcal{A}_1(x')). \tag{2}$$

In Definition 2 the probability $p[\mathcal{A}_0, \mathcal{A}_1]$ that SoftwareUnclRespGame $(\mathcal{A}_0, \mathcal{A}_1)$ returns 1 (over random x and coin flips in $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{S}$) is equal to the average

$$p[\mathcal{A}_0, \mathcal{A}_1] = \sum_{x' \in \{0,1\}^\lambda} Prob(x' \leftarrow \mathcal{A}_0(1^\lambda)) p[\mathcal{A}_1](x'). \tag{3}$$

In Definition 2 we only require that the average $p[\mathcal{A}_0, \mathcal{A}_1]$ should be negligible in λ. As explained above, we need to formulate security in terms of a worst-case analysis. We may ask whether the adversary can be lucky (the worst-case) and somehow select in $\mathcal{A}_0$ a x' which "fits" k well in that $p[\mathcal{A}_1](x')$ is (much) larger than the average $p[\mathcal{A}_0, \mathcal{A}_1]$. In order to analyze this we introduce $\alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ as the probability (over coin flips used in $\mathcal{A}_0$ and $\mathcal{A}_1$) that game SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ produces a simulator $\mathcal{S}$ which correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with probability (over random x and coin flips in $\mathcal{S}$) $\leq 2^{-h}$. We note that

$$\alpha_h[\mathcal{A}_0, \mathcal{A}_1] = \sum_{x' : p[\mathcal{A}_1](x') \leq 2^{-h}} Prob(x' \leftarrow \mathcal{A}_0(1^\lambda)). \tag{4}$$

We want both 2^{-h} small and $\alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ large as this implies that (1) the probability that the adversary is able to construct a "lucky" simulator is equal to $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$, which is small, and (2) when the adversary constructs a "normal" (i.e., "not lucky") simulator, then the simulator correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with small probability $\leq 2^{-h}$. We can think of $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ as the probability mass of the tail of distribution $p[\mathcal{A}_1](x')$ that describes the lucky scenarios x' for the adversary, i.e., the worst-case scenarios from a security point of view.

We want $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ and 2^{-h} to be in balance and this leads to the definition of

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min_h \max\{1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1], 2^{-h}\}. \tag{5}$$

$\alpha[\mathcal{A}_0, \mathcal{A}_1]$ is the smallest value α with the property that the probability that game SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ produces a simulator $\mathcal{S}$ which correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with probability $> \alpha$ is at most α, in formula,

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min \left\{ \alpha : Prob \left(\begin{array}{l} \mathcal{S} \leftarrow \mathcal{A}_1(x') \text{ such that} \\ Prob(F_k(x) \leftarrow \mathcal{S}(x)) > \alpha \end{array} \right) \leq \alpha \right\},$$

where the inner probability is over random $x \leftarrow \{0,1\}^\lambda$ and the outer probability is over $x' \leftarrow \mathcal{A}_0(1^\lambda)$.

Definition 3. *For a software unclonable function $F_k(x)$ we define the 'average exponential growth factor'*

$$\mathbf{p} = \limsup_{\lambda\to\infty} -(\log \sup_{(\mathcal{A}_0,\mathcal{A}_1)} p[\mathcal{A}_0,\mathcal{A}_1])/\lambda$$

and we define the 'worst-case exponential growth factor'

$$\mathbf{a} = \limsup_{\lambda\to\infty} -(\log \sup_{(\mathcal{A}_0,\mathcal{A}_1)} \alpha[\mathcal{A}_0,\mathcal{A}_1])/\lambda,$$

where $p[\mathcal{A}_0,\mathcal{A}_1]$ is a function of λ given by (2, 3) and $\alpha[\mathcal{A}_0,\mathcal{A}_1]$ is a function of λ given by (2, 4, 5).

A software unclonable function $F_k(x)$ has better security if $\mathbf{a}$ is larger, and is more light-weight if the gate equivalence of its circuit implementation is smaller. In this paper we propose three constructions and compare them along these metrics.

Notice that by (3, 4), $p[\mathcal{A}_0,\mathcal{A}_1] \geq (1-\alpha_h[\mathcal{A}_0,\mathcal{A}_1])2^{-h}$. Combined with (5), this implies

$$\alpha[\mathcal{A}_0,\mathcal{A}_1]) \leq \min_h \max\{p[\mathcal{A}_0,\mathcal{A}_1]/2^{-h}, 2^{-h}\} = \sqrt{p[\mathcal{A}_0,\mathcal{A}_1]}$$

(since h can be any real number; in the analysis of our constructions we consider only integers h). This proves that the exponential growth factors $\mathbf{p}$ and $\mathbf{a}$ satisfy $\mathbf{a} \geq \mathbf{p}/2$. This argument is in essence the birthday paradox.

In the next sections we analyze for several candidate software unclonable functions both $p[\mathcal{A}_0,\mathcal{A}_1]$ as well as $\alpha[\mathcal{A}_0,\mathcal{A}_1]$ together with their exponential growth factors. It turns out that for these functions the probability mass of the tail of distribution $p[\mathcal{A}_0](x')$ is large so that $\mathbf{a} \approx \mathbf{p}/2$.

3 MULTIPLY-ADD

Below we prove that the "MULTIPLY-ADD" function

$$F_{(k_0,k_1)}(x) = k_0 x + k_1 \bmod 2^\lambda, \text{ for } k_0, k_1, x \in \{0,1\}^\lambda, \tag{6}$$

where k_0 and x are multiplied modulo 2^λ and $+$ modulo 2^λ is binary addition with carry truncated after λ bits, is a software unclonable function. In what follows when we write $+$ we mean addition modulo 2^λ.

Theorem 1. *For the MULTIPLY-ADD function defined in (6),*

$$p[\mathcal{A}_0,\mathcal{A}_1] \leq (\lambda+2)2^{-\lambda-1} \text{ and } \alpha[\mathcal{A}_0,\mathcal{A}_1] \leq 2^{-\lfloor(\lambda-1)/2\rfloor}$$

for all algorithm pairs $(\mathcal{A}_0,\mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). These upper bounds can be met with equality, this implies average and worst-case exponential growth factors

$$\mathbf{p} = 1 \text{ and } \mathbf{a} = 1/2.$$

Proof. We will first translate the problem of creating a simulator with maximum possible successful prediction probability to an equivalent problem which we are able to analyze precisely:

Suppose the adversary knows the pair $(x', F_{(k_0,k_1)}(x'))$ and wants to built a simulator which predicts $F_{(k_0,k_1)}(x)$ for random x. We notice that for

$$z = F_{(k_0,k_1)}(x) - F_{(k_0,k_1)}(x'), \quad v = k_0, \text{ and } w = x - x',$$

$$z = vw \bmod 2^\lambda. \tag{7}$$

Also, notice that given x', since x is random, w is random; and since k_0 is random, v is random. These observations can be used to show that predicting $F_{(k_0,k_1)}(x)$ for a randomly selected input x based on $(x', F_{(k_0,k_1)}(x'))$ where F is defined by (6) is equivalent to (notice that k_1 is unknown and random) predicting z in (7) for a randomly selected input w and unknown/random v. This implies that the probability of SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returning 1 is equal to the probability of ZWGameMA$(\mathcal{A})$, see Algorithm 2, returning 1. This probability is maximized for $\mathcal{A}$ outputting a simulator $\mathcal{S}$ which on input w outputs a z that maximizes $|\{v : z = vw\}|$. In other words, z maximizes the number of collisions v that yield the same $z = vw$. In formula, $p[\mathcal{A}_0, \mathcal{A}_1]$ (where $\mathcal{A}_0$ and $\mathcal{A}_1$ are derived from $\mathcal{A}$ according to the transformation described above) is equal to

$$2^{-\lambda} \sum_w \max_z Prob_{v \leftarrow \{0,1\}^\lambda}[z = vw] = 2^{-\lambda} \sum_w \max_z \frac{|\{v : z = vw\}|}{2^\lambda}. \tag{8}$$

Algorithm 2. Finding z based on w

```
1: function ZWGameMA(A)
2:     v ∈ {0,1}^λ is a random input;
3:     S ← A(1^λ);
4:     w ∈ {0,1}^λ is a random input;
5:     if S(w) = vw then
6:         b = 1;
7:     else
8:         b = 0;
9:     end if
10:    Return b;
11: end function
12: /*On input w, an optimal S outputs a z which maximizes |{v : z = vw}|.*/
```

We will now analyze (8) by distinguishing the cases $w \neq 0$ and $w = 0$.

Let $w \neq 0$. If $2^{\lambda-h}$ is the largest power of 2 dividing w, then $z = vw \bmod 2^\lambda$ is equivalent to $z = v(w/2^{\lambda-h}) \bmod 2^h$. Since $w/2^{\lambda-h}$ is an odd integer, it has an inverse modulo 2^h. This implies that there is exactly one $v = z(w/2^{\lambda-h})^{-1} \bmod 2^h$ for which $z = v(w/2^{\lambda-h}) \bmod 2^h$. Therefore there are $2^{\lambda-h}$ possible v for

which $z = vw \bmod 2^\lambda$ (these v are equal to $z(w/2^{\lambda-h})^{-1} \bmod 2^h$ plus some multiple of 2^h).

If $w = 0$, then only for $z = 0$ there exists a v such that $z = vw$; in this case all 2^λ possible v satisfy $z = vw$.

Let W_h, $1 < h \leq \lambda$, be the number of integers w, $0 < w < 2^\lambda$, for which $2^{\lambda-h}$ is the largest power of 2 dividing w. Define $W_0 = 1$. Then (8) is equal to

$$2^{-\lambda} \sum_{h=0}^{\lambda} W_h 2^{-h}.$$

We notice that $W_h = 2^{h-1}$ for $1 \leq h \leq \lambda$. Hence, (8) is equal to

$$2^{-\lambda}(1 + \sum_{h=1}^{\lambda} 2^{-1}) = (\lambda + 2)2^{-\lambda-1}.$$

This proves $p[\mathcal{A}_0, \mathcal{A}_1] = (\lambda + 2)2^{-\lambda-1}$ and $\mathbf{p} = 1$.

The above derivation also proves that the number of w for which $\max_z Prob_{v \leftarrow \{0,1\}^\lambda}[z = vw] \leq 2^{-h}$ is equal to $\sum_{i=h}^{\lambda} W_i = 2^\lambda - \sum_{i=0}^{h-1} W_h = 2^\lambda - (1 + \sum_{i=1}^{h-1} 2^{i-1})$ for $h \geq 2$. The probability that such a w is selected is equal to $\sum_{i=h}^{\lambda} W_i / 2^\lambda$. This can be interpreted as

$$\alpha_h[\mathcal{A}_0, \mathcal{A}_1] = \sum_{i=h}^{\lambda} W_i / 2^\lambda = 1 - (1 + \sum_{i=1}^{h-1} 2^{i-1})/2^\lambda = 1 - 2^{-(\lambda+1-h)},$$

which in turn proves

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min_h \max\{2^{-(\lambda+1-h)}, 2^{-h}\} = 2^{-\lfloor(\lambda-1)/2\rfloor} \text{ and } \mathbf{a} = 1/2.$$

4 ADD-XOR

Below we prove that the "ADD-XOR" function

$$F_{(k_0,k_1)}(x) = (k_0 + x \bmod 2^\lambda) \oplus k_1, \text{ for } k_0, k_1, x \in \{0,1\}^\lambda, \tag{9}$$

where + modulo 2^λ is binary addition with carry truncated after λ bits and where $\oplus$ represents XOR, is a software unclonable function.

In Appendix B of the full version [15], we prove the following theorem:

Theorem 2. *For the ADD-XOR function as defined in (9),*

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq 2^{-0.234 \cdot \lambda} \text{ and } \alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2 \cdot 2^{-0.141 \cdot \lambda}$$

for all algorithm pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). This implies average and worst-case exponential growth factors

$$\mathbf{p} \geq 0.234 \text{ and } \mathbf{a} \geq 0.141.$$

Simulations with the optimal simulator constructed in [15] show that the bounds for $\mathbf{p}$ and $\mathbf{a}$ are very tight. (Notice that for ADD-XOR, $\mathbf{a} > \mathbf{p}/2$.)

5 S-Box-CBC

In this section we introduce a construction which uses the idea of S-boxes in block-cipher design together with CBC mode [16].

Suppose we have a non-linear mapping

$$S \in \{0,1\}^m \rightarrow \{0,1\}^m,$$

where m is generally very small (we will propose small powers of two, $m = 4$ or $m = 8$). Mapping S is called an S-Box. Since we use a software unclonable function for authentication in the NVM-based supply chain management scheme, this means that the software unclonable function does not necessarily need to be invertible given knowledge of the keys. It turns out that ADD-XOR and MULTIPLY-ADD are invertible; in this section, however, we will construct a non-invertible software unclonable function based on a non-invertible S-box mapping S.

Our construction is iterative following the design principle used in CBC mode for symmetric key encryption (where we replace encryption by our S-box): For n with $nm = \lambda$, we use the vector notation $x = (x_1, \ldots, x_n) \in \{0,1\}^\lambda$ with $x_i \in \{0,1\}^m$. For keys $k_0 = (k_1^0, k_2^0, \ldots, k_n^0)$ and $k_1 = (k_1^1, k_2^1, \ldots, k_n^1)$ and input x, we recursively compute

$$y_{i+1} = S(y_i \oplus x_{i+1} \oplus k_{i+1}^0) \oplus k_{i+1}^1 \tag{10}$$

for $0 \leq i \leq n-1$ with $y_0 = 0$. We define

$$F_{(k_0,k_1)}(x) = y. \tag{11}$$

In the construction we mask input x_i with k_i^0 and we mask the output of the S-box with k_i^1. The S-box is a kind of non-linear obfuscation mapping. Forwarding y_i into the computation of y_{i+1} corresponds to the main design principle used in CBC mode for symmetric key encryption.

Below we will prove (in a couple of steps) that the S-box construction leads to a software unclonable function.

We start with analyzing the average case:

Theorem 3. *Let $F_{(k_0,k_1)}(x)$ be defined by the S-Box-CBC construction in (10, 11) for $\lambda = nm$. For the S-box mapping S used in F, we define*

$$\rho[S](w) = \max_{z \in \{0,1\}^m} |\{v : z = S(v) \oplus S(v \oplus w)\}|/2^m, \text{ and}$$

$$\rho[S] = \sum_{w \in \{0,1\}^m} \rho[S](w)/2^m.$$

Then, for all algorithm pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt),

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq \rho[S]^n \text{ with } \mathbf{p} = -(\log \rho[S])/m.$$

Proof. We will first translate the problem of creating a simulator with maximum possible successful prediction probability to an equivalent problem which we are able to analyze precisely:

Suppose the adversary knows the pair $(x', y' = F_{(k_0,k_1)}(x'))$ and wants to built a simulator which predicts $F_{(k_0,k_1)}(x)$ for random x. We notice that for $z = F_{(k_0,k_1)}(x') \oplus F_{(k_0,k_1)}(x)$ the recursive definition of F in (10, 11) implies $z_{i+1} = y'_{i+1} \oplus y_{i+1} = S(y'_i \oplus x'_{i+1} \oplus k^0_{i+1}) \oplus S(y_i \oplus x_{i+1} \oplus k^0_{i+1})$. If we define $v_{i+1} = y'_i \oplus x'_{i+1} \oplus k^0_{i+1}$, and $w_{i+1} = (y'_i \oplus y_i) \oplus (x'_{i+1} \oplus x_{i+1}) = z_i \oplus (x'_{i+1} \oplus x_{i+1})$, then

$$z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1}). \tag{12}$$

Notice that given x'_{i+1} and y'_i, since k^0_{i+1} is random and y'_i only depends on k^0_j for $j \leq i$, v_{i+1} is random. Therefore, by induction on i, given x', v is random. We also notice that given x'_{i+1} and z_i, since x_{i+1} is random and z_i only depends on x_j for $j \leq i$, w_{i+1} is random. Therefore, by induction on i, given x', w is random.

The above observations can be used to show that predicting $F_{(k_0,k_1)}(x)$ for a randomly selected input x based on $(x', F_{(k_0,k_1)}(x'))$ where F is defined by (10, 11) is equivalent to (notice that k_1 is unknown and random) predicting z in (12) for a randomly selected input w and unknown/random v. This implies that the probability of SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returning 1 is equal to the probability of ZWGameSB$(\mathcal{A})$, see Algorithm 3, returning 1. This probability is maximized over $\mathcal{A}$ outputting a simulator $\mathcal{S}$ which on input w outputs a z that maximizes $|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|$. In other words, z maximizes the number of collisions v that satisfy the same set of equations $z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})$. In formula, $p[\mathcal{A}_0, \mathcal{A}_1]$ (where $\mathcal{A}_0$ and $\mathcal{A}_1$ are derived from $\mathcal{A}$ according to the transformation described above) is equal to

$$\begin{aligned}
&2^{-\lambda} \sum_w \max_z Prob_{v \leftarrow \{0,1\}^\lambda}[\forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})] \\
&= 2^{-\lambda} \sum_w \max_z \frac{|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|}{2^\lambda}. \\
&= 2^{-\lambda} \sum_w \prod_{i=0}^{n-1} \rho[S](w_{i+1}) = \prod_{i=0}^{n-1} \sum_{w_{i+1}} \rho[S](w_{i+1})/2^m = \rho[S]^n.
\end{aligned} \tag{13}$$

This concludes the proof of Theorem 3.

According to Theorem 3, the smaller $\rho[S]$, the larger the average exponential growth factor $\mathbf{p}$. The next lemma shows a lower bound on $\rho[S]$ and describes an S-box $\mathcal{S}$ which meets this lower bound leading to the largest possible $\mathbf{p}$ for the S-Box-CBC construction:

Lemma 1. *(i) For any S-box S, $\rho[S](w) \geq 2/2^m$ for $w \neq 0$. If $w = 0$, then $\rho[S](w) = 1$. As a consequence $\rho[S] \geq (3 - 1/2^{m-1})2^{-m}$. This lower bound can be met with equality: (ii) Let $m \geq 3$. If we represent elements in $\{0,1\}^m$ as finite field elements in $GF(2^m)$ and define $S(x) = x^3$ in $GF(2^m)$, then $\rho[S](w) = 2/2^m$ for $w \neq 0$ and $\rho[S] = (3 - 1/2^{m-1})2^{-m}$.*

Algorithm 3. Finding z based on w

function ZWGAMESB($\mathcal{A}$)
  $v \in \{0,1\}^\lambda$ is a random input;
  $\mathcal{S} \leftarrow \mathcal{A}(1^\lambda)$;
  $w \in \{0,1\}^\lambda$ is a random input;
  if $\forall_i z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})$ **then**
    $b = 1$;
  else
    $b = 0$;
  end if
  Return b;
end function
/*On input w, an optimal $\mathcal{S}$ outputs a z which maximizes $|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|$.*/

Proof. Let $z, v, w \in \{0,1\}^m$ (in the proof of the previous theorem $z, v, w \in \{0,1\}^\lambda$). The first part of the lemma follows immediately from the observation that if $z = S(v) \oplus S(v \oplus w)$ then $v' = v \oplus w$ also satisfies this equation. If $w \neq 0$, then $v \neq v'$ and this shows that if a solution v for $z = S(v) \oplus S(v \oplus w)$ exists, then there also exists a second different solution, hence, $\rho[S](w) \geq 2/2^m$. If $w = 0$, then $S(v) \oplus S(v \oplus w) = S(v) \oplus S(v) = 0$ for all v. As a consequence $\rho[S](0) = 2^m/2^m = 1$ and $\rho[S] \geq 1/2^m + (1 - 1/2^m)2/2^m = (3 - 1/2^{m-1})2^{-m}$.

In the second part of the lemma we take $S(x) = x^3$ where we consider binary vectors $x \in \{0,1\}^m$ to represent elements in $GF(2^m)$. Notice that $\oplus$ becomes addition in $GF(2^m)$. This means that the equation $z = S(v) \oplus S(v \oplus w)$ translates to $z = v^3 + (v + w)^3 = v^2 w + v w^2 + w^3$ in $GF(2^m)$. This is equivalent to

$$wv^2 + w^2 v + (w^3 + z) = 0,$$

a quadratic equation in v for $w \neq 0$ (notice that w^2 does not reduce to a linear expression in w since the irreducible polynomial defining $GF(2^m)$ has degree ≥ 3 for $m \geq 3$). If $w \neq 0$, then the equation becomes $v^2 + wv + (w^2 + zw^{-1}) = 0$ which has at most 2 solutions, hence, $\rho[S](w) = 2/2^m$.

Corollary 1. *For the S-Box-CBC construction in (10, 11) for $\lambda = nm$ and the S-box specified in Lemma 1(ii),*

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq (3 - 1/2^{m-1})^{\lambda/m} 2^{-\lambda} \text{ and } \mathbf{p} = 1 - \frac{\log(3 - 1/2^{m-1})}{m}$$

for all algorithms pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). There exist algorithm pairs for which the upper bound holds with equality.

We prove our result for the worst-case scenario in Appendix C in the full version [15].

Theorem 4. *For the S-Box-CBC construction in (10, 11) for* $\lambda = nm$ *and S-box* $\mathcal{S}$ *specified in Lemma 1(ii),*

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \frac{\lambda(m-1)}{m(2m-1)} 2^{-\frac{(m-1)^2}{m(2m-1)}\lambda} \text{ with } \mathbf{a} = \frac{(m-1)^2}{m(2m-1)},$$

for all algorithms pairs $(\mathcal{A}_0, \mathcal{A}_1)$ *(unrestricted, the theorem does not require* $(\mathcal{A}_0, \mathcal{A}_1)$*, and produced simulators to be ppt). There exist algorithm pairs for which the upper bound is almost tight. Notice that* $\mathbf{a}$ *is only slightly larger than* $\mathbf{p}/2$*.*

6 Comparison

Our theorems state for unbounded adversaries (we prove information theoretical security as opposed to PHOTON which assumes computational hardness)

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2^{-\lfloor(\lambda-1)/2\rfloor} \text{ for MULTIPLY-ADD,}$$
$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2 \cdot 2^{-0.141 \cdot \lambda} \text{ for ADD-XOR, and}$$
$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \frac{\lambda(m-1)}{m(2m-1)} 2^{-\frac{(m-1)^2}{m(2m-1)}\lambda} \text{ for S-Box-CBC.}$$

Table 1 lists for which λ the different constructions give rise to upper bounds approximately equal to 2^{-16}, 2^{-32}, 2^{-40}, and 2^{-64}.

Table 1. Comparison light-weight constructions.

$\alpha[\mathcal{A}_0, \mathcal{A}_1]$	MULT.-ADD	ADD-XOR	S-Box-CBC $m = 4$	S-Box-CBC $m = 8$	PHOTON
$\leq 2^{-16}$	$\lambda = 33$ 1200b NVM $\geq$ 726 GE	$\lambda = 121$ 2960b NVM 372 GE (308+64)	$\lambda = 60$ 1680b NVM 440 GE (277+163)	$\lambda = 48$ 1360b NVM 726 GE (261+465)	N/A
$\leq 2^{-32}$	$\lambda = 65$ 1840b NVM $\geq$ 1430 GE	$\lambda = 243$ **5360**b NVM	$\lambda = 112$ 2640b NVM 478 GE (315+163)	$\lambda = 88$ 2160b NVM 748 GE (283+465)	N/A
$\leq 2^{-40}$	$\lambda = 81$ 2160b NVM $\geq$ 1782 GE	$\lambda = 291$ **6320**b NVM	$\lambda = 140$ **3280**b NVM	$\lambda = 112$ 2640b NVM 764 GE (299+465)	$\lambda = 80$ 1200b NVM 1428 GE [2] (563+865)
$\leq 2^{-64}$	$\lambda = 129$ **3120**b NVM	$\lambda = 461$ **9680**b NVM	$\lambda = 216$ **4720**b NVM	$\lambda = 168$ **3760**b NVM	$\lambda = 128$ 1680b NVM 1795 GE (673+1122)

Since we want to prevent an adversary from successfully cloning an RFID tag in order to tag and insert a counterfeit product into the supply chain, we

are actually fine with very small unconditional collision resistance. Even 2^{-16} is acceptable for the following reason: Our constructions are unconditional secure in that even an adversary with unbounded computational resources cannot create a cloned RFID tag which can do better than answering future reader queries with probability $> 2^{-16}$. This implies that only one out of 64K inserted counterfeit products makes it successfully through the supply chain. This is an economical impractical proposition for the adversary.

One reason to have a higher collision resistance like 2^{-32} is if fake trapdoored products are very cheap and the adversary wants to disrupt consumers of these products, e.g., the military. In this case the adversary is not economically motivated, he simply wants to be able to get a hardware footprint in order to be able to launch future attacks. In this case a collision resistance of 2^{-32} would imply an enormous number (4 billion) fake products needed by the adversary making such an attack quite impractical. When we put our crypto-hat on, we may even want the psychological safe collision resistance of 2^{-64}.

For each solution, it is important to verify whether it fits the 2–3K bit NVM requirement. A reader event is 40 bits [2] with a λ-bit output of the software unclonable function, which will replace one 40-bit key for a one time pad in the NVM based scheme and two λ-bit keys for our software unclonable constructions – while PHOTON only needs one λ-bit key. Since the RFID NVM is assumed to be read out per byte, we will round λ bits up to a multiple of bytes. Hence, for each reader event, a total of $5 + 2\lceil\lambda/8\rceil\}$ bytes in NVM are needed. Following [2], we expect at most 10 reader events per path through the supply chain. This gives a total of $10 \cdot 8 \cdot (5 + 2\lceil\lambda/8\rceil)$ required NVM bits. In bold are indicated which entries violate the 2–3K bit requirement.

Table 1 also compares constructions (that do not violate the 2–3K bit NVM requirement) with respect to the Gate Equivalence (GE) – measured in number of 2-input NAND gates – of their circuit implementations[1]. In the table "261+465" under e.g. the S-Box-CBC entry for $m = 8$ and $\lambda = 48$ indicates that 465 GE is spend on the software unclonable function with its own control logic and another 261 GE is spend on general control logic for a full NVM-based supply chain management scheme implementation; $261 + 465$ GE makes a total of 726 GE. Appendix D in [15] lists and explains the optimal implementations of each construction – we list the implemented results for ADD-XOR and S-Box-CBC, and the estimated lower bound $\geq 22 \cdot \lambda$ GE for the MULTIPLY-ADD construction.

PHOTON [17] has two light-weight variations: [2] uses PHOTON 80/20/16 as software unclonable function. It takes 80 bits input and generates 80 bits output with 40 bits collision resistance considering state-of-the-art attacks[2]. Table 1 also shows PHOTON 128/16/16 with 64 bits collision resistance.

The shaded entries in Table 1 minimize the circuit area given a 2–3K bit NVM constraint.

[1] GE is a metric for comparing the size of hardware implementation regardless of the manufacturing technology.

[2] [2] states 64 bits collision resistance, but this is incorrect.

7 Conclusion

We introduced a formal definition of software unclonable functions and constructed several light-weight options with a rigorous security analysis. Our ADD-XOR and S-Box-CBC ($m = 4$) constructions significantly reduce the circuit size of the implementation of the NVM-based supply chain management scheme of Maleki et al. [2] to 372 GE for $\alpha \leq 2^{-16}$ and 478 GE for $\alpha \leq 2^{-32}$. When compared to PHOTON, S-Box-CBC ($m = 8$) gives the smaller area size of 764 GE for $\alpha \leq 2^{-40}$, while PHOTON 128/16/16 with 1795 GE is the preferred choice for very small $\alpha \leq 2^{-64}$. For an economically motivated adversary, it turns out that $\alpha \leq 2^{-16}$ offers sufficient protection.

A NVM-Based RFID Scheme

The NVM-based scheme of Maleki et al. [2] implements the following steps:

Initialization RFID tag. The NVM of the RFID tag is initialized with a sequence of keys $(k^1, k^2, \ldots, k^u)$ and a pointer $p = 1$. The back-end server stores the RFID identity ID together with the sequence of keys.

Initialization Reader. Each RFID tag reader is initialized with its own key K. The back-end server stores the reader identity together with K.

Reader Event. The RFID tag is read out by a reader of a supply chain partner:

1. The RFID tage transmits its ID to the reader.
2. The reader creates a message m which has the reader identity and time stamp of the event.
3. The reader computes $x = MAC_K(m, ID)$. This binds the reader to the event.
4. The reader transmits (m, x) to the RFID tag.
5. The RFID tag receives (m, x). This triggers the RFID tag to read a next key k^p from its NVM and to increment pointer p by 1. Key k^p is large enough in order to be split up into a first part $k^{p,0}$ and a second part $k^{p,1}$. The tag computes the pair $y^p = (m \oplus k^{p,0}, F_{k^{p,1}}(x))$, where F is a software unclonable function. Since k^p is unique to the RFID tag, the use of function F binds the RFID tag to the event. The key part $k^{p,0}$ serves as a one time pad which prevents traceability.
6. The RFID tag stores y^p at the spot where k^p was stored in NVM.

Exit. When the tag exits the supply chain, its NVM is read out and communicated to the back-end server. I.e., the internal logic only allows NVM to be read out up to but not including the address pointed at by pointer p. This means that the back-end server receives ID together with $(y^1, \ldots, y^{p-1})$. The ID is used to look up the sequence of keys corresponding to the tag. For each y, this allows the server to first reconstruct the messages m, second to extract the corresponding reader identity with key K from its database, third to compute the mac value $x = MAC_K(m, ID)$, fourth to evaluate F on x with the appropriate key, and finally verify that this is part of y. If all checks pass, then the recorded reader events were not impersonated and they can be verified

to correspond to a legitimate path through the supply chain. The server will invalidate the tag for future use in its database.

A detailed explanation, security analysis, and discussion around how to make the scheme reliable with respect to miss reads and miss writes can be found in [2].

A comparison of state-of-the-art schemes over a range of metrics can be found in [18]. Besides being unique in that, unlike any other scheme, the need for persistent online communication or local databases is avoided, the NVM-based scheme also compares well with most competitive other schemes. The only dimension on which the NVM-based scheme scores negatively is its lack of being able to resist physical attack (where a strong adversary attempts to circumvent the read and write interface in order to clone all the keys stored in NVM). We notice that even though the trace based scheme [8] can withstand physical attacks, the scheme cannot distinguish between a fake and legitimate tag which possibly results in significant financial loss. Current PUF based schemes [19–21] are not secure against physical attack because of recent machine learning modeling attacks [22–24] – however, as soon as improved PUF designs will resist these modeling attacks, PUF based schemes will resist physical attacks as opposed to the NVM-based scheme. Inherent to current PUF-based schemes, they do need persistent online communication. Also an improved PUF design will likely lead to a higher gate count than the 500–1000 GE for current PUF-based schemes – and this is where the NVM based scheme performs better as well.

As a final note, Sect. 6 discusses several upper bounds on the collision resistance. Obviously, if the resistance is set to 2^{-32} or 2^{-64}, then creating a cloned or fake RFID tag which successfully passes the supply chain becomes very unlikely. In fact too many counterfeit products labelled with fake RFID tags are needed in order to be successful and this makes such an attack economically infeasible. In the introduction we state "An adversary who can circumvent the interface circuitry by means of a physical attack is not considered." Clearly, the weak link in the NVM-based scheme for high collision resistance will now be its lack of resistance against physical attack.

References

1. OECD/EUIPO: Trade in counterfeit and pirated goods: mapping the economic impact. https://doi.org/10.1787/9789264252653-en
2. Maleki, H., Rahaeimehr, R., Jin, C., van Dijk, M.: New clone-detection approach for RFID-based supply chains. In: Hardware Oriented Security and Trust. IEEE (2017)
3. Shen, J., Choi, D., Moh, S., Chung, I.: A novel anonymous RFID authentication protocol providing strong privacy and security. In: Multimedia Information Networking and Security (MINES), pp. 584–588. IEEE (2010)
4. Ilic, A., Lehtonen, M., Michahelles, F., Fleisch, E.: Synchronized secrets approach for RFID-enabled anti-counterfeiting. In: Demo at Internet of Things Conference (2008)

5. Dimitriou, T.: A lightweight RFID protocol to protect against traceability and cloning attacks. In: Security and Privacy for Emerging Areas in Communications Networks, pp. 59–66. IEEE (2005)
6. Bu, K., Xu, M., Liu, X., Luo, J., Zhang, S.: Toward fast and deterministic clone detection for large anonymous rfid systems. In: Mobile Ad Hoc and Sensor Systems (MASS), pp. 416–424. IEEE (2014)
7. Hsu, C.-H., Wang, S., Zhang, D., Chu, H.-C., Lu, N.: Efficient identity authentication and encryption technique for high throughput RFID system. Secur. Commun. Netw. **9**(15), 2581–2591 (2016)
8. Zanetti, D., Capkun, S., Juels, A.: Tailing RFID tags for clone detection. In: NDSS (2013)
9. Zanetti, D., Fellmann, L., Capkun, S., et al.: Privacy-preserving clone detection for RFID-enabled supply chains. In: RFID, pp. 37–44. IEEE (2010)
10. Kerschbaum, F., Oertel, N.: Privacy-preserving pattern matching for anomaly detection in RFID anti-counterfeiting. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 124–137. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16822-2_12
11. Koh, R., Schuster, E.W., Chackrabarti, I., Bellman, A.: Securing the pharmaceutical supply chain. White Paper, pp. 1–19. Auto-ID Labs. Massachusetts Institute of Technology (2003)
12. EPCglobal: EPC radio-frequency identity protocols generation-2 UHF RFID; specification for RFID air interface protocol for communications at 860 MHZ-960 MHZ. EPCglobal Inc., November 2013
13. Juels, A., Weis, S.A.: Authenticating pervasive devices with human protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_18
14. Simmons, G.J.: Authentication theory/coding theory. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 411–431. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_32
15. van Dijk, M., Jin, C., Maleki, H., Nguyen, P.H., Rahaeimehr, R.: Weak-unforgeable tags for secure supply chain management. IACR Cryptology ePrint Archive: Report 2017/1221 (2017)
16. Dworkin, M.J.: Recommendation for block cipher modes of operation: the CMAC mode for authentication. Special Publication (NIST SP)-800-38B (2016)
17. Guo, J., Peyrin, T., Poschmann, A.: The `PHOTON` family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_13
18. Maleki, H., Rahaeimehr, R., van Dijk, M.: SoK: RFID-based clone detection mechanisms for supply chains. In: Proceedings of the Workshop on Attacks and Solutions in Hardware Security(ASHES), pp. 33–41. ACM (2017)
19. Devadas, S., Suh, E., Paral, S., Sowell, R., Ziola, T., Khandelwal, V.: Design and implementation of PUF-based unclonable RFID ICs for anti-counterfeiting and security applications. In: RFID, pp. 58–64. IEEE (2008)
20. Tuyls, P., Batina, L.: RFID-tags for anti-counterfeiting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 115–131. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_8
21. Ranasinghe, D., Engels, D., Cole, P.: Security and privacy: modest proposals for low-cost RFID systems. In: Auto-ID Labs Research Workshop (2004)
22. Tajik, S., et al.: Physical characterization of arbiter PUFs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 493–509. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_27

23. Becker, G.T.: The gap between promise and reality: on the insecurity of XOR arbiter PUFs. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 535–555. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_27
24. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 237–249 (2010)

Proof of Censorship: Enabling Centralized Censorship-Resistant Content Providers

Ian Martiny[1(✉)], Ian Miers[2], and Eric Wustrow[1]

[1] University of Colorado, Boulder, USA
{ian.martiny,ewust}@colorado.edu
[2] Johns Hopkins University, Baltimore, USA
imiers@cs.jhu.edu

Abstract. Content providers often face legal or economic pressures to censor or remove objectionable or infringing content they host. While decentralized providers can enable censorship-resistant storage, centralized content providers remain popular for performance and usability reasons. But centralized content providers can always choose not to respond to requests for a specific file, making it difficult to prevent censorship. If it is not possible to prevent, is it possible to detect and punish censorship on a centralized service?

A natural approach is to periodically audit the service provider by downloading the file. However, failure to download a file is not a proof of censorship. First, the provider could claim benign failure. Second, the proof is non-transferable: verifying censorship requires third parties to individually request the censored file. Moreover, a content provider may only selectively deny access to particular users or only for a short time frame. As such, checking by downloading does not work even for third parties who are online and willing to make queries.

In this paper, we introduce *proof of censorship*, whereby a content provider cannot delete or otherwise selectively remove content from their service without creating transferable cryptographic proof of their misdeed. Even if the provider restores the file at a later date, the proof remains valid, allowing the reputation of a content provider's commitment to censorship resistance to be based on the existence (or absence) of such proofs.

1 Introduction

Online censorship is done in many ways. In addition to blocking access to websites, censors also use legal means to remove information from content providers directly, such as through laws like the DMCA [5] in the United States. In the second half of 2016, Twitter received over 5,000 removal requests from government and police agencies worldwide, and complied with 19% of them [22]. In one example from August 2016, Twitter complied with a Brazilian government request to remove a tweet comparing then-mayoral candidate Rafael Greca to a fictional villain from the 1992 film Batman Returns [16].

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 99–115, 2018.
https://doi.org/10.1007/978-3-662-58387-6_6

This style of censorship can be significantly less apparent than overt website blocking. Because the service remains accessible, users are unaware of the content that may be secretly withheld from them. While some content providers (including Twitter) publish transparency reports containing overall statistics or examples of removed content [7,8,22], these reports are difficult to confirm or refute: How can users know if the transparency reports themselves have not been similarly censored?

While decentralized or distributed tools provide a way to make content robust against censorship, these techniques do not fully solve the problem. In a distributed censorship-resistant scheme, users must download and use specialized software which may itself be blocked by a censor. Due to this requirement, these tools are only utilized by well-informed users, while the majority of people rely exclusively on centralized content providers. Thus, there are clear advantages to providing censorship resistance in a centralized content storage context.

One idea proposed to incentivize censorship-resistant content providers to not delete data is to use *proof of retrievability* (PoR) to enable content providers to prove that they still retain a copy of a file. In PoR, providers respond to requests or challenges for specific subsets of a file they have agreed to store. A provider responds with the subset, proving (over many challenges) that they likely still store the entire contents of the file. With enough successful PoR challenge responses, a client can reconstruct the file, meaning that a provider that gives valid PoR responses for a file necessarily provides access to it. While this is a useful starting point, there are two major problems with this approach.

First, the content provider can always choose to not respond to a PoR request. Note that this is different from providing an invalid response to a request. By simply leaving a connection open and providing no response, the service never provides any incriminating evidence that they are not providing access to the file. While other clients can see this suspicious behavior, the provider could choose to respond correctly to some subset of users, seeding confusion and distrust over claims of censorship.

Second, the content provider can cease this behavior at any time, with no evidence that they ever censored a file. For example, a provider could wait until a certain number of users realized a file was removed or journalists started to ask questions before they reverted the decision to censor a file. Such a provider could pretend they never censored a file, and those that observed that they did would have no transferable proof. This allows a provider to censor files at will, and restore them only when convenient.

In this paper, we describe a way that a content provider can be held accountable for the files that they censor. We do this via a *proof of censorship* that a content provider unavoidably produces should they censor a file. Furthermore, these proofs are transferable, meaning that once one has been produced, others can verify the cryptographic proof and see that it attests to a censored file. Even if the content provider restores the original file, this proof still maintains the evidence of previous censorship.

To construct our proof of censorship, we use private information retrieval (PIR) which hides what information a client is requesting from a server. At a high level, servers commit to storing (encrypted) files by signing hashes of those files on upload. To download a file, a client performs a PIR query, which the server responds to, signing both the PIR query and the server's response. Because the server does not know what file a client is requesting, it cannot selectively answer queries. To censor a file, the provider must return garbage data. Upon decryption of the PIR response, the client can confirm the file has been censored, and can reveal its PIR queries and the signed response to the world. Anyone with this proof (and the content provider's long-term public key) can verify that the content provider has removed the file.

Our proofs of censorship are compact and relatively efficient to verify: in the event of censorship, for a reasonable size database, the proof is on the order of a few megabytes, and takes approximately 50 ms to verify on a laptop computer.

2 Background

To describe our proof of censorship, we first provide background on private information retrieval (PIR) [13]. PIR allows a client to request data from a server without revealing to the server what data is being requested. A naive way to accomplish this is for the client to download the entire database from the server, then select its data locally. This method is bandwidth inefficient, however, and does not scale to large datasets.

There are two settings for PIR. In *information-theoretic* PIR (ITPIR), a set of N servers share the database, with clients making requests to all of them and combining their responses to obtain the requested files. If at least one of the servers is not colluding with the others, the client's request remains private. In *computational* PIR (CPIR), a single server stores the database, and clients use

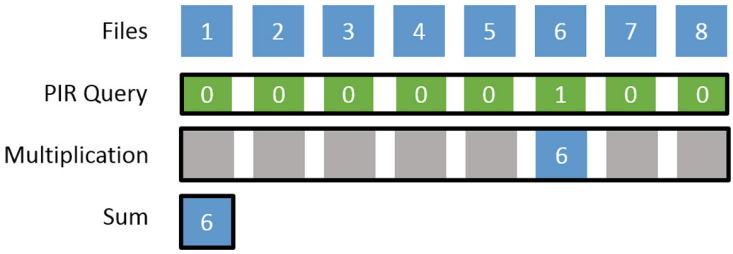


Fig. 1. Vector-matrix PIR—private information retrieval involves a client making a query against a database. In Vector-Matrix PIR the client encrypts a vector of query elements and the server then multiplies the query elements against the database blocks homomorphically, using homomorphic addition to produce a single sum. This results in an encrypted value corresponding to the element where the client encrypted 1 (instead of 0), which is sent back to the client for decryption.

homomorphic-encrypted requests to retrieve data. As long as the computational hardness assumptions on the underlying cryptography hold, the server cannot determine what information the client requested. As we are focused on a single centralized content provider, in this paper, we only consider and describe CPIR.

Many CPIR schemes fall into the "matrix-vector" model [10], which is illustrated in Fig. 1. In this model, CPIR uses a (IND-CPA-Secure, additive) homomorphic encryption that allows operations done on ciphertext to correspond to operations done on the corresponding plaintext. As an example, consider an additively homomorphic encryption function $E()$ with corresponding decryption function $D()$ that allows addition of ciphertext and multiplication by a constant; i.e,:

$$D(E(a) + E(b)) = a + b$$

and

$$D(E(a) \cdot c) = a \cdot c.$$

To perform a PIR query, a client constructs a vector of encrypted 0s and 1s. For example, if the server has n blocks, and the client wishes to retrieve block x from the server where $0 \leq x < n$, the client sends a query vector Q to the server, comprised of $q_{i \neq x} = E(0)$, and $q_x = E(1)$. Note that the IND-CPA security of the encryption scheme implies that the server cannot determine which q_i is the encryption of 1.

With Q, the server (homomorphically) multiplies each q_i with its corresponding data block d_i in the database. The server then takes the homomorphic sum over the result. When $i \neq x$, the multiplication by $E(0)$ will ensure the corresponding block does not contribute to the later-decrypted sum. The response is sent back to and decrypted by the client, resulting in $D(\sum_i q_i \cdot d_i) = D(E(1) \cdot d_x) = 1 \cdot d_x$.

The PIR library that we use (XPIR [2]) provides two optimizations: recursion and aggregation. Recursion allows clients to send fewer query elements to the server, by breaking the database into a d-dimensional cube, and having the query select the row/column vectors. For example with a database of 100 records, it can be broken into a 10×10 table of elements. The client sends 10 queries, which the server copies and applies to each row effectively selecting a singular column. Then, a separate 10 queries can be used to select a single element from that column. Thus the client sends a total of 20 query elements, as opposed to 100 (with no recursion).

Aggregation allows a client and server to pack multiple plaintext files into a single element. For example, if the ciphertext allows for an absorption of 768 bytes, but files are only 128 bytes, the database could utilize aggregation, fitting 6 files into each block. Clients then select the block of 6 files that contains their requested file, and locally discard the other 5. With aggregation, the client sends fewer request elements to the server, resulting in smaller queries.

3 Threat Model

We consider a setting consisting of a single centralized content provider with multiple clients who upload and download content. Clients upload files to the provider, and distribute tickets that enable others to download the file. The information in a ticket can be summarized in a URL to be provided to others. This allows clients to easily share tickets in the same way they would distribute online content to others. In this model, we wish to detect a censoring content provider while preventing malicious clients from making false accusations. We achieve two properties:

Non-repudiation of Targeted Censorship. A provider cannot selectively censor a file while responding to queries from honest clients without producing a transferable proof of their misdeed.

Unforgeability. Against a non-censoring content provider, an attacker cannot forge a proof that a file was censored.

We note that our threat model does not prohibit a provider that chooses to delete or remove access to all of its files: a provider can always shut down entirely, or refuse to sign statements with its private key.

4 System Design

In this section, we describe the details of our proof of censorship. A proof of censorship has two parts: a commitment from the server that it will store and distribute a file, and second, a later proof that it failed to uphold that commitment. The first part is obtained during file upload, where the server returns a signed and timestamped *ticket* that efficiently represents the commitment to store the file, while the second part is obtained when a client downloads the file the server is attempting to censor. We assume the server has a widely published long-term public signing key.

We begin our description assuming that a file fits in a single block that can be requested in a single PIR query to the server. In Sect. 4.2, we describe how to efficiently extend this idea to provide proofs of censorship for files that span multiple blocks.

4.1 Proof of Censorship Construction

Ticket Construction. On upload, a client will upload an encrypted block that represents the file. The server chooses an index for the block to be stored, and provides a signature over the data and the index. For block data B_i stored at index i, the server hashes the block data to obtain $H(B_i)$. The server then signs the tuple containing a timestamp t, the index i, and the block hash $H(B_i)$ using its long-term private key. This signature, along with t, i, and $H(B_i)$ are sent to the client in the form of a *ticket*.

This ticket can be encoded into a URL that can be distributed easily to other users in the form of:

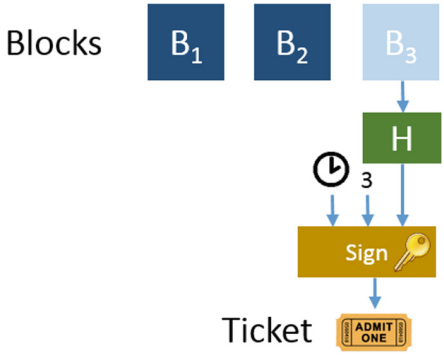


Fig. 2. Ticket creation—when a file is uploaded, the server produces a signed ticket that commits the server to storing the block of data. The server signs a timestamp, the index where the data is stored, and a hash of the data, and returns this to the client for local storage.

$$\texttt{https://<service>/\#}H(B_i)/i/t/sig$$

where *sig* is the server's signature of the ticket. This gives the client all of the information it needs to verify as well as a simple way to distribute the ticket. Additionally the sensitive information is provided to the client in such a manner that it is not passed to the server, as values after a # symbol are only handled locally in browsers.

The client then verifies that the ticket is well formed and valid. First, the client hashes the data it has uploaded to obtain $H(B)$. Then, it ensures that the timestamp t it received from the server is a recent timestamp[1]. Finally, the client checks the signature of the ticket, using t, i, and the client-computed $H(B)$. If the signature is valid, the client stores the ticket for this block. If any checks fail, the client can attempt to re-upload their file until they receive a well-formed ticket. Figure 2 illustrates how tickets are generated during file upload.

File Download. To download a file, a client creates a PIR query that will result in a proof of censorship if it does not receive the file requested. During a normal PIR request, a client encrypts a vector of 0 and 1 elements using random coins for the encryption. In our scheme, these coins are the output of a pseudorandom generator with a *seed* (Q_{seed}) randomly selected by the client. This allows us to later reproduce the same query with only the seed and the index i. With this we create a compressed representation of the query which consists of the queried index (in the clear), and the seed.

The client creates the request Q and sends it to the server, along with the public key (as well as any cryptographic parameters needed to allow the server to perform homomorphic operations on the query[2]). The server then performs the

[1] The client must ensure t is not located days in the future to avoid a server producing invalid proofs later.

[2] E.g. in Paillier, this involves the public modulus generated by the client.

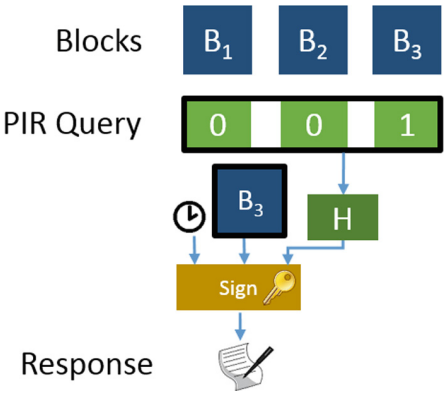


Fig. 3. Proof of censorship—to verify a file is still in place, a client constructs a PIR query for its block. The server, without knowing which block was requested, signs the encrypted PIR query and the response it produces. The client decrypts the response, and can verify the data is correct. If it is not, the client can combine the signed response, the parameters it used to generate the PIR query, and the original ticket for this file to produce a stand-alone proof-of-censorship that can be verified by a third party.

query over its database using PIR, producing a short reply R that contains an encrypted representation of B_i. The server then hashes Q (including the public key) to obtain $H(Q)$, and produces a signature over a timestamp t, the query hash $H(Q)$, and the reply R. The server sends this signature, along with t and R to the client (Fig. 3).

The client then extracts B_i from R using standard PIR extraction (i.e. decrypting R with its Q_{seed}-derived private key). The client then checks if this is the expected file by comparing it to the hash in the corresponding ticket for this file. If the client detects that the block has been censored, it now has a proof of censorship in the form of the server's response. The client publishes the original ticket, the server's signed PIR reply, and its Q_{seed} as a proof of censorship.

During this process, the server does not know what file is being requested. Therefore, to censor a file, the server must either not respond to any queries, or risk revealing that it has removed a file from its database.

Verifying Proofs of Censorship. Given the server's public key, a signed ticket, a signed reply, and a query seed, a third party can verify whether the proof is valid evidence of censorship. To verify a proof of censorship, a verifier must perform several steps, detailed below.

1. **Check Timestamps.** The verifier checks that the ticket's timestamp (t_t) is before the reply's timestamp (t_r), ensuring that the query took place after the server committed to storing the file.
2. **Check Ticket Signature.** Given the ticket's timestamp t_t, the requested index i, and the hash of the file $H(B_i)$, the verifier validates the ticket's signature with the server's public key.

3. **Regenerate PIR Query.** Verifier uses Q_{seed} and the index i to deterministically derive the PIR query Q, and computes the hash over the query $H(Q)$.
4. **Check Reply Signature.** Given the reply's timestamp t_r, the computed hash $H(Q)$, and the server's reply, the verifier checks the reply's signature using the server's public key.
5. **Extract Reply.** Again using the key derived from Q_{seed}, the client extracts the PIR reply R to obtain B_i.
6. **Check Data Hash.** The verifier finally compares the hash of the extracted data B_i to the hash committed to in the ticket. If the hashes are equal, the server did *not* censor the file, and returned the expected result. However, if the hashes do not match, this is a valid proof of censorship.

4.2 Handling Multiple Blocks

So far, we have described how a client can efficiently receive a proof of censorship for a single uploaded block, but this does not address what happens if a file is larger than a single block. While a client could easily split up a file into multiple blocks, and receive a ticket for each block, storing each of these signed tickets may be expensive.

Instead, we can extend our design to allow for multiple blocks to receive a single ticket, while still allowing a proof to catch if an individual block is censored. To do this, our ticket will contain a Merkle tree root [18] in place of the single block's hash. The leaves of the Merkle tree will be the hashes of each block's data, and the ticket will consist of the list of hashes, and the final signature over the Merkle root, timestamp, and block index range.

To verify a proof, the verifier reconstructs the Merkle root from the suspected censored block and its Merkle tree-adjacent neighbors, and uses the root to verify the ticket. Thus, a verifier does not have to know all of the hashes of the tree, but rather only those that allow it to reconstruct the Merkle root ($log(N)$ hashes for an N-block file). This allows proofs to stay relatively compact even as the file size grows. After validation of the ticket, the verifier can be assured that the hash of the suspected block ($H(B_i)$) they have been given is the correct hash for the given block B_i, and can perform the remaining checks described previously.

4.3 Security Argument

Non-repudiation. Assuming an honest client, that H is a collision resistant hash function, and the query privacy of the PIR scheme, we can show that the non-repudiation property holds (that a selectively censoring server will produce a proof of censorship).

The server can attempt to censor a file in two ways: by not responding to queries for the file, or by creating a malformed response. Malformed responses themselves can be made by either changing the data as it is stored/responded to, or by creating invalid signatures over the reply depending on the query.

A server that chooses to change its response based on the query (either not respond or produce an invalid signature) is prevented by the privacy of the PIR

scheme. The server cannot know what file is being requested, and thus cannot selectively perform different behaviors based on the file requested. Honest clients will detect and reject malformed responses, effectively censoring the file they requested without the server knowing which file it is censoring.

The server can modify a file block directly, but because they have committed to the hash of the block's data in the ticket, and because of the collision resistance of the hash, the server is unable to change this data without producing a proof when that data is next requested.

Unforgeability. We show that the scheme is unforgeable assuming the servers signature scheme is unforgeable, H a random oracle, and the PIR scheme is the "vector-matrix type" [10] with an underling encryption scheme that is "homomorphic quasi-committing".

Homomorphic Quasi-Committing. A homomorphic encryption scheme gen, enc, dec is homomorphic quasi-committing if an attacker cannot produce a ciphertext ct and values $m_1, m_2, c_1, c_2, pk, sk, r_{keygen}, r_1, r_2$, such that

$$pk, sk \leftarrow gen(\lambda; r_{keygen})$$

$$ct = c_1 \cdot enc(pk, m_1; r_1) + c_2 \cdot enc(pk, m_2; r_2)$$

where

$$dec(sk, ct) \neq c_1 \cdot m_1 + c_2 \cdot m_2$$

This property is closely related to correctness, which states that for all choices of random coins and query indexes, an incorrect result is returned with negligible probability. However, crucially, correctness is insufficient here. First, it does not ensure that a given query could be correct for two different choices of coins (and therefore private keys) and query indexes. I.e, it doesn't explicitly prevent a forgery where an attacker opens a legitimate query to a different result. Second, correctness effectively says there is a negligibly set of bad random coins which cause the scheme to fail. It provides no protections against an attacker choosing from that subset intentionally.

Homomorphic quasi-committing is a strong property to require of an encryption scheme. For the rest of the paper, we assume that the underlying encryption scheme used in the PIR is homomorphic quasi-committing in the random oracle model where r_{keygen}, r_1, r_2 are the output of a random oracle h evaluated on a nonce $nonce$ and the query index i.

Consider an attacker interacting with an honest content provider who produces a ticket and a proof of forgery for that ticket. Either the ticket or the PIR transcript must be forged. Forgeries in the ticket are prevented by the collision resistance of h and the security of the signature scheme.

Security Argument. We now consider the case of forgeries in the PIR transcript. By the collision resistance of the hash function and security of the signature scheme, a attacker cannot substitute either the query or response. As such the

PIR transcript itself must represent the actual query to and response from the provider. The only freedom an attacker has here is the choice of openings for the transcript (i.e. the random coins used for encryption and the public and private keys.) and she must reveal those as part of the proof. Thus the attacker reveals

$$pk, sk, r_{keygen}, r_1, \ldots, r_n, ct$$

for a query on block i such that

$$ct = B_i \cdot enc(pk, 1; r_i) + \sum_{j=0, j \neq i}^{N} B_j \cdot enc(pk, 0; r_j)$$

and

$$dec(sk, ct) \neq B_i$$

It follows that $0, 1, \sum B_j, B_i, pk, sk, r_{keygen}, \sum r_j, r_i$ violates the "homomorphic quasi-committing" property of the encryption scheme for $B_* = \sum_{j=0, j \neq i}^{N} B_j \cdot enc(pk, 0; r_j)$ By assumption, this is not possible.

5 Implementation and Evaluation

We implemented our proof of censorship construction on top of XPIR, a fast PIR tool written in C++ [2]. Our implementation consisted of several hundred lines of modifications to XPIR, as well as an approximately 500-line application using the resulting libXPIR library. We used OpenSSL to perform the necessary signatures and hashes in our protocol.

The parameters we selected for testing were motivated by a simple text-only Twitter-like application, where messages are short (256 bytes, enough to include a short post and its metadata). We tested against a database of 1 million simulated messages in order to evaluate the time and size of different parts of the system. We used a quad-core Intel Core i5 CPU (3.30 GHz) with 32 GBytes of RAM to evaluate our prototype.

Table 1. Ticket and proof size—we implemented a prototype of our proof-of-censorship system, simulating a database of 1 million 256-byte messages. We are able to keep a constant size of our proof at 2 MB. This allows clients that receive a proof of censorship to be able to easily distribute it.

	Size	Generation time (std-dev)	Validation time (std-dev)
Ticket	120 bytes	334 µs (0.53)	381 µs (4.61)
Query	3.8 Mbytes	28 ms (0.44)	n/a
Reply/proof	2.0 Mbytes	2.8 s (0.19)	52 ms (0.54)

Table 1 shows the size and time to generate and validate our ticket, PIR query, and PIR reply (containing the proof). For our XPIR parameters, we

choose to use XPIR's Learning with Errors implementation with 248 bits of security, using a polynomial of degree 2048 and a modulus of 60 bits, with recursion level (d) 3, and aggregation (α) 16. All of the client operations are relatively fast: ticket validation (395 µs), query generation (27 ms), plaintext extraction (3.5 ms), and proof validation (45 ms) suggest that clients could be implemented in smartphones or even Javascript web pages without significant performance issues [1].

5.1 Scalability

We performed measurements of our proof of censorship and underlying PIR library to determine how the system scales. We measured the performance of downloading a single file with various database sizes, ranging from 5,000 to 1 million files of both small (256 byte) and large (1 MByte) size. For each database size, we used XPIR's built-in optimizer to determine the best parameters. XPIR chooses between two encryption options (Paillier, and the lattice-based learning with errors (LWE)), varying parameters of each to be optimal given a bandwidth estimate, file size, and number of files. We encouraged XPIR to optimize for small query and response sizes by providing a low bandwidth. Although Paillier produced the smallest query/responses, its server-side processing time was many orders of magnitude slower than LWE, effectively making it impractical. As a compromise between bandwidth and processing time, we selected LWE encryption optimized for small query and response sizes. For small files (256 bytes), the XPIR optimizer selected a polynomial of degree 2048 and 60-bit modulus with varying recursion and aggregation values depending on database size. When considering 1 MByte files the optimizer selected differing LWE flavors, all above the 80-bit security level, with polynomial degrees ranging from 2048 bits to 4096 bits and modulus bits ranging from 60 bits to 180 bits.

Table 2. Small file scaling—we measured query and response sizes generated for several different database sizes with 256 byte files to determine how the system scales for a Twitter-like content server. We find that even at millions of files, the system remains relatively practical: an 8 MByte query and 2 MByte response are needed to select a single file privately (with accompanying proof of censorship) from 10 million files.

Number of files	5k	10k	50k	100k	500k	1M	10M	100M
Recursion depth	1	1	2	2	2	3	3	3
Aggregation value	300	420	96	64	126	16	16	15
Query size (MBytes)	0.53	0.75	1.44	2.5	3.9	3.8	8.0	17.7
Reply size (MBytes)	0.56	0.78	1.44	1.0	2.0	2.0	2.0	2.4

For each database size we measured the size of queries a client would have to generate and the time to generate them. On the server side we measured the amount of replies it needed to send (based on file size and aggregation value)

and the time it took to construct those queries. And finally, back on the client side, we measured the time it took to extract the actual response from the given replies. The results of our experiments are shown in Tables 2, 3 and Fig. 4.

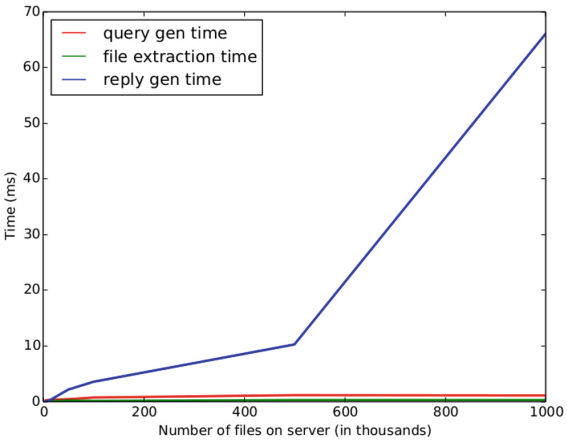


Fig. 4. Amount of time taken for the actions necessary for our tool to work. The time taken for the client to generate queries, and to extract the response from the server replies is very minimal compared to the amount of time that is spent by the server generating the replies to send back to the server. This time however can be decreased by using servers with hardware specifically designed to do these computations.

For comparison, we allowed the optimizer to select Paillier 1024, which generates a 34.7 KB query and a 9.6 KB response when querying a 256 byte file from 1 million files. While this is several orders of magnitude smaller than LWE, server reply generation took nearly 2 h for a single query. However, query generation and extraction times on the client were still fast (100 ms and 77 ms respectively). This suggests that with specialized modular arithmetic hardware on the centralized server, reply times could be considerably improved, potentially making this scheme practical and very attractive for its small query and response sizes.

We also note that a server need not contain all of its files in a single PIR database. Instead, one could partition the file space into many buckets, and requests could be made over a much smaller number of files in each bucket, similar to an idea proposed in bbPIR [25]. This allows for a tradeoff between the granularity at which a server can censor without producing a proof, and the efficiency or scalability of the system as a whole. With more buckets, a server could choose to not respond to queries for an entire bucket without producing a proof of censorship. If each bucket only contained a few files, the collateral damage to censoring the entire bucket could be small. In addition, if multiple buckets exist, a server that wishes to censor could place new files in their own bucket, allowing for free censorship of the file in the future should it desire. To combat this, clients could choose which buckets they insert files into.

Table 3. Large file scaling—we also measured query and response sizes assuming 1 MByte files, to approximate our system being applied to an image or video-streaming content server. The XPIR optimizer chooses different security, polynomial and modulus parameters for each database size for 1 MByte files, due to it attempting to limit the amount of data the client has to send and receive over the network. While the parameters that are chosen are not always the most secure or privacy preserving, these parameters do result in the least amount of bandwidth necessary for clients to use. Additionally, while overheads are low, even with only 50 thousand files in a database, responses are 73 MBytes for a single megabyte file, likely making application of proof of censorship impractical to video streaming content providers.

Number of files	5k	10k	50k	100k	500k	1M	10M	100M
Recursion depth	2	2	2	2	2	2	3	3
Aggregation value	1	1	1	1	1	1	1	1
Query size (MBytes)	17.8	25.0	14.0	19.8	44.2	62.5	121.3	174.1
Reply size (MBytes)	32.8	32.8	73.1	73.1	83.7	83.7	138.4	199.4

6 Discussion

In this section, we discuss possible attacks and defenses on our proof of censorship system, as well as describe potential incentives and future applications for this type of proof.

6.1 Attacks

Server Produces Invalid Tickets. The server can choose to either not sign or not produce tickets during file upload, allowing it to delete those files later. However, the server does not know what file is being uploaded at the time of upload: the file could be encrypted, where the information to download the file (e.g. URL) contains the key used to decrypt it. Thus, the server must choose to produce tickets or not without knowing file contents, making it difficult to target specific content. In addition, clients that do not receive a valid ticket could reupload the file (perhaps through an anonymous proxy [6]) until they receive one.

Server Doesn't Sign Replies. By using PIR, the server does not know what file is being downloaded. Therefore, it cannot know if a particular request is for the file it wishes to censor. It can choose to never sign replies (or sign them randomly), but it does so without knowledge of the file involved. In this case, we can require that honest clients refuse to extract downloaded files unless the PIR reply contains a valid signature, meaning that the server effectively would be censoring unknown files that were requested. This is effectively the same as a server shutting down its service by not responding to any queries.

Incorrect Timestamps. A server can advance timestamps in tickets (or delay them in replies), tricking verifying clients into thinking a proof of censorship

is invalid because the reply appears to come after the ticket (a feature used to protect against client forgeability). To solve this, we require clients to check the timestamps of tickets and replies and ensure they are recent before considering the response valid. This may still leave room for an equivocating server to leave a small window to censor a file, but if uploaded files are encrypted, the server will have little information to determine if the file should be censored before the window has passed.

6.2 Incentives and Applications

How do we use a proof of censorship? The first answer is as a reputation sentinel: Proofs of censorship can be used to show others that a censoring content provider cannot be trusted. However, this only works if the proof is widely distributed (and not itself censored). As the proof is on the order of a few megabytes, it should be transferable to other verifiers via social media, email, or other content-sharing sites.

An intriguing possibility is to use proofs of censorship to impose a financial penalty. This could take the form of a smart contract in Ethereum [26] that controls a bond posted by the provider which is payable (in part) to whoever submits a valid proof of censorship.

Another option is to force the provider to reveal some sensitive secret if they ever censor a file. This can be accomplished via either multi party computation or trusted hardware. In either case, the content provider is forced to blindly interact with someone and evaluate a function that either returns nothing or, if fed a valid proof of work, returns the secret. For example, every request for a file could be accompanied by a query to the provider's SGX enclave [11] via an encrypted channel that is terminated within the enclave. The enclave could derive a private key from the server's secret, and use it to sign responses (requiring the enclave to have the secret loaded). If the enclave receives a valid proof of censorship, it returns the secret encrypted under the requester's key. Otherwise, it returns a dummy value. If the server chooses to provide no response at all, the honest client aborts[3]. This forces a censoring provider to either take down their whole system service once they have censored a file, or risk leaking their secret.

Proofs of censorship may also be purposefully created by a provider to disclose the extent of what they have censored. In an effort toward transparency, many content providers voluntarily report takedown notices and removal requests to Lumen [15] (formerly Chilling Effects). However, there is no guarantee that a provider hasn't withheld reports from this database. To combat this, providers could submit their own proofs of censorship to the database, and any proofs of censorship not present in the database serve as evidence of the provider attempting to secretly censor without reporting.

[3] To prevent lazy but well meaning clients simply ignoring empty enclave responses, the enclave could instead return a decryption key needed for the particular file.

7 Related Work

Previous research has explored alternative solutions to the problem of content censorship.

One such approach is proof of retrievability, proposed by Juels et al. in 2007 [12]. In this model, servers provide cryptographic proof to users that they have access to a file. However, as previously mentioned, this does not mean that a server must provide such a proof for every file requested: if the server knows what portion of a file is being requested, they can censor specific parts or files by simply not responding.

Several works have provided monetary incentives for successful proofs of retrievability. Permacoin proposes a cryptocurrency with proof of retrievability of an agreed-upon file in place of the traditional proof of work [19]. This encourages miners to keep portions of the file around in order to qualify for mining rewards associated with the currency. Lavinia incentivizes publishers to store documents by having a third-party verifier check and provide payments to well-behaved publishers in a distributed system [3].

Numerous projects have detailed the idea of combining files to discourage their removal from servers. Tangler [23] stores files across multiple servers and "entangles" newly uploaded files with existing files using Shamir secret sharing [20]. This entanglement means that deleting an old file may inadvertently remove more recently uploaded files that depend on it, increasing the collateral damage in censoring a file. Dagster [21] `xor`s blocks of data together requiring users to obtain multiple blocks from the server in order to retrieve one block of desired data. This combining of blocks ties files together in such a way that if older blocks are removed, newer blocks are no longer accessible. However, newly uploaded files are not protected, and access patterns of files could be used to detect what file is being downloaded.

Others have leveraged distributed systems to make content harder to censor. Publius [24] allows clients to upload encrypted files to multiple servers along with parts of a split encrypted key in such a way that a threshold of servers behaving honestly will allow the file to be retrievable. Freenet [4] provides a volunteer-based peer-to-peer network for file retrieval with the aim of allowing both authors and readers to remain anonymous. Tor [6] supports hidden services, where a central provider could potentially obscure its network location while hosting objectionable content. However, all of these schemes lack a mechanism to discourage servers or participants from misbehaving, opting instead to either hide or share responsibility of hosted content. Moreover, they provide no guarantees on file lifetimes, which is determined by the resource constraints of the participating servers.

8 Future Work

Proof of censorship could be extended in several directions and applications. As mentioned previously, monetary incentives built on top of such proofs could encourage content providers to deploy such a system and keep them honest.

Beyond this, there are many open problems in how to apply proof of censorship to different applications in an efficient manner. For instance, applying this scheme to a video streaming service would be a difficult engineering task, as large file sizes, database volumes, and high bandwidth demands require low-overhead efficient solutions. To solve this problem, it may be possible to combine proof of censorship with other scalable private media retrieval systems such as Popcorn [9].

Proof of censorship could also augment existing key transparency applications, such as Certificate Transparency [14] or CONIKS [17]. Although these systems already detect server equivocation when a server modifies a particular object, they fail to provide any sort of guarantee on responses for every object in their certificate or key store. Using proof of censorship, these systems could provide such an assurance in addition to the protections provided.

9 Conclusion

Content censorship from providers remains a growing problem. As network effects push users and content toward more centralized provider platforms, legal and political pressures have followed suit. While centralized providers can claim they stand for free speech or open access, they have no mechanism to prove they do.

In this paper, we have presented a scheme whereby a content provider can stand behind such a claim cryptographically. By deploying this scheme, providers will create a cryptographically verifiable and transferable proof of censorship should they delete or remove access to a specific file. The threat of this proof provides a disincentive to content providers from even temporarily censoring a file, as their reputation with respect to Internet freedom is at stake.

Acknowledgements. We would like to thank the anonymous reviewers for their feedback on our work, as well as our shepherd Ryan Henry, who provided useful direction and thoughtful discussion on this paper.

References

1. Web cryptography API. https://www.w3.org/TR/2016/PR-WebCryptoAPI-20161215/
2. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.-O.: XPIR: private information retrieval for everyone. Proc. Priv. Enhanc. Technol. **2**, 155–174 (2016)
3. Bocovich, C., Doucette, J.A., Goldberg, I.: Lavinia: an audit-payment protocol for censorship-resistant storage. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 601–620. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_34
4. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_4
5. Congress, U.: Digital millennium copyright act. Public Law **105**(304), 112 (1998)

6. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. Technical report, DTIC Document (2004)
7. Facebook: Government requests report (2017). https://govtrequests.facebook.com/
8. Google: Transparency report (2017). https://transparencyreport.google.com/
9. Gupta, T., Crooks, N., Mulhern, W., Setty, S.T., Alvisi, L., Walfish, M.: Scalable and private media consumption with popcorn. In: NSDI, pp. 91–107 (2016)
10. Hafiz, S.M., Henry, R.: Querying for queries: indexes of queries for efficient and expressive IT-PIR. Cryptology ePrint Archive, Report 2017/825 (2017). https://eprint.iacr.org/2017/825
11. Intel: Intel Software Guard Extensions. https://software.intel.com/en-us/sgx
12. Juels, A., Kaliski Jr, B.S.: PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 584–597. ACM (2007)
13. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, pp. 364–373. IEEE (1997)
14. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. Technical report (2013)
15. Lumen: Lumen database. https://lumendatabase.org/
16. Lumen: Brazil - court order to twitter (2016). https://www.lumendatabase.org/notices/12866354
17. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: bringing key transparency to end users. In: Usenix Security, pp. 383–398 (2015)
18. Merkle, R.C.: Method of providing digital signatures. US Patent 4,309,569, 5 January 1982
19. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: repurposing Bitcoin work for data preservation. In: 2014 IEEE Symposium on Security and Privacy (SP), pp. 475–490. IEEE (2014)
20. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
21. Stubblefield,A., Wallach, D.S.: Dagster: censorship-resistant publishing without replication. Rice University, Technical Report TR0l-380 (2001)
22. Twitter: Removal requests (2017). https://transparency.twitter.com/en/removal-requests.html
23. Waldman, M., Mazieres, D.: Tangler: a censorship-resistant publishing system based on document entanglements. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 126–135. ACM (2001)
24. Waldman, M., Rubin, A.D., Cranor, L.F.: Publius: a robust, tamper-evident censorship-resistant web publishing system. In: 9th USENIX Security Symposium, pp. 59–72 (2000)
25. Wang, S., et al.: Generalizing PIR for practical private retrieval of public data. DBSec **10**, 1–16 (2010)
26. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)

Economic and Usability Analysis

An Economic Study of the Effect of Android Platform Fragmentation on Security Updates

Sadegh Farhang[1(✉)], Aron Laszka[2], and Jens Grossklags[3]

[1] Pennsylvania State University, State College, USA
farhang@ist.psu.edu
[2] University of Houston, Houston, USA
alaszka@uh.edu
[3] Technical University of Munich, Munich, Germany
jens.grossklags@in.tum.de

Abstract. Vendors in the Android ecosystem typically customize their devices by modifying Android Open Source Project (AOSP) code, adding in-house developed proprietary software, and pre-installing third-party applications. However, research has documented how various security problems are associated with this customization process.

We develop a model of the Android ecosystem utilizing the concepts of game theory and product differentiation to capture the competition involving two vendors customizing the AOSP platform. We show how the vendors are incentivized to differentiate their products from AOSP and from each other, and how prices are shaped through this differentiation process. We also consider two types of consumers: security-conscious consumers who understand and care about security, and naïve consumers who lack the ability to correctly evaluate security properties of vendor-supplied Android products or simply ignore security. It is evident that vendors shirk on security investments in the latter case.

Regulators such as the U.S. Federal Trade Commission have sanctioned Android vendors for underinvestment in security, but the exact effects of these sanctions are difficult to disentangle with empirical data. Here, we model the impact of a regulator-imposed fine that incentivizes vendors to match a minimum security level. Interestingly, we show how product prices will decrease for the same cost of customization in the presence of a fine, or a higher level of regulator-imposed minimum security.

1 Introduction

Android, the mobile operating system released under open-source licenses as the Android Open Source Project (AOSP), has the largest market share among smartphone platforms worldwide with more than one billion active devices [2]. Due to the openness of the platform, vendors and carriers can freely customize

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 119–137, 2018.
https://doi.org/10.1007/978-3-662-58387-6_7

features to differentiate their products from their competitors. This differentiation includes customizing the hardware, but there is also a substantial fragmentation in the software packages utilized in the Android ecosystem [15,22].

The fragmentation of the software base available from various vendors is due to various customization steps, including the modification of the open source Android codebase as well as the addition of proprietary software. Product differentiation may benefit consumers by providing Android devices for sale that better match consumer tastes, and may also benefit businesses by helping them to sidestep intense price competition of homogeneous product markets [24].

However, we also observe that Android platform fragmentation is associated with a number of security challenges [23,25,26]. For example, Wu et al. showed that a large proportion of security vulnerabilities in the Android ecosystem are due to vendor customization. They calculated that this proportion is between 64% to 85% for different vendors [25]. Similarly, Zhou et al. showed how customized drivers for security-sensitive operations on Android devices available by different vendors often compare unfavorably to their respective counterparts on the official Android platform [26]. Thomas et al. provided evidence for the substantial variability of security patch practices across different vendors and carriers [23]. Using a dataset about over 20,000 Android devices, they showed that on average over 87% of the devices are exposed to at least one of 11 known critical (and previously patched) vulnerabilities.[1]

The Android ecosystem fragmentation and the associated security problems have caused consumer protection agencies to intervene in the marketplace. In 2013, the Federal Trade Commission (FTC) charged a leading vendor because it "failed to employ reasonable and appropriate security practices in the design and customization of the software on its mobile devices" [9]. The case was settled and the vendor was required to "establish a comprehensive security program designed to address security risks during the development of new devices and to undergo independent security assessments every other year for the next 20 years" [9]. Not observing significant improvements in the Android ecosystem, the FTC recently solicited major vendors to provide detailed information about their security practices including what vulnerabilities have affected their devices as well as whether and when the company patched those vulnerabilities [8].

In this paper, we propose a product differentiation model that captures key facets of the Android ecosystem with a focus on the quality of security. We consider multiple competing vendors, who can customize Android for their products in order to differentiate themselves from their competitors. We consider both security-conscious consumers, who value security quality, and naïve consumers, who do not take security issues into consideration when they make adoption choices. When consumers are naïve, vendors do not have any incentives to address security issues arising from the customization. In order to incentivize investing in security, a regulator may impose a fine on vendors that do not

[1] Further compounding the problem scenario is how third-party apps targeting outdated Android versions and thereby disabling important security changes to the Android platform cause additional fragmentation [19].

uphold a desired level of security. We show that fines can achieve the desired effect, and we study how they impact the competitive landscape in the Android ecosystem.

Roadmap. In Sect. 2, we provide background on Android customization and the associated security challenges. Section 3 presents the economic model on Android customization. We analyze the model without a fine in Sect. 4 and consider how to calculate the parameters in our model in Sect. 5. We extend the model to the case with a regulator-imposed fine in Sect. 6. We support our analysis with numerical results in Sect. 7. We conclude in Sect. 8.

2 Background

Customization: One approach to measure the level of customization by vendors is *provenance* analysis [25], which studies the distribution and origin of apps on Android devices. There are mainly three sources of app origins on Android devices: (1) AOSP: apps available in the default AOSP that, however, can be customized by a vendor; (2) Vendor: apps that were developed by that vendor; and (3) Third-party: apps that are not in AOSP and were not developed by the vendor.

Table 1 summarizes the published findings of a provenance analysis of five popular vendors: Google, Samsung, HTC, Sony, and LG [25]. The authors found that on average 18.22%, 64.41%, and 17.38% of apps originate from AOSP, vendors, and third parties, respectively. Further, the number of apps and lines of code (LoC) associated with the devices are increasing with newly released versions. Likewise, the complexity of the baseline AOSP is increasing over time [25].

Table 1. Provenance analysis [25].

Vendor	Device	Version and Build#	#apps	#LOC	AOSP		Vendor		3rd-party	
					#apps	#LOC	#apps	#LOC	#apps	#LOC
Samsung	Galaxy S2	2.3.4; 19100XWKI4	172	10M	26	2.4M	114	3.5M	32	4.1M
Samsung	Galaxy S3	4.0.4; 19300UBALF5	185	17M	30	6.3M	119	5.6M	36	5.3M
HTC	Wildfire S	2.3.5; CL362953	147	9.6M	24	2.7M	94	3.5M	29	3.3M
HTC	One X	4.0.4; CL100532	280	19M	29	4.7M	190	7.3M	61	7.5M
LG	Optimus P350	2.2; FRG83	100	6.1M	27	1.1M	40	0.6M	33	4.4M
LG	Optimus P880	4.0.3; IML74K	115	12M	28	3.1M	63	3.2M	24	5.6M
Sony	Xperia Arc S	2.3.4; 4.0.2.A.0.62	176	7.6M	28	1.1M	123	2.6M	25	3.8M
Sony	Xperia SL	4.0.4; 6.1.A.2.45	209	10M	28	1.8M	156	4.1M	25	4.7M
Google	Nexus S	2.3.6; GRK39F	73	5.2M	31	1M	41	2.8M	1	1.3M
Google	Nexus 4	4.2; JOP40C	91	15M	31	2.5M	57	12M	3	1.1M

One of the challenges in this fragmented ecosystem is the security risk that arises from the vendors' and carriers' customization to enrich their systems' functionality without fully understanding the security implications of their customizations. In this paper, our focus is on security issues resulting from such

customization. We provide an overview of relevant work in this area in the following subsection.

Security Impact of Customization: The problems related to security aspects of Android customization are mainly due to vendors' change of critical configurations. These changes include altering security configurations of Linux device drivers and system apps, etc. One approach for better understanding the effect of customization is to compare security features of different Android devices with each other, which is called differential analysis. Aafer et al. proposed a number of security features to take into account [1]. First, *permissions* which protect data, functionalities, and inner components can be analyzed. In Android, we have four level of permissions: *Normal*, *Dangerous*, *Signature*, *SystemOrSignature*. The goal of differential analysis is to find a permission with a different (and typically lower) level of protection on some devices. Second, group IDs (*GIDs*) are another feature to take into account. Some lower-level GIDs are given Android permissions, which could potentially be mapped into a privileged permission due to customization. Protected broadcasts sent by system level processes are a third important security feature. Due to customization, some protected broadcasts could be removed and, as a result, apps can be triggered by not only system-level processes but also by untrusted third-party apps.

By comparing these security features, Aafer et al. found that the smaller the vendor is, the more significant inconsistencies are observable for the different security features. One interpretation is that the cost of investment in security is too high for those vendors (e.g., hiring of security experts). The results also imply that different vendors invest in security to different degrees.

Research aiming to understand Android customization is clearly demonstrating that customization is a pervasive feature in Android, and this is associated with a wide variety of security challenges and vulnerabilities. Further, we are unaware of any research that provides evidence for security improvements resulting from customization, which outweighs the aforementioned risks. At the same time, research is missing that aims to understand the economic forces associated with the customization process, which is the objective of our work.

Product Differentiation: Hotelling proposed a widely cited model for product differentiation in which a linear city of fixed length lies in the horizontal axis, and consumers are distributed uniformly in this interval [16]. Firms strategically chose a location in this space, since consumers appreciate firms who are closer to their location. We draw from this basic setup, and a more tractable extension using a quadratic function for consumer preferences for distance [5,24]. An alternative product differentiation model was proposed by Salop with consumers located uniformly on a circular city [21]. These two models are typically referred to as spatial competition or horizontal differentiation. In contrast, vertical (quality) differentiation has been used to formalize quality competition [11]. Our model also draws on quality differentiation by considering different levels of security investments by Android vendors.

Another type of (*perceived*) product differentiation is related to the lack of complete information about the characteristics of different products by

consumers, which is called *information differentiation*. Advertising is a key factor affecting the perceived product differentiation and resulting consumer demand [24]. Kaldor [17] proposed that advertising yields information benefits to consumers, while other researchers suggest that advertisement misleads consumers [12,24]. In our model, we take the view that consumers are largely uninformed about the security quality of Android devices by different vendors when they make adoption decisions. We are unaware of any research studying explicitly users' awareness of Android OS security, however the recent FTC request for information from Android vendors provides indirect evidence of the opaqueness of Android security practices [8]. In addition, a multitude of papers have addressed lack of awareness about third-party app security (e.g., [20]). While we believe that at least a small segment of consumers is concerned about Android security, and takes proactive steps to inform themselves (see, for example, the following paper on Android permissions [10]), we defer research on populations with mixed levels of security-awareness to future work.

3 Model Definition

In this section, we propose our baseline model in the tradition of *game theory* and the *theory of product differentiation* [16,24]. Our model considers three types of entities: (1) AOSP, (2) vendors, such as Samsung or LG, and (3) consumers.

AOSP: Google, the developer of Android, provides monthly security updates for its devices and for base Android. However, other vendors have to adjust AOSP security updates for their Android devices because of their customization. Further, customization may also introduce new security vulnerabilities.

To incorporate these effects into our model, we assume that a customized version of Android can be represented by a point on the segment $[0, 1]$. Our analysis could be extended to multidimensional customizations in a straightforward way, but we assume one dimension for ease of presentation, since our focus is on the relative level of customization rather than its direction. Moreover, the location of each Android customization is independent of objective measures of product quality. In other words, we map the features of a mobile device to a point on the segment $[0, 1]$ to quantify its difference (e.g., percentage of customized code) from the base version of Android provided by AOSP. In our model, Z_A denotes the point corresponding to the base version of AOSP. Since AOSP aims to provide a base version that maximizes the market share of Android, it provides a version that can attract the widest range of consumers. Hence, in the numerical analysis, we assume that AOSP is in the middle, i.e., $Z_A = 0.5$.

Vendor: There are multiple vendors selling Android devices. Likewise, carriers can also sell the vendors' Android devices with their own prices and customizations. Here, we will use the term "vendor" to refer to both vendors and carriers. The price and the market share of the device sold by vendor i are denoted by p_i and D_i, respectively. Further, q_i denotes the security quality of patches delivered by vendor i. We assume that $p_i \geq 0$ (product prices are non-negative) and $q_i \geq 0$

(security quality is represented by a non-negative number) for every vendor i. Similar to the AOSP base version, a point $z_i \in [0, 1]$ represents the customization of the Android version of vendor i.

We consider two types of costs for customization. First, through customization, the vendor makes its product different from what Google has developed in AOSP. Hence, the vendor incurs development cost, which is related to the degree of customization. Here, we model this cost as a convex quadratic function of the difference between the vendor's position and the positions of the AOSP base version. Second, the security related cost of a vendor depends not only on the quality and frequency of security updates provided by the vendor, but also on the difference due to customization. Vendors receive security patch updates from AOSP, but due to customizations, vendors need to adapt these security patches before distribution. Often, vendors degrade the quality or frequency of security patches in order to save development and distribution costs [23]. Hence, the security-related cost is affected by both the customization level and the security quality. In our model, we employ a convex quadratic function to capture how the security cost of vendor i depends on q_i. The utility of vendor i is equal to:

$$\pi_i = p_i D_i - C_i \left(z_i - Z_A\right)^2 - S_i q_i^2 \left(z_i - Z_A\right)^2, \tag{1}$$

where C_i and S_i are constants representing cost per unit of customization and security quality, respectively. Note that we focus on security issues resulting from Android customization rather than security-related cost of AOSP.

We have considered quadratic functions for the cost terms, which is a common assumption for modeling customization costs, e.g., see [4] and [6]. The quadratic cost function captures the fact that the cost of customization increases as the customization increases. In a similar way, with an increase in the cost of customization or the quality of security, the security cost resulting from customization increases. It would be possible to use any functional form with increasing marginal cost, such as an exponential cost function, which would lead to the same qualitative results as the ones presented here.

We also consider the quality of security patch updates provided by AOSP, denoted by Q, to be an exogenous parameter in our model, which applies to all vendors in the same way. Note that we observe that in practice, vendors virtually never provide better security quality. Further, we are primarily interested in studying the effect of customization on security; hence, we will not consider vendors implementing additional security measures that are independent of customization. Hence, we assume that the value of q_i is upper bounded by Q.

Consumers: Consumers choose mobile devices primarily based on prices and how well the devices match their preferences, but they may also consider security quality. A consumer's preference, similar to a vendor's customization, can be represented by a point x in $[0, 1]$. Consumers' preferences for smartphone selection are heterogeneous and we assume that the consumers' preferences are distributed uniformly in $[0, 1]$. We consider security-conscious consumers who take security into account when choosing their product. The utility of consumer j for choosing

Android type i given that consumer j is at x_j is:

$$u_j^i = \beta q_i - p_i - T\left(x_j - z_i\right)^2, \tag{2}$$

where T represents the consumer's utility loss for one unit of difference between its preference and the location of the product, which we call *customization-importance*. Similarly, β represents the consumer's utility gain for one unit of security quality, which we call *security-importance*. Naïve consumers, who do not understand or care about security quality, can be modeled by letting $\beta = 0$.

Our utility for consumers is in agreement with literature in economics [5]. It is common to consider quadratic term in economics to model utility.

Game Formulation: For tractability, we consider a two-player game between vendor 1 and vendor 2 without any other vendors.[2] In our analysis, we assume that vendors are on different sides of AOSP. Further, we let $a = z_1$ and $1-b = z_2$. Without loss of generality, we assume that $0 \leq a \leq 1 - b \leq 1$. Figure 1 shows the location of vendor 1 and vendor 2.

The utilities of vendors 1 and 2 are then as follows:

$$\pi_1 = p_1 D_1 - C_1\left(a - Z_A\right)^2 - S_1 q_1^2\left(a - Z_A\right)^2, \tag{3}$$

$$\pi_2 = p_2 D_2 - C_2\left(1 - b - Z_A\right)^2 - S_2 q_2^2\left(1 - b - Z_A\right)^2. \tag{4}$$

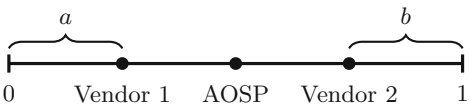


Fig. 1. Location of vendor 1 and vendor 2.

To calculate the Nash equilibrium, we need to define the stages of the game, i.e., the order in which the two players choose their prices, locations, and security levels. For our analysis, we consider the following stages:

- **Stage 1:** Both vendors simultaneously choose their location parameters a and b. They also choose their level of security quality, i.e., q_1 and q_2.
- **Stage 2:** Both vendors simultaneously choose their prices p_1 and p_2.

The reason is that a vendor freely modifies the AOSP code base, adds its developed proprietary software, and installs a diverse set of third-party apps to customize its device. These changes, however, result in the change of critical

[2] While we restrict our model to two vendors, we are aware that in practice, there are more than two vendors competing with each other. However, we believe that similar to classic economic studies with two companies in the context of product differentiation, our model provides a meaningful understanding of the customization in the Android ecosystem and of security quality.

configurations leading to security issues [23,25,26]. Therefore, it is reasonable to consider that the customization and security quality effort happen at the same stage. Then, by taking into account its effort in customization and security quality, the vendor chooses its price. We use backward induction to solve our game. First, we consider stage 2 and calculate the price Nash equilibrium for given locations and quality. Then, we consider stage 1 and calculate the location and quality equilibrium assuming a price equilibrium in stage 2.

Table 2 shows a list of the symbols used in our model.

Table 2. List of symbols

Symbol	Description
Z_A	Point corresponding to AOSP
D_i	Market share of vendor i
z_i	Customization of the Android version of vendor i
p_i	Price of vendor i
q_i	Security quality of patches delivered by vendor i
S_i	Cost per unit of security quality
C_i	Cost per unit of customization
π_i	Utility of vendor i
Q	Quality of security patch updates provided by AOSP
β	Consumer's security-importance
T	Consumer's customization-importance
x_j	Consumer's location
u_j^i	Utility of consumer j for choosing Android type i
q^{min}	Minimum level of security from the regulator's point of view
f_i	Fine function for vendor i
F	Monetary value of fine for each unit of violation from q^{min}

4 Analytical Results

In this section, we analyze our proposed model. Before considering the two stages, we first have to find the market shares of both vendors. To do so, we need to find the point in which a consumer j is indifferent between choosing vendor 1's product and vendor 2's product. This means that a user's preference at this point is identical for the two products. Hence, we have:

$$u_j^1 = u_j^2 \;\Rightarrow \beta q_1 - p_1 - T\left(x_j - a\right)^2 = \beta q_2 - p_2 - T\left(1 - b - x_j\right)^2. \tag{5}$$

Solving the above equation yields:

$$D_1 = x_j = a + \frac{1 - a - b}{2} + \frac{\beta\left(q_1 - q_2\right)}{2T\left(1 - a - b\right)} + \frac{p_2 - p_1}{2T\left(1 - a - b\right)}. \tag{6}$$

All of the consumers that are on the left side of x_j choose the product of vendor 1. As a result, the market share of vendor 1 is $D_1 = x_j$. This means that for equal prices and security qualities, vendor 1 controls its own "turf" of size a and the consumers located between vendor 1 and vendor 2 that are closer to vendor 1 than vendor 2. The last two terms represent the effect of security quality and price differentiation on the demand, respectively.

We restrict the model to consumers who definitely choose between these two products, which is a reasonable assumption for a wide range of parameters given the "cannot-live-without-it" desirability of modern phones, which is a valid assumption in economics, see [24]. Hence, the remaining consumers choose vendor 2's product, and its demand is accordingly:

$$D_2 = 1 - D_1 = b + \frac{1-a-b}{2} + \frac{\beta\,(q_2 - q_1)}{2T\,(1-a-b)} + \frac{p_1 - p_2}{2T\,(1-a-b)}. \tag{7}$$

If two vendors are at the same location, they provide functionally identical products. For a consumer who takes into account customization, price, and security quality, the factors that matter in this case are security quality and price. To increase their market share, vendors have to decrease their prices or increase their security quality. This will lead to lower product prices and higher costs due to higher security quality, and significantly lower – and eventually zero – utility for both vendors. Hence, vendors have no incentives for implementing customizations that result in identical product locations.

Price Competition: In the following, we state the price Nash equilibrium.

Theorem 1. *The unique price Nash equilibrium always exists, and it is*

$$p_1^* = \frac{\beta}{3}\,(q_1 - q_2) + T\,(1-a-b)\left(1 + \frac{a-b}{3}\right), \tag{8}$$

$$p_2^* = \frac{\beta}{3}\,(q_2 - q_1) + T\,(1-a-b)\left(1 + \frac{b-a}{3}\right). \tag{9}$$

The proof of Theorem 1 can be found in the extended version of the paper [7].

In Theorem 1, the price of a product depends on both the security quality and the customization level of both vendors. Further, the price depends on the customization importance T and security-importance β constants, which model the consumers in our model. A vendor can increase its price by improving its security quality or customizing its devices more.

Quality and Product Choice: To calculate the Nash equilibrium of both vendors in terms of location and security quality, we consider the following optimization problems.

Vendor 1 maximizes its utility in q_1 and a considering that p_1 is calculated according to Eq. 8. For vendor 1, we have:

$$\begin{aligned} \underset{a,\,q_1}{\text{maximize}} \quad & p_1^* D_1 - C_1\,(a - Z_A)^2 - S_1 q_1^2\,(a - Z_A)^2 \\ \text{subject to} \quad & p_1^* \geq 0,\ 0 \leq a \leq Z_A,\ 0 \leq q_1 \leq Q. \end{aligned} \tag{10}$$

The constraints in the above optimization problem reflect our previous assumptions about the parameters in our model definition. For each value of b and q_2, the solution of the above optimization problem provides vendor 1's best response. In a similar way, for vendor 2, we have:

$$\begin{aligned} \underset{b,\, q_2}{\text{maximize}} \quad & p_2^* D_2 - C_2 \left(1 - b - Z_A\right)^2 - S_2 q_2^2 \left(1 - b - Z_A\right)^2 \\ \text{subject to} \quad & p_2^* \geq 0,\ 0 \leq b \leq Z_A,\ 0 \leq q_2 \leq Q. \end{aligned} \tag{11}$$

For given values of a and q_1, the above optimization problem provides vendor 2's best response. Based on the Nash equilibrium definition, the intersection of these two optimization problems gives the Nash equilibrium of our proposed game, i.e., a^*, b^*, q_1^*, and q_2^*. In the extended version of the paper [7], we provide our method for solving these two optimization problems and for finding the Nash equilibrium.

Lemma 1. *When consumers take into account security, zero investment in security for both vendors, i.e., $q_1 = q_2 = 0$, is not a Nash equilibrium.*

The proof of Lemma 1 is provided in the extended version of the paper [7].

The above lemma shows that when consumers take into account security, then vendors have to invest to improve their security quality. However, it is challenging for the majority of consumers to measure by themselves the security quality of a product, or in this case, to make a comparison between the security quality of many customized versions of Android provided by the vendors. Consumers mainly rely on information that is made available to them.[3] However, in the absence of any reliable market signal, any unsubstantiated communication/advertisements by vendors about security quality have to be considered with caution.[4]

Previous research has shown that businesses aim to exploit such information barriers. In particular, the theory of *informational market power* posits that when it is hard for consumers to understand and/or observe certain features of a product (e.g., security quality), then businesses are incentivized to underinvest in these product features and rather focus on easily observable aspects such as product design and price [3].[5] Any effect of informational market power is emphasized by well-known human biases such as *omission neglect* [18]. This

[3] While we have identified a small set of research projects which aim to understand the security impact of customization, e.g., [23,25,26], we are unaware of any well-known market signals regarding the security of different Android versions. The recent FTC initiative to solicit security-relevant data from vendors may contribute to such signals in the future [8].

[4] In fact, research by Wu et al. shows that vendors of different reputation (which may also influence perceptions regarding Android security) all suffer from similar challenges due to Android customization [25].

[5] Note that it is not required that businesses have an accurate assessment of the security quality of their own product (or competitors' products) for informational market power to be exploited.

describes the human lack of sensitivity about product features that are not the focus of advertisements or product communications; to paraphrase, consumers will often not consider in their *perceived utilities* product features which are not emphasized. Therefore, we consider an important baseline case of naïve consumers. In particular, we show that our model is in agreement with what we have seen in practice, i.e., vendors do not invest in security when consumers are naïve. Further, we determine under what conditions maximal differentiation, i.e., $a^* = b^* = 0$, is Nash equilibrium; see the extended version of the paper [7].

5 Parameter Selection

In the previous section, our analyses were focused on six variables: p_1, p_2, q_1, q_2, a, and b. In addition to these six variables, we have six parameters, which are β, T, C_1, C_2, S_1, and S_2. In this section, we discuss how we can quantify these six parameters in practice. In doing so, we use a reverse approach. First, we measure the values of p_1, p_2, q_1, q_2, a, and b. Then, based on our analyses in the previous section, we calculate the values of the constants in our model.

Table 3. Origin of apps in two devices [25].

Vendor	Device	Version and Build#	#apps	#LOC	AOSP		Vendor		3rd-party	
					#apps	#LOC	#apps	#LOC	#apps	#LOC
HTC (vendor 1)	One X	4.0.4; CL100532	280	19M	29	4.7M	190	7.3M	61	7.5M
Samsung (vendor 2)	Galaxy S3	4.0.4; 19300UBALF5	185	17M	30	6.3M	119	5.6M	36	5.3M

Location Quantification: In order to quantify customization and map it to a location, we need to quantify how different two Android versions are in terms of pre-loaded apps. To do so, we can access the image of an Android OS version, e.g., see [25] and [1], and investigate how many apps a vendor has developed for a specific version.

To quantify customization, we use the results of Table 3 and calculate the proportion of the code that was developed by a vendor. Note that in our model, we assume that the locations are in the interval $[0, 1]$. First, we need to specify the location of Z_A and then select the locations of the other vendors. Here, we assume that $Z_A = 0.5$. For the HTC One X (i.e., vendor 1), 7,354,468 LoC were developed by the vendor and 7,550,704 LoC stem from third-party apps. This means that about 75.95% LOC were added by that vendor to the baseline AOSP version. Here, we interpret this number as the level of difference between the device and AOSP. In order to keep the value of a in the interval $[0, 0.5]$, we let $a = Z_A - (percentage/2)$. Therefore, we have $a = Z_A - (0.7595/2) = 0.1203$. In a similar way, for the Samsung Galaxy 3 (i.e., vendor 2), 5,660,569 LOC were developed by the vendor in addition to 5,334,152 LoC coming from third-party

apps, which is equal to 63.41% of the total number of LOC. In a similar way, we let $b = 1 - Z_A - (0.6341/2) = 0.1830$.

Quality: To quantify q_1 and q_2, we use the analysis reported in [25]. For a sample of 10 devices, they found that the maximum number of vulnerabilities for a device is 40. Some of these vulnerabilities are the result of vendor customization. For the HTC One X, 15 vulnerabilities were found, and 10 of these vulnerabilities are due to vendor customization. By dividing the number of vulnerabilities resulting from customization with the maximum number of vulnerabilities, we get 0.25. In order to calculate security quality, we let

$$q_1 = 1 - \frac{\#Customization\,Vulnerabilities}{Maximum\#Vulnerabilities} = 0.75.$$

In a similar way, for the Samsung Galaxy S3, 40 vulnerabilities were found, and 33 of these are the result of customization. Hence, we have $q_2 = 1 - \frac{33}{40} = 0.1750$.

Price: The prices of the HTC One X and the Samsung Galaxy S3 are equal to €170 [13] and €190 [14], respectively. GSM Arena (http://www.gsmarena.com/) groups both of these devices as group 4 out of 10 for their price. Here, we consider $p_1 = p_2 = 4$.

Parameter Estimation: By inserting these six values into our model analysis, we can calculate the six constants in our model. Here, we assume that both vendors are completely rational and as a result they have chosen their customization levels, prices, and security qualities following the dependencies captured by our model. Therefore, we can calculate the parameters in our model in a reverse way. By inserting our quantified parameters, i.e., a, b, q_1, q_2, p_1, and p_2, into Eqs. 8 and 9, we have two equations and two variables, i.e., β and T. The system of equations then yields the values of β and T. Note that these two equations are linear in T and β. Therefore, the resulting answer is unique.

To calculate the values of C_1, C_2, S_1, and S_2, we assume that the measured values of q_1, q_2, a, b form a Nash equilibrium of our game. Since the vendors' strategies are mutual best responses in a Nash equilibrium, q_1, q_2, a, b are solutions to the corresponding best-response equations, which are available in the extended version of the paper [7]. We have four variables and four equations. The solution of this system of equations provides the values of S_1, C_1, S_2, C_2, which are unique. Therefore, based on the measured values, we have $\beta = 0.4362$, $T = 5.7414$, $S_1 = 0.6723$, $C_1 = 1.4882$, $S_2 = 4.1338$, and $C_2 = 2.4875$.

6 Fine Model and Analysis

The background on actual security practices and our analysis provide evidence and explanations for vendors' unsatisfactory security practices in the context of customization. In particular, vendors will not adequately invest in security if consumers do not take security sufficiently into account. In the following, we propose a mechanism to incentivize a vendor to invest in security quality.

In doing so, we introduce a regulator whose role is to define a corresponding policy. More specifically, we propose the following fine function for vendor i for a regulatory policy which takes as input the vendor's security quality and outputs the monetary value of the fine imposed on the vendor:

$$f_i(q_i) = \begin{cases} F\left(q^{min} - q_i\right) & \text{if } q^{min} \geq q_i \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where F and q^{min} are constants defined by the regulator. q^{min} is the minimum acceptable level of security from the regulator's point of view and the regulator tries to force each vendor to satisfy at least this security level. F is a coefficient relating quality to monetary value and denotes the monetary value of fine for each unit of security violation from q^{min} for a vendor. The monetary value of the fine should be proportional to the market share, since a higher market share of a vendor with security issues results in a higher number of consumers with vulnerabilities. In our model, we multiplied f_i by the market share of that vendor.

In this section, we show that under certain conditions, a regulator can force a vendor to spend on security issues resulting from customization. Moreover, we prove that the product's price *decreases* as the vendor invests in the adequate level of security imposed by the regulator, for the same value of customization cost. More specifically, our analysis shows that under some conditions, the higher the security quality imposed by the regulator is, the lower the product's price is.

By imposing a fine, the vendors' utilities change to the following:

$$\pi_1 = p_1 D_1 - C_1 \left(a - Z_A\right)^2 - S_1 q_1^2 \left(a - Z_A\right)^2 - f_1 D_1. \tag{13}$$

$$\pi_2 = p_2 D_2 - C_2 \left(1 - b - Z_A\right)^2 - S_2 q_2^2 \left(1 - b - Z_A\right)^2 - f_2 D_2. \tag{14}$$

It is worth mentioning that the consumers' utility does not change. Hence, all of Eqs. 5, 6, and 7 are still valid for the case when there is a fine. The validity of these equations implies that the formulae for the vendors' market share is the same for both cases. However, the vendors' equilibrium prices are different compared to the previous case.

Similar to the case without a fine, here we have the same two stages with the same ordering. The regulator's goal is to force the vendors to invest in an adequate security level. Hence, in our analysis, we focus on the case where the regulator forces the vendor to invest in an adequate security quality level.

Theorem 2 characterizes both vendors' prices in Nash equilibrium when the regulator imposes a fine.

Theorem 2. *The Nash equilibrium in prices, which always exists, is*

$$p_1^* = \frac{\beta}{3}\left(q_1 - q_2\right) + T\left(1 - a - b\right)\left(1 + \frac{a - b}{3}\right) + \frac{2f_1}{3} + \frac{f_2}{3}, \tag{15}$$

$$p_2^* = \frac{\beta}{3}\left(q_2 - q_1\right) + T\left(1 - a - b\right)\left(1 + \frac{b - a}{3}\right) + \frac{2f_2}{3} + \frac{f_1}{3}. \tag{16}$$

This proof of the theorem is included in the extended version of the paper [7].

By comparing the above two equations with Eqs. 8 and 9, we observe that the introduction of a fine will increase the product price of the vendors for fixed locations and security level.

Naïve Consumers: Based on Theorem 2, by letting $\beta = 0$, we can characterize the price NE for naïve consumers (see the extended version of the paper [7]). Lemma 2 introduces the sufficient conditions to force vendors to invest in adequate level of security, when consumers do not take security into account.

Lemma 2. *Both vendors invest in $q_1^* = q_2^* = q^{min}$, if the following conditions are satisfied for the optimal locations of both vendors:*

$$F^2 - 18TS_1\,(1-a-b)\,(a-Z_A)^2 \geq 0, \tag{17}$$

$$F^2 - 18TS_2\,(1-a-b)\,(1-b-Z_A)^2 \geq 0, \tag{18}$$

$$3 + a - b - \frac{Fq^{min}}{T\,(1-a-b)} \geq 0. \tag{19}$$

Proof of the above lemma is provided in the extended version of the paper [7].

Lemma 3 calculates the location Nash equilibrium of both vendors considering that the regulator forces the vendors to invest in adequate levels of security.

Lemma 3. *For a given b, vendor 1's best response for location, when the consumers do not take security into account and conditions of Lemma 2 are satisfied, is as follows:*

- $C_1 + S_1\left(q^{min}\right)^2 \leq \frac{T}{12Z_A}$*: Vendor 1 differentiates its product the most, i.e., $a^*(b) = 0$.*
- $C_1 + S_1\left(q^{min}\right)^2 \geq \frac{T}{9Z_A}$*: The positive root of the following quadratic equation is called a_2. In this case, for vendor 1 we have $a^*(b) = \min\{a_2, Z_A\}$.*

$$\begin{aligned} -3Ta^2 + a\left(2Tb - 10T - 36\left(C_1 + S_1\left(q^{min}\right)^2\right)\right) \\ +T\left(b^2 - 2b - 3\right) + 36\left(C_1 + S_1\left(q^{min}\right)^2\right)Z_A = 0 \end{aligned} \tag{20}$$

- $\frac{T}{12Z_A} < C_1 + S_1\left(q^{min}\right)^2 < \frac{T}{9Z_A}$ *and* $b \leq \min\{1 - \sqrt{4 - \frac{36\left(C_1+S_1(q^{min})^2\right)}{T}}Z_A, Z_A\}$*: Vendor 1 chooses its location as $a^*(b) = \min\{a_2, Z_A\}$.*
- $\frac{T}{12Z_A} < C_1 + S_1\left(q^{min}\right)^2 < \frac{T}{9Z_A}$ *and* $1 - \sqrt{4 - \frac{36\left(C_1+S_1(q^{min})^2\right)}{T}}Z_A \leq Z_A$ *and* $1 - \sqrt{4 - \frac{36\left(C_1+S_1(q^{min})^2\right)}{T}}Z_A \leq b \leq Z_A$*: Vendor 1 differentiates its product the most, i.e., $a^*(b) = 0$.*

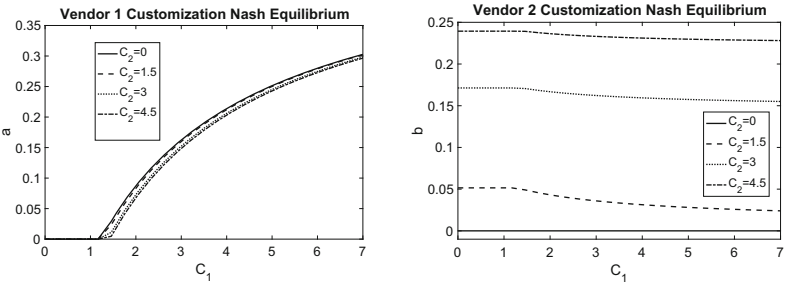
By changing C_1 to C_2, S_1 to S_2, a to b, and Z_A to $(1 - Z_A)$, in the above lemma, we can derive the same results for vendor 2. Proof of Lemma 3 is provided in the extended version of the paper [7].

Comparing Lemma 3 with the case without fine, maximal differentiation occurs when the customization cost is lower than when there is no fine, since a

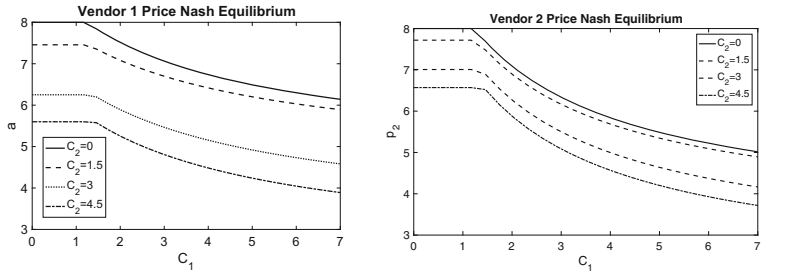
vendor's cost is affected by both the costs of customization and security quality. Further, according to Eq. 20, the location NE depends on q^{min} rather than F. However, both F and q^{min} have the effect to satisfy the conditions for forcing a vendor to invest in an adequate level of security, i.e., Lemma 2.

7 Numerical Illustration

In this section, we evaluate our findings numerically. First, we evaluate the case in which consumers are naïve, but a regulator imposes fines. Then, we compare the equilibrium prices and locations in the absence and in the presence of the regulatory fine. Interestingly, we observe that the *products' prices* (of both vendors) *decrease* in the presence of fines, and both vendors invest in the minimum level of security q^{min} set by the regulator. Finally, we evaluate the case in which there are no fines but the consumers take into account security quality.



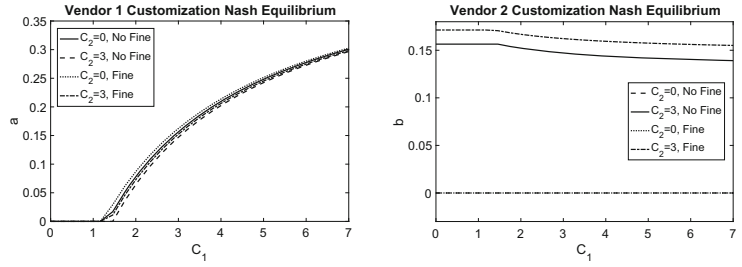
(a) a^* (b) b^*



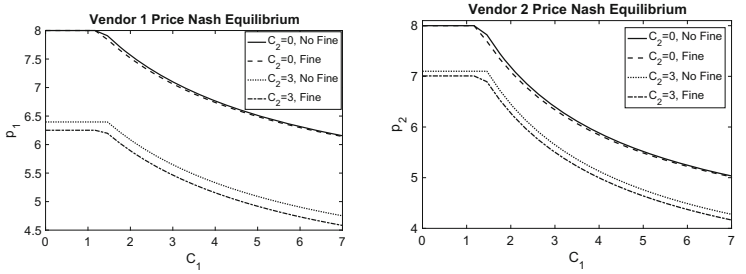
(c) p_1^* (d) p_2^*

Fig. 2. Equilibrium locations and prices for various values of C_1 and C_2. Here, consumers are naïve, but there is a regulatory fine, and $T = 8$, $S_1 = 0.602$, $S_2 = 1.54$, $F = 10$, and $q^{min} = 0.4$. For these values of C_1 and C_2 and choices of a and b, both vendors invest in q^{min} set by the regulator.

In Fig. 2, we examine the effect of regulation on location, price, and security quality for various values of C_1 and C_2. In our evaluation, in the presence of a regulator, the conditions of Lemma 2 are satisfied. As a result, both vendors invest in q^{min} set by the regulator. Similar to the case without any fine, the higher the customization cost (e.g., C_1), the lower is the differentiation from the baseline AOSP (e.g., the higher the value of a^* is). Similarly, we again observe little changes in a vendor's location in response to changes in its opponent's customization cost and the customization level. Further, the equilibrium prices of both vendors are decreased by an increase in customization costs, since both of them choose lower levels of customization and enter a price competition. Note that even in the presence of fines, vendor 2 chooses the maximum level of customization (i.e., $b^* = 0$) when $C_2 = 0$, considering that its cost of security is proportional to its level of customization and $S_2 > S_1$. The reason for this is that vendor 2 is reluctant to enter a price competition.



(a) a^* (b) b^*



(c) p_1^* (d) p_2^*

Fig. 3. Comparison between the presence and the absence of a fine for naïve consumers. We have $T = 8$, $S_1 = 0.602$, $S_2 = 1.54$, $F = 10$, and $q^{min} = 0.4$.

In Fig. 3, we compare the equilibria in the presence and the absence of a regulatory fine, when consumers do not take security into account. Based on Fig. 3(c), vendor 1 chooses a lower level of customization when a fine exists.

Figure 3(d) shows that vendor 2 chooses the same location in both cases as we discussed earlier when $C_2 = 0$. For $C_2 = 3$, vendor 2 chooses a lower level of customization (i.e., higher value of b^*) compared to the case when there is no fine. Consequently, the prices of both vendors are lower for higher customization costs due to the fact that both vendors are moving closer to the AOSP baseline model. Moreover, the existence of regulation and the fine leads to higher values of a^* and b^* (i.e., lower customization levels) for the same customization costs compared to the case without a fine, since each vendor tries to maximize its utility by avoiding a regulatory fine through investing in the minimum level of security quality q^{min}. Therefore, each vendor has to pay both the cost of customization as well as the cost of security quality resulting from customization. To decrease these costs, each vendor chooses a lower level of customization. Further, choosing higher values of a^* and b^* (i.e., lower customization level) leads to lower prices for both vendors. Therefore, the existence of a regulatory fine leads to **more secure products** at **lower prices** when consumers cannot evaluate security properties by themselves.

To find equilibrium locations and security qualities when consumers take security into account but there is no fine, we calculate each vendor's best-response security quality and location for its opponent's given location and security quality. Then, the Nash equilibrium is the intersection of these best responses. Table 4 shows the equilibrium for various values of C_1, where $T = 1.6$, $\beta = 0.6$, $C_2 = 1.3$, $Q = 1$, and $S_1 = S_2 = 1$. In this case, due to the consumers' security considerations, both vendors invest in security. For $C_1 = 0$, vendor 1 maximizes its differentiation from baseline AOSP. Because of the consumers' security awareness, vendor 1 invests in security, but at a lower level than Q. It is interesting to see that vendor 2 does not differentiate its product from the baseline AOSP version due to maximal differentiation of vendor 1 and consequently, it does not have any security issues resulting from customization (i.e., $q_2^* = Q = 1$).

Table 4. The vendors' equilibrium prices and security qualities for various values of C_1. Here, we have $S_1 = S_2 = 1$, $T = 1.6$, $\beta = 0.6$, $Q = 1$, and $C_2 = 1.3$.

C_1	a^*	q_1^*	b^*	q_2^*
0	0	0.2612	0.5	1
0.3684	0.2888	1	0.3639	1
0.7368	0.3195	1	0.3638	1
1.1053	0.3452	1	0.3637	1
1.4737	0.3677	1	0.3636	1

It is noteworthy that both vendors invest in the maximum level of security when both vendors' customization costs are greater than zero. This observation shows that if all consumers are capable of measuring security quality and it

is one of the factors affecting their product choice, then vendors will invest in security. Similar to the case where consumers do not take security into account, the higher the customization cost, the lower is the customization level. In other words, increasing C_1 results in higher values of a^*. Moreover, changing the value of C_1, while C_2 is fixed, results in little changes in b^*.

8 Conclusion

Our model shows that vendors have to invest in security quality for security-conscious consumers. Further, for naïve consumers, our proposed model captures the fact that vendors underinvest in security. To incentivize vendors to invest in security for naïve consumers, a regulator may assign a fine to those vendors that do not uphold a desired level of security, which is a well-motivated scenario given Android-related FTC actions [9].

We show that the imposed fine structure achieves the expected effect in addition to changes in the competitive landscape. First, the price of the product decreases for the same cost of customization compared to the case without any fine. Second, a higher level of security quality imposed by the regulator leads to a lower product price, if certain conditions are satisfied. Our findings suggest that requiring higher baseline levels of security investments (as triggered by recent FTC actions [9]) does not impose higher product prices on naïve consumers, which is important from a technology policy perspective. Moreover, increasing consumers' attention about security is substantiated by our analysis as a positive and meaningful factor to address challenges related to informational market power and neglected security efforts.

Acknowledgments. We thank the anonymous reviewers for their comments. The research activities of Jens Grossklags are supported by the German Institute for Trust and Safety on the Internet (DIVSI). Aron Laszka's work was supported in part by the National Science Foundation (CNS-1238959) and the Air Force Research Laboratory (FA 8750-14-2-0180).

References

1. Aafer, Y., Zhang, X., Du, W.: Harvesting inconsistent security configurations in custom Android ROMs via differential analysis. In: USENIX Security Symposium (2016)
2. Android market share: Android market share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Accessed 11 Apr 2018
3. Beales, H., Craswell, R., Salop, S.C.: The efficient regulation of consumer information. J. Law Econ. **24**(3), 491–539 (1981)
4. Cavusoglu, H., Raghunathan, S.: Selecting a customization strategy under competition: mass customization, targeted mass customization, and product proliferation. IEEE Trans. Eng. Manag. **54**(1), 12–28 (2007)
5. d'Aspremont, C., Gabszewicz, J., Thisse, J.-F.: On Hotelling's "Stability in competition". Econometrica **47**(5), 1145–1150 (1979)

6. Dewan, R., Jing, B., Seidmann, A.: Product customization and price competition on the internet. Manag. Sci. **49**(8), 1055–1070 (2003)
7. Farhang, S., Laszka, A., Grossklags, J.: An economic study of the effect of Android platform fragmentation on security updates. arXiv preprint arXiv:1712.08222 (2017)
8. Federal Trade Commission: FTC to study mobile device industry's security update practices. https://www.ftc.gov/news-events/press-releases/2016/05/ftc-study-mobile-device-industrys-security-update-practices. Accessed 11 Apr 2018
9. Federal Trade Commission: HTC America settles FTC charges it failed to secure millions of mobile devices shipped to consumers. https://www.ftc.gov/news-events/press-releases/2013/02/htc-america-settles-ftc-charges-it-failed-secure-millions-mobile. Accessed 11 Apr 2018
10. Felt, A., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: user attention, comprehension, and behavior. In: Symposium on Usable Privacy and Security, pp. 3:1–3:14 (2012)
11. Gabszewicz, J., Thisse, J.-F.: Price competition, quality and income disparities. J. Econ. Theory **20**(3), 340–359 (1979)
12. Galbraith, J.: The New Industrial State. Princeton University Press, Princeton (2015)
13. GSMarena: HTC One X price. http://www.gsmarena.com/htc_one_x-4320.php. Accessed 11 Apr 2018
14. GSMarena: Samsung Galaxy S3 price. http://www.gsmarena.com/samsung_i9300_galaxy_s_iii-4238.php. Accessed 11 Apr 2018
15. Han, D., Zhang, C., Fan, X., Hindle, A., Wong, K., Stroulia, E.: Understanding Android fragmentation with topic analysis of vendor-specific bugs. In: 19th Working Conference on Reverse Engineering, pp. 83–92 (2012)
16. Hotelling, H.: Stability in competition. Econ. J. **39**(153), 41–57 (1929)
17. Kaldor, N.: The economic aspects of advertising. Rev. Econ. Stud. **18**(1), 1–27 (1950)
18. Kardes, F.: Omission neglect. In: Baumeister, R., Vohs, K. (eds.) Encyclopedia of Social Psychology, vol. 1. Sage (2007)
19. Mutchler, P., Safaei, Y., Doupé, A., Mitchell, J.: Target fragmentation in Android apps. In: IEEE Security and Privacy Workshops, SPW, pp. 204–213 (2016)
20. Mylonas, A., Kastania, A., Gritzalis, D.: Delegate the smartphone user? Security awareness in smartphone platforms. Comput. Secur. **34**, 47–66 (2013)
21. Salop, S.: Monopolistic competition with outside goods. Bell J. Econ. **10**(1), 141–156 (1979)
22. Singh, S.: An analysis of Android fragmentation. http://www.tech-thoughts.net/2012/03/analysis-of-android-fragmentation.html#.WA_OxoMrKUk. Accessed 11 Apr 2018
23. Thomas, D., Beresford, A., Rice, A.: Security metrics for the Android ecosystem. In: ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 87–98 (2015)
24. Tirole, J.: The Theory of Industrial Organization. MIT Press, Cambridge (1988)
25. Wu, L., Grace, M., Zhou, Y., Wu, C., Jiang, X.: The impact of vendor customizations on Android security. In: ACM Conference on Computer & Communications Security, pp. 623–634 (2013)
26. Zhou, X., Lee, Y., Zhang, N., Naveed, M., Wang, X.: The peril of fragmentation: security hazards in Android device driver customizations. In: IEEE Symposium on Security and Privacy, pp. 409–423 (2014)

The Rules of Engagement for Bug Bounty Programs

Aron Laszka[1], Mingyi Zhao[2(✉)], Akash Malbari[3], and Jens Grossklags[4]

[1] University of Houston, Houston, USA
[2] Snap Inc., Santa Monica, USA
mingyi.zhao@snap.com
[3] Pennsylvania State University, University Park, USA
[4] Technical University of Munich, Munich, Germany

Abstract. White hat hackers, also called ethical hackers, who find and report vulnerabilities to bug bounty programs have become a significant part of today's security ecosystem. While the efforts of white hats contribute to heightened levels of security at the participating organizations, the white hats' participation needs to be carefully managed to balance risks with anticipated benefits. One way, taken by organizations, to manage bug bounty programs is to create rules that aim to regulate the behavior of white hats, but also bind these organizations to certain actions (e.g., level of bounty payments). To the best of our knowledge, no research exists that studies the content of these program rules and their impact on the effectiveness of bug bounty programs.

We collected and analyzed the rules of 111 bounty programs on a major bug bounty platform, HackerOne. We qualitatively study the contents of these rules to determine a taxonomy of statements governing the expected behavior of white hats and organizations. We also report specific examples of rules to illustrate their reach and diversity across programs. We further engage in a quantitative analysis by pairing the findings of the analysis of the program rules with a second dataset about the performance of the same bug bounty programs, and conducting statistical analyses to evaluate the impact of program rules on program outcomes.

1 Introduction

Opposing malicious hackers, so-called *white hats* or *ethical hackers* strive to contribute to the security efforts of organizations by finding and reporting security vulnerabilities. Various factors motivate white hats, such as monetary benefits, increasing reputation, or job opportunities. Others aim to acquire knowledge by identifying bugs in systems.

White hats frequently work for specific target organizations under the umbrella of paid or unpaid bug bounty *programs* [9,12,25]. Further, these programs are now often facilitated by bug bounty *platforms* such as HackerOne, BugCrowd, Cobalt, etc. As part of bug bounty programs, organizations allow

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 138–159, 2018.
https://doi.org/10.1007/978-3-662-58387-6_8

white hats to perform ethical hacking on their systems, to identify the loopholes that their internal security teams could not identify (given personnel, time, expertise, and cost constraints) and which could become important targets of black hats. Platforms facilitate the process by, for example, managing the payment of bounties, serving as a point of contact for conflicts between white hats and bug bounty programs or even law enforcement, and perhaps most importantly, acting as a central point of attraction for white hats and organizations alike.

In this paper, we are contributing to existing work on vulnerability discovery by conducting the first study of bug bounty programs' *rules of engagement*, which are the program rules governing the interaction between white hats and organizations. They fulfill at least two key functions. First, they state for each bug bounty program the expectations regarding white hats' behaviors when they engage in vulnerability discovery on the program's site, and when they submit vulnerability reports to the program. Second, they also bind organizations to certain actions, e.g., the size of bounty payments for specific types of discovered vulnerabilities, expected speed of resolving identified issues, etc.

A careful management of these various factors specified in the program rules may also help to address two key problems, a high number of reports that are later categorized as *invalid* [13] and a high number of *duplicate* findings [26]. The annual reports from bug bounty platforms show that the resulting outcomes can be quite inefficient (see, for example, [5,6]).

Going beyond the direct relationship between a program and white hats, the rules likely also shape the competitive process between the different programs on a platform. The stated terms may contribute to attract white hat researchers, or they may dissuade them from working for a particular program. All these factors are so far largely unexplored.

In this paper, we collected and analyzed the program rules of 111 bounty programs on a major bug bounty platform, HackerOne. We qualitatively study the contents of these rules to determine a taxonomy of statements governing the expected behavior of white hats and organizations. We also report specific examples of rules to illustrate their reach and diversity across programs. We further engage in a quantitative analysis by pairing the findings of the analysis of the program rules with a second dataset about the performance of the same bug bounty programs, and conducting statistical analyses to evaluate the impact of program rules on program outcomes.

We will proceed as follows. In Sect. 2, we discuss related work. In Sect. 3, we describe our dataset. In Sect. 4, we present qualitative results. We then provide a quantitative investigation of the program rules and program performance including a regression analysis in Sect. 5. In Sect. 6, we discuss various aspects of the rules along with some suggestions for program improvements. We offer concluding remarks in Sect. 7.

2 Related Work

In the academic domain, several studies have focused on the discovery of software vulnerabilities (e.g., [4,7,17,18,21,23]), and software vulnerability markets [1–3,19]. And, in recent years, there has also been a growing research interest in bug bounty programs. Researchers have conducted multiple empirical analyses of independently run bug-bounty programs. Studying the incentives and practices of organizations and white hats initiating and participating in such programs, respectively, is crucial to understand their economic viability and impact on security. For example, Finifter et al. empirically investigated the Google Chrome and Mozilla bug-bounty programs [9], and suggested that these programs are more cost-effective compared to hiring full-time security researchers in terms of finding security flaws. In an effort to better understand the human side of vulnerability discovery, Edmundson et al. conducted an experiment where participants were asked to identify seven security vulnerabilities embedded in a code sample, but no participant was able to accomplish this task alone. However, when the researchers collected a random sample of 50% of the participants, the probability of finding all bugs increased to 95% [8].

Researchers have also studied the dynamics of bug bounty platforms [11,15, 25]. Zhao et al. conducted a comprehensive study of two emerging bug bounty platforms, Wooyun and HackerOne, to understand their characteristics, trajectories and impact [25]. One key finding of their work is that not only top contributors are important for bug bounty platforms, but that also the long tail of white hat researchers makes significant and diverse contributions (as a group). In follow-up research, Maillart et al. empirically studied reward distribution and hacker enrollments of public bounty programs on HackerOne and found that growing rewards cannot match the increasing difficulty of vulnerability discovery, and thus hackers tend to switch to newly launched programs to find bugs more easily [15]. Our work may help to understand how programs aim to attract (or retain) white hats by offering attractive rules of participation.

These research efforts also helped to identify hurdles which may limit the further growth of bug-bounty platforms and programs. The data suggests that bug-bounty platforms suffer from a high rate of submitted reports which for a variety of reasons are considered invalid. The percentage of invalid reports is currently significant, ranging from 35% to 55% on different platforms [14]. Programs may try to regulate the in-flow of invalid reports by adjusting the rules and incentives to discourage certain behaviors [13]. Further, due to the decentralized nature of vulnerability discovery, white hats may discover the same issues and file reports which are then recorded as so-called duplicates [14]. This problem can potentially be alleviated by designing an allocation plan for white hats' efforts and diversifying the workforce [26].

We are unaware of any work which has investigated the program rules for vulnerability discovery to complement the aforementioned research efforts.

3 Dataset

Since its inception in 2012, HackerOne has continuously grown as a community and has been attracting numerous white hats and organizations to participate in its bounty platform. By September 2017, white hats on HackerOne have successfully contributed over 50,000 bug reports (which have been fixed), and have been paid bounties totaling over $20 million. Participating organizations have thanked over 4,500 hackers.

We collected data in January 2016. For the (then available) 112 public programs on HackerOne's website, we downloaded the program descriptions and the announced minimum bounty paid by the program. Of the 112 organizations, 52 register themselves with a clause of paying *"no minimum bounty."* In our further analysis, we consider 111 organizations by excluding HackerOne, which runs a bounty program on its own platform, and consider only outside organizations participating on HackerOne.

Of these 111 programs, we were able to download the *detailed history* of rule description changes, bugs resolved, and hackers thanked for 77 programs. For each one of these programs, we determined the date of the last major update to the rule description before January 2016[1], and we counted the number of bugs resolved and hackers thanked in the interval between the last major rule update and January 2016. By dividing these numbers with the length of the time interval, we were able to determine the rate of hackers thanked and bugs resolved for the program description that was in effect in January 2016.

In the subsequent sections, we focus on the program descriptions of the bounty programs, which include the rules of engagement, and provide a qualitative analysis of their contents. We also conduct several analyses correlating discernible features of these program rules with important metrics such as the number of bugs resolved and hackers thanked.

We further pair the aforementioned data with an additional dataset. We collected, following the methodology of previous research [15,24,25], data about the average bounty and the age of bounty programs, and their Alexa ranks (which is based on web traffic data). This additional data is used for a deeper assessment of the rules' impact in a regression analysis.

We are also aware of the limitations of this dataset. One limitation is that we do not have data for private bug bounty programs, which are only accessible to a selected group of (internal) researchers. In addition, other competing bug bounty platforms, such as BugCrowd, could affect white hat behaviors on HackerOne as well. However, our current dataset does not include these competing platforms.

4 Qualitative Study

On a high level, each line of a program description carries with it some meaning and may provide important information to white hats who are interested in a

[1] We used the Python `difflib` implementation of the Ratcliff/Obershelp matching algorithm [20] with parameter 0.9.

particular bug bounty program. On HackerOne, each organization has been given the liberty to specify the description of the program in its own way, which on the one hand promotes diversity, but on the other hand may complicate matters for white hats and may affect the comprehensiveness of the information covered. Since there is no framework for structuring rules, the necessary first step in studying rules is to construct a *general taxonomy*. In this section, we aim to provide such a taxonomy of the contents of the rules on the basis of "what they are trying to convey" to capture a generic structure for the program description.

To achieve this objective, the program descriptions were parsed statement by statement and in iterations to extract information and to tabularize the different statements contained in the rules according to the evolving taxonomy. We also cross-verified the extracted information to check whether we followed a consistent classification process. If a statement in the rules was found to fit in more than one category, then it was marked accordingly. If a statement conveyed no concrete guidance to the white hat, we categorized it under "other instructions." Based on this process, the rules specified by the 111 organizations could be categorized and defined according to the following taxonomy. To save space, we list example rule statements in Appendix A and only reference them in the main text.

4.1 In-scope Areas

Statements of this type define the exact scope of the bug-bounty program. The organizations state typically the list of system and product areas on which white hats should work. The majority of the organizations will list their core *production websites* as the target of bug bounty. Some organizations also list *staging websites*, and encourage or require white hats to conduct vulnerability research only against them (Example 1). Some organizations may also provide source code to white hats (Example 2). In addition to web applications, there may be other types of components, such as *APIs*, *mobile applications*, and *desktop applications*, which are in scope. Further, vulnerability discovery may also extend to *physical products* with digital components. For example, ToyTalk allows the search for "a security issue in our products or service" which include a doll and a playhouse with voice capabilities.

4.2 Out-of-scope Areas

Each organization can also explicitly specify all the domains and areas that they do *not* wish the white hats to work on. We have identified the following reasons for listing an area as out-of-scope. First, organizations typically exclude web applications (e.g., blog, support, community, etc.) hosted by third-parties, as these websites are not controlled by the organization, and/or have low risk (e.g., no user data) (Example 3). Second, customer websites or services are usually out of scope as well (Example 4). Third, some organizations also exclude areas that belong to business partners or subsidiaries (Example 5). Fourth, as discussed above, organizations may set up staging sites, but also *explicitly* declare their production site to be out of scope (Example 6).

4.3 Eligible Vulnerabilities

This category provides additional detailed rules focused on the types of vulnerabilities which the organizations want the white hats to find. In general, organizations encourage white hats to spend their effort on those types of vulnerabilities which they likely consider of particular severity to their organizations. Frequently mentioned vulnerability types across many organization are: SQL Injection, Remote Code Execution, Cross-Site Request Forgery, Directory Traversal, Cross-Site Scripting, Information Disclosure and Logical Issues.

Rules can be fairly precise and may even include additional conditions such as the potential for financial damage. One example is from Coinbase (an exchange for digital assets) (Example 7). However, some organizations may not rely on specifying a set of vulnerability types. For example, Envoy's rules state that reports should be about "*issues that are very clearly security problems.*"

4.4 Non-eligible Vulnerabilities

Certain types of vulnerabilities are often excluded from being rewarded with a bug-bounty, because they have very low or no security risk from the perspective of an organization. Frequently mentioned examples include Self-XSS, Logout CSRF, no maximum password length, etc. Some of these issues may also be rather easy to identify for a white hat. Listing such non-eligible vulnerabilities in an upfront manner can reduce the cost of processing reports which may even be declared invalid. Please note that invalid reports are very common, and a significant challenge to bug bounty programs [13]. Denial of Service (DoS) is another typical type of non-eligible vulnerability as some organizations already know their general vulnerability to this type of attack, or doubt that any white hat report would yield novel insights (Example 8).

4.5 Deepening Engagement with Organizations

This category includes further instructions to the white hats (going beyond the scope and eligible vulnerabilities categories) in regards to how they can better engage in vulnerability research for the organization. Specifying such instructions helps the white hats to align their effort with the organization's interest, and to more likely find bugs which will be rewarded. For example, some organizations ask white hats to create dedicated test accounts (Example 9). Another interesting case is Square's Capture-the-flag (CTF) challenge within its bug-bounty program. Basically, Square hides a secret flag inside its system, and whoever finds it can qualify for a $1,000 reward. Understanding the impact of such mechanisms on white hat engagement is an interesting aspect for future work.

4.6 Prohibited or Unwanted Actions

Rules in this category list further instructions to the white hats in regards to what they should not do (going beyond the scope restrictions and non-eligible

vulnerabilities) when they are searching for vulnerabilities for the organizations. These instructions specify detailed bounds to the work of white hats which organizations may use to protect their business interests, while participating in crowdsourced security research.

There are several subcategories within this rule. First, many organizations forbid or limit the use of automated scanning, because they can lead to a large amount of false positive reports, and may cause a significant amount of traffic to the site (Example 10). Another subcategory rule disallows interaction with other users' accounts, in order to reduce the risk potentially caused by vulnerability research (Example 11). Third, other dangerous activities, such as social engineering (Example 12) and physical access to a data center (Example 13), are also prohibited.

Disregarding rules in this section may disqualify the white hats from receiving a bounty reward or participating in the program in the future. Violations could also cause white hats to face legal actions against them or exclusion from the entire platform.

4.7 Legal Clauses

Some organizations explicitly specify details of legal issues related to bug bounty. Statements of one subcategory promise not to bring legal actions against white hats, if rules are followed (Example 14). Another subcategory is to remind the white hats to comply with all relevant laws and regulations (Example 15). Another type of statements withhold the right to modify the rules at anytime (Example 16). We will analyze the occurrence of legal statements in Sect. 5.1.

4.8 Participation Restrictions

Although bug-bounty platforms are known for their openness to welcome white hats from around the world, some organizations explicitly exclude certain types of individuals from participating. Some organizations disallow their employees to participate, possibly in fear of misaligned incentives (Example 17). Organizations might also restrict participation based on white hats' nationality (Example 18). Some programs include explicit age restrictions (Example 19).[2]

4.9 Submission Guidelines

In this category, the organizations may describe what kind of details about discovered vulnerabilities they wish to have included in reports submitted by the white hats. Some organizations are very particular about report standards and they expect reports in a specific format with sufficient details like screenshots, pages visited, etc. (Example 20).

[2] HackerOne's Privacy Policy (https://www.hackerone.com/privacy/) states as a general policy that "we welcome minors to submit reports to HackerOne." However, the site is not directed at minors below 13 who would need to have their parents/guardians submit vulnerability reports and to set up an account.

4.10 Disclosure Guidelines

Organizations may also state whether they allow white hats to engage in public disclosure of the identified problems, or they may ask white hats to give them enough time to triage and fix the issue before public disclosure. We will discuss this aspect in Sect. 5.1 (Example 21). In addition, a bug bounty platform may have specific rules that apply to all programs concerning disclosure. HackerOne, for example, listed a process in its Vulnerability Disclosure Guidelines.[3]

4.11 Reward Evaluation

Rules in this category specify the concrete point system or evaluation process that the organization uses to determine whether white hats' submissions are eligible for rewards or appreciation. Some organizations list detailed reward evaluation criteria for different types of vulnerabilities. For example, Twitter provides a table in its rules statement which matches reward amounts to specific types of vulnerabilities, areas of the site, and various other conditions. Other organizations may simply specify a minimum reward (Example 22).

Another rule in this category is a "Duplicate Report Clause," which states whether only the first submission of a particular vulnerability is eligible for a reward, or whether later submissions may also receive (partial) rewards (Example 23). We will analyze the occurrence of such statements in Sect. 5.1. Further, organizations often state that they have the final decision authority whether a reward will be given (Example 24).

In addition, rewards are not restricted to monetary bounties, but could also represent other forms of appreciation, such as hacker points or swags (Example 25). As previously stated, about 50% of the organizations do not pay any monetary rewards.

4.12 Company Statements

In our efforts to iteratively classify rules, this last category contains statements which are more of a description rather than clear instructions or other reward-relevant information. A key objective of this category appears to be demonstrating an organization's willingness to improve security, and to collaborate with the white hat community (Example 26).

5 Quantitative Analysis

In this section, we provide an exploratory quantitative analysis of our rule dataset. Note that for measuring the rates of bugs resolved and hackers thanked,

[3] HackerOne's Vulnerability Disclosure Guidelines (https://www.hackerone.com/disclosure-guidelines/).

we restrict our analysis to the 77 programs for which detailed history was available. This restriction ensures that we compute the rates for each program in a time interval when the rules of the program did not change significantly.

We first study whether more detailed rule descriptions lead to greater success (Sect. 5.1). To answer this question, we study the relationships between description length and the number of bugs resolved and hackers thanked by a program. We show that programs with longer descriptions generally tend to be more successful, which suggests that detailed program descriptions are an important factor in success. Then, we investigate the readability of program rules using established metrics from the field of readability studies (Sect. 5.1), and show that the readability of program descriptions could be significantly improved in practice. Next, we study three important clauses that program rules may include: duplicate reports, legal actions and public disclosure (Sect. 5.1). We show that the presence or absence of these clauses can have a very strong impact on the success of a program, which implies that organizations need to include them in their rules if they wish to be successful. We also study statements for staging sites, test accounts, and source code availability in the program description. Finally, we perform a detailed regression analysis (Sect. 5.2), and study the combined effects of description length, various clauses, etc.

5.1 Descriptive Analysis

Length of Bug Bounty Rules. The average length of program rules is 481 words ($N = 77$). The shortest description is 72 words (Vulners) and the longest one is provided by ownCloud with 1,744 words. To provide an initial overview of the data, we categorize the organizations into four groups based on program description length (measured as number of words). These groups contain programs with word counts of (1) 0–250 words ($N = 13$), (2) 250–500 words ($N = 36$), (3) 50–750 words ($N = 16$), and (4) 750+ words ($N = 12$). Figure 1 shows the average rates of bugs resolved and hackers thanked for each group. Note that the rates were computed for each program over a time interval in which the description was unchanged, dividing the number bugs resolved and hackers thanked by the length of the interval in years. Tables 2, 3, and 4 in Appendix B list the descriptive statistics of word count, bugs resolved, hackers thanked and bounty paid for all organizations (Table 2), organizations paying minimum bounty (Table 3) and organizations paying no minimum bounty (Table 4), respectively.

It is noteworthy that the *length of program rules is positively associated with the average rate of bug resolved as well as the average rate of hackers thanked* (see Fig. 1 and Table 2). These observations also hold for all organizations paying a minimum bounty ($N = 44$, see Table 3). For organizations that are not paying a minimum bounty ($N = 33$), these relationships hold *very consistently* (see Table 4). However, there is no obvious trend observable for the relationship between the length of program rules and the average bounty paid by programs for valid discoveries (see Tables 2 and 3).

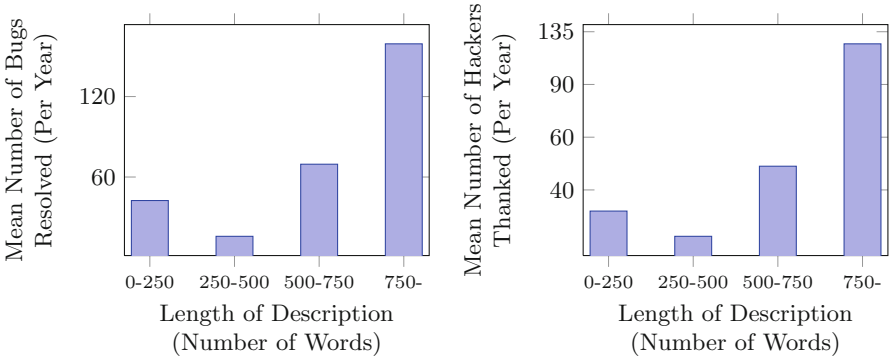


Fig. 1. Statistics for programs grouped by description length (measured as number of words). Please note the logarithmic scaling of the vertical axes.

Readability of Program Rules. We computed various metrics for the readability of program rules. For brevity, we report the results only for the Flesch Reading-Ease Score [10] here (see Fig. 2), which is an established metric in the field of readability studies. Results for other metrics are shown in Fig. 6 in Appendix D, but they do not differ in a meaningful way regarding the following basic observations.

The higher the score, the easier a document is to read. Scores towards 100 indicate that a minor in 5th grade would likely understand the document without problems. A score of 30 and below typically requires a college degree. Law review articles and technical documents frequently score in the 30s.

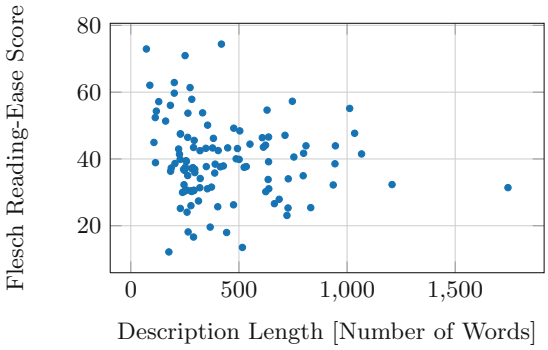


Fig. 2. Length and readability of program descriptions.

We find that the average level of the Flesch Reading-Ease Score for the sample of program rules is 39.6, indicating a set of documents requiring some college education (on average). The least readable document scored 12.2, whereas the

most readable document had a score of 74.4. There are 18 program rules that score below 30, which indicates documents that are very difficult to read.

While our analysis does not yet account for the specific characteristics of program rule documents (e.g., technical terminologies, tables etc.), it is indicative that *improvements could be undertaken to make these documents more approachable*. Perhaps in contrast to the going practice for many forms of legal agreements, program rules should be written with the intention of being read and understood by white hats who search for vulnerabilities in a particular program. A lack of readability may be a contributing factor to inefficient outcomes, and might encourage white hats to prefer other programs.

Statements About Duplicate Reports, Legal Actions, and Public Disclosure. In the taxonomy section, we already provided an initial overview of the various rules included in the program description. In the following, we study three key rules numerically.

First, we recorded whether the program rules include a duplicate report clause, i.e., whether the organization explicitly specifies if submitting a duplicate report will affect the white hat's eligibility for a bounty or not. Please note that duplicates occur very frequently in practice, and may pose a significant problem. For Google's bug bounty program and on the BugCrowd platform, the number of duplicates is higher than the number of valid reports. The ratio of duplicates is lower on HackerOne, but still substantial. It is therefore likely that white hats prefer to work with programs that are aware of this challenge and discuss it in their program rules.

Second, we identified programs that have some form of a legal action clause. Using a legal action clause, an organization informs white hats under what conditions it may (or may not) bring a lawsuit against them. Due to several highly-publicized incidents, where companies sued white hat hackers, or prevented them from speaking at conferences or other events, we believe that such statements can influence a white hat's decision to work for a specific program.

Third, we investigated which programs include a specific statement regarding public disclosure. Organizations may be particularly concerned about the internal security of their systems and applications, hence they may prohibit white hats from disclosing any identified vulnerabilities to other entities for a specified time period or until the bug has been fixed. HackerOne's Vulnerability Disclosure Guidelines allow white hats to publicly disclose information about bugs 180 days after they have submitted the report. Hence, organizations who take the extra step to alter their program policy may have specific concerns, and the presence of such a clause may also influence white hat behavior.

In general, we believe that specifying these three policies is indicative of a better developed program by the organization. To verify this, we investigated the 111 programs and found that 51 organization mention at least one of the three clauses in their programs. For these 51 organizations, we further show their status in Fig. 3. Only 10 out of these 51 organizations have all three clauses.

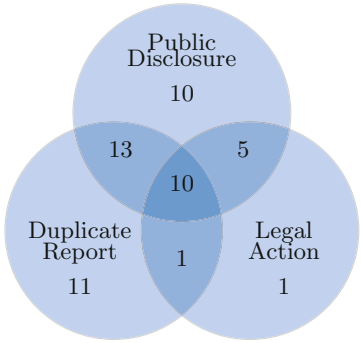


Fig. 3. Venn diagram explaining extensibility of rules.

In Fig. 4 (and Table 5 in Appendix C), we provide descriptive statistics of how the presence or absence of these three rule clauses are related to the rate of hackers thanked and rate of bugs resolved. We observe that the presence of these rules is associated with more active programs. Both the rate of hackers thanked and the rate of bugs resolved are *significantly higher on average for programs that include these statements* in their program rules.

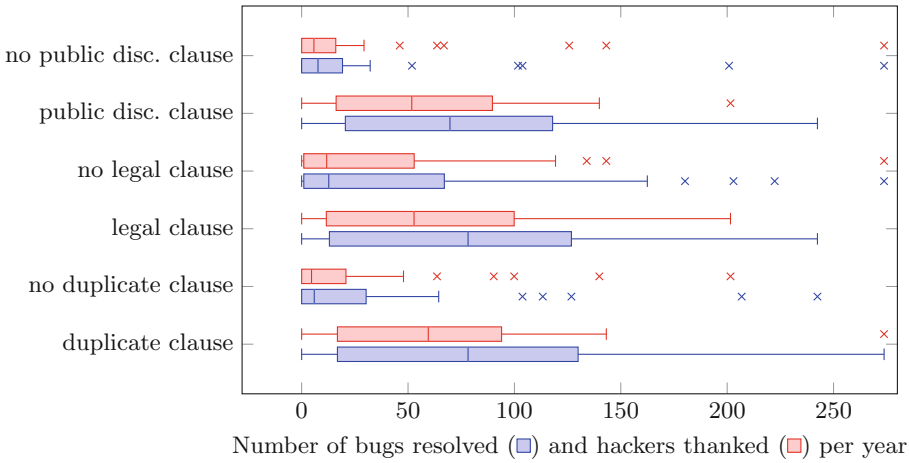


Fig. 4. Statistics for duplicate report, legal action, and public disclosure clauses.

Statements About Staging Sites, Test Accounts, and Source Code Availability. Other important indicators of sophistication are whether the organization provides white hats with a staging site for identifying vulnerabilities, whether it asks them to create designated testing accounts, and whether it allows them to download a copy of the application/software for testing.

Our classification shows that 5 out of 111 organizations have staging sites, 24 ask white hats to use a test account, and 13 provide source code of the application/software. When we investigate whether the availability of a staging site or source code impacts the rate of approved vulnerability reports and hackers thanked, we do not observe a very strong pattern (see Fig. 5 below and Table 6 in Appendix C). However, the requirement to use test accounts appears to have positive impact.

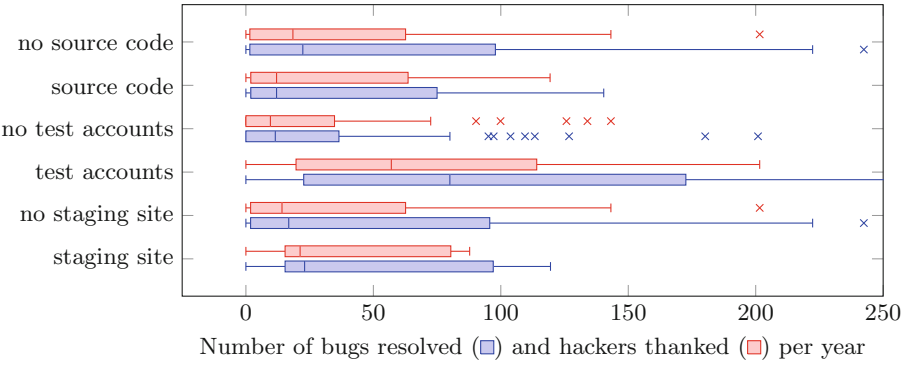


Fig. 5. Statistics for statements of staging sites, test accounts, and source code.

5.2 Regression Analysis

For the findings stated above, further quantitative analysis is needed to substantiate the observed effects. Particularly, we want to study the combined effects of rule features, including the length of the rule (L), the Flesch Reading-Ease Score that measures the readability of the description (R), the existence of legal action, duplicate report, public disclosure, staging site, test account, and source code download statements (LE, DU, DI, ST, TA, SC) on the success of a program, which is measured by the number of bugs resolved (V). Therefore, we build the following least square regression model:

$$V = \beta_0 + \beta_1 L + \beta_2 R + \beta_3 LE + \beta_4 DU + \beta_5 DI + \beta_6 ST + \beta_7 TA + \beta_8 SC + \boldsymbol{\beta_9 Z} + \epsilon. \quad (1)$$

In the regression model, we have considered other characteristics of bug bounty programs that could affect both the success of a program and the textual features, in order to mitigate the correlated omitted variable bias. More specifically, we add three control variables (represented as a vector $\boldsymbol{Z}$ in the regression model), based on previous work [15,25]: B is the average bounty paid by the program,[4] T is the age of the bug bounty program, and A is the log of the

[4] Since not all programs disclose their average bounty, we have to restrict our analysis to 58 data points in this subsection.

Alexa rank of the organization's website.[5] Alexa rank proxies for the complexity of the website, and a more complex website is likely to have both longer rule descriptions and inherently more vulnerabilities to find.

Table 1. Regression results

Variables	(1) V	(2) V	(3) V	(4) V
Length of the rule (L)	0.18***	0.09*	0.09*	0.01
	(0.04)	(0.04)	(0.04)	(0.05)
Average bounty (B)		0.12*	0.12*	0.09*
		(0.05)	(0.05)	(0.05)
Age of the program (T)		0.05	0.05	0.13***
		(0.04)	(0.04)	(0.05)
Log(Alexa rank) (A)		−4.65	−4.30	−4.20
		(2.86)	(2.98)	(2.88)
Readability score (R)			−0.51	
			(0.79)	
Has legal clause (LE)				23.04
				(27.41)
Has duplicate report clause (DU)				47.39*
				(22.08)
Has public disclosure clause (DI)				60.41**
				(24.45)
Has staging site (ST)				1.10
				(40.70)
Has source code (SC)				45.56*
				(27.03)
Asks to use test accounts (TA)				1.01
				(26.78)
Constant	−15.21	23.21	34.25	−14.40
	(18.95)	(39.10)	(45.93)	(39.34)
Observations	54	54	54	54
R-squared	0.27	0.43	0.44	0.57

Robust standard errors in parentheses *** p<0.01, ** p<0.05, * p<0.1

[5] A lower value of Alexa rank represents a more popular website. For example, an Alexa rank of 1 indicates the most-visited website.

The results of our regression analysis can be found in Table 1[6]. We incrementally add the factors to the regression model, beginning with a simple model explaining the number of discovered vulnerabilities through the varying length of program rules. Other more complex models follow.

Our first observation is that the length of the rule description is positively correlated with the number of vulnerabilities discovered. The correlation is significant in three out of the four models. Several hypotheses could explain this positive correlation. First, a long rule may indicate that the organization spends more effort on improving the engagement with white hats (e.g., by giving more guidance on what to look for), which in turn makes white hats more productive. Second, a long rule could also be associated with larger scope, leading to more opportunities for finding bugs.

We do not observe a significant correlation between the readability score and the number of bugs resolved. We see positive and statistically significant coefficients for the existence of duplication clause, disclosure statements, and source code.

In summary, the results indicate that rules with more content (e.g., more detailed list of included/excluded areas and issues) and explicit statements on duplication, disclosure, etc., are associated with more bugs resolved. This suggests that rules could have indeed a tangible impact on bug bounty program performance, and organizations should spend more effort to maintain and improve their rules. On the other hand, we are also aware of limitations of our regression analysis. Particularly, the sample size is rather small, and the dataset has some additional limitations, as we have discussed in Sect. 3. As such, a potential future work item is to include more data points into our regression. One could also consider other bug bounty platforms, such as BugCrowd and Cobalt, to conduct a cross-platform study.

6 Discussion

The emergence of bug bounty platforms allows many different organizations, such as Yahoo!, General Motors, and even the U.S. Department of Defense, to harvest the power of the global white hat hacker community for improving security. However, as previous research shows (e.g., [13]), effectively and efficiently engaging white hats is a challenging task. In addition, there are risks for both sides. For organizations, there are security risks associated with vulnerability research and disclosure. For white hats, there are legal risks to worry about as well as a potential lack of adequate appreciation of their findings. The program rules serve as a key method to control the risk and facilitate engagement.

Currently, program rules are primarily created by participating organizations independently. Therefore, they vary by content, length, style and many other factors. The bug bounty rule taxonomy we assembled in Sect. 4 is a first step

[6] Note that we use data from the entire history of each bug bounty program. We have also tested the models using only data available after the last major rule update of each program. The regression analysis shows the same directionality of effects, but the dataset is much smaller to report a robust analysis.

toward organizing and studying these widely different bug bounty rules. Based on HackerOne's public bug bounty programs, we created 12 categories of rule statements. In the future, this taxonomy can be referenced by organizations when they create or update their own bug bounty rules. Further, the taxonomy also provides a basis for academia to further analyze program rules on different bug bounty platforms.

Our research mainly focuses on the rules of individual bug bounty programs. However, it is also possible that a platform-wide rule influences hacker engagement. We also examined the platform rule created by HackerOne, and determined that it only provides some high-level guidance, and that it primarily refers to individual program's rules for critical issues. In addition, we find a potential issue with the following statement in the platform rule: *"Security Teams will publish a program policy designed to guide security research into a particular service or product. You should always carefully review this policy prior to submission as they will supersede these guidelines in the event of a conflict."* It is surprising that the guidelines do not state that white hats should read the policy *before* doing vulnerability research, as the investigative process can bring harm to an organization's system if not properly conducted. We suggest that platforms shall work more closely with individual programs to make the platform rules consistent with the diverse program rules. Also, we suggest platforms to create more comprehensive rules for cases not covered by individual rules.

7 Conclusion

As bug bounty platforms gain in perceived sophistication and impact, the participation of organizations and white hats will likely continue to increase. As such, it will become increasingly important to appropriately manage the rules which govern the interactions between the different stakeholders.

Our analyses demonstrate that bug bounty programs are on average associated with better success characteristics if the level of comprehensiveness of their rules of engagement increases. We demonstrate this finding for a high level metric (i.e., program length) as well as detailed characteristics such as the presence of legal action clauses, rules for duplicate submissions and rules for public disclosure. These observations are novel to the research literature on bug bounty platforms. We anticipate that our analysis will be motivation to bug bounty programs and platforms to pay greater attention to the detailed rules in order to provide a fair and more effective workplace for white hat researchers.

Further work is desirable to solidify and extend our findings. In particular, we plan an additional iterative analysis of the program rules to extract more performance-relevant criteria and to embed them in the statistical analysis. Further, the scope of the analysis could be broadened to include more bug bounty platforms in order to add robustness to the findings.

Acknowledgment. We thank the anonymous reviewers for their comments. The research activities of Jens Grossklags are supported by the German Institute for Trust and Safety on the Internet (DIVSI).

A Example Bug Bounty Rule Statements

We list example rule statements below:

1. *"Please report serious vulnerabilities in our website* (https://staging.factlink.com), *proxy* (https://staging.fct.li), *or other components (our annotation library, Wordpress plugin, browser extensions, and gems)"* (Factlink)
2. *"Please note that Binary.com's front-end code is open-sourced at [...] - please feel free to report any vulnerabilities found in this code by submitting a pull-request in github"* (Binary)
3. *"Not in scope: shopify.asia, go.shopify.com and investors.shopify. com are operated by third parties, and are not in scope"* (Shopify)
4. *"Any Sucuri customer website are out of the scope of this disclosure program"* (Sucuri)
5. *"Mattel websites and services are owned and operated by Mattel and are explicitly outside the scope of this bug bounty program"* (ToyTalk)
6. *"Please only use our staging environments for testing, they are otherwise identical to production"* (Factlink)
7. *"In general, anything which has the potential for financial loss or data breach is of sufficient severity, including: XSS, CSRF, Authentication bypass or privilege escalation, Click jacking, Remote code execution, Obtaining user information, Accounting errors"* (Coinbase)
8. *"We are generally not interested in DoS vulnerabilities that are perceived by a lack of rate-limiting or captcha. As a web-scale service, our threshold for rate limiting is higher than you would probably expect. Of course, if you think you have found an exception to this rule, please let us know"* (Automattic)
9. *"Please create a free account and (pen) test away our GhostMail, ChostChat and GhostBox"* (Flox)
10. *"Please note that automated testing is not permitted! System will ban you permanently if you do"* (DigitalSellz); *"If you employ automated scanning tools, their requests must be rate limited to not exceed 3 requests per second without prior approval"* (Vimeo)
11. *"Do not attempt to gain access to another user's account or confidential information"* (Adobe)
12. *"You are not allowed to conduct social engineering attacks against our support team"* (Coinbase)
13. *"While researching, we'd like to ask you to refrain from: [...] Any physical attempts against BitHunt property or data centers"* (BitHunt)
14. *"In order to encourage responsible disclosure, we promise not to bring legal action against researchers who point out a problem provided they do their best to follow the above"* (Openfolio)
15. *"You must comply with all applicable laws in connection with your participation in this program. You are also responsible for any applicable taxes associated with any reward you receive"* (Twitter)
16. *"Yahoo reserves the right to change or modify the terms of this program at any time"* (Yahoo)

17. *"Yahoo employees and contingent workers, as well as their immediate family members and persons living in the same household, are not eligible to receive bounties or rewards of any kind under the Yahoo Bug Bounty Program, whether hosted by Yahoo or any third party"* (Yahoo)

18. *"You must be eligible to work within the U.S.; meaning you are a U.S. citizen, a noncitizen national of the U.S., a lawful permanent resident, or an alien authorized to work within the U.S."* (Hack the Army)

19. *"You must be 18 years of age or older. Please be an adult when messaging us. We want to work with serious security professionals only"* (Envoy)

20. *"Share with us the full details of any problem found. Detailed steps on reproducing the bug. If valuable, please include any screen-shots, links you clicked on, pages visited, etc. Provide us with a concrete attack scenario. How will the problem impact Bookfresh or our customers? Put the problem into context"* (BookFresh)

21. *"Provide us a reasonable amount of time to resolve the issue before any disclosure to the public or a third-party"* (ownCloud)

22. *"Minimum reward is $100 for security vulnerabilities. The reward depends on the vulnerability severity and will be paid via HackerOne only. Every researcher with accepted vulnerability will be mentioned on* http://hackerone.com/algolia/thanks*"* (Algolia)

23. *"We only reward the first reporter of a vulnerability"* (DropBox)

24. *"Twitter will determine in its discretion whether a reward should be granted and the amount of the reward"* (Twitter)

25. *"Post on our Hall of Fame. Your very own Informatica Bug Bounty T-Shirt With More Awesome Swag to Come"* (Informatica)

26. *"Security and privacy are top priorities at Coursera. We believe that no technology is perfect and that working with skilled security researchers across the globe is crucial in identifying weaknesses in our technology"* (Coursera)

B Statistics for Programs Grouped by Description Length

Table 2. Statistics for programs grouped by description length

Number of words	Number of programs	Mean number of words	Mean number of bugs resolved (per year)	Mean number of hackers thanked (per year)	Mean bounty paid
0–250	13	200	49	34	225
250–500	36	340	36	28	392
500–750	16	633	67	48	59
750–	12	1011	189	123	105
Overall	77	481	69	48	250

Table 3. Statistics for programs paying a minimum bounty (grouped by description length)

Number of words	Number of programs	Mean number of words	Mean number of bugs resolved (per year)	Mean number of hackers thanked (per year)	Mean bounty paid
0–250	8	211	74	49	366
250–500	15	318	13	10	941
500–750	12	630	68	49	79
750–	9	1042	194	143	140
Overall	44	532	76	55	437

Table 4. Statistics for programs paying no minimum bounty (grouped by description length)

Number of words	Number of programs	Mean number of words	Mean number of bugs resolved (per year)	Mean number of hackers thanked (per year)	Mean bounty paid
0–250	5	182	10	8	0
250–500	21	355	53	41	0
500–750	4	641	66	46	0
750–	3	918	176	65	0
Overall	33	415	59	39	0

C Statistics for Programs Grouped by Clauses

Table 5. Statistics for duplicate report, legal action, and public disclosure clauses

Clause	Present	Number of programs	Mean number of bugs resolved (per year)	Mean number of hackers thanked (per year)
Duplicate	yes	35	117	79
	No	42	29	22
Legal action	yes	17	134	95
	No	60	50	35
Public disclosure	yes	38	102	73
	No	39	36	24

Table 6. Statistics for staging sites, test accounts, and downloading source

Clause	Present	Number of programs	Mean Number of bugs resolved (per year)	Mean number of hackers thanked (per year)
Staging site	yes	5	51	41
	No	72	70	49
Test account	yes	24	133	97
	No	53	40	26
Source code	yes	13	40	34
	No	64	75	51

D Further Readability Analysis

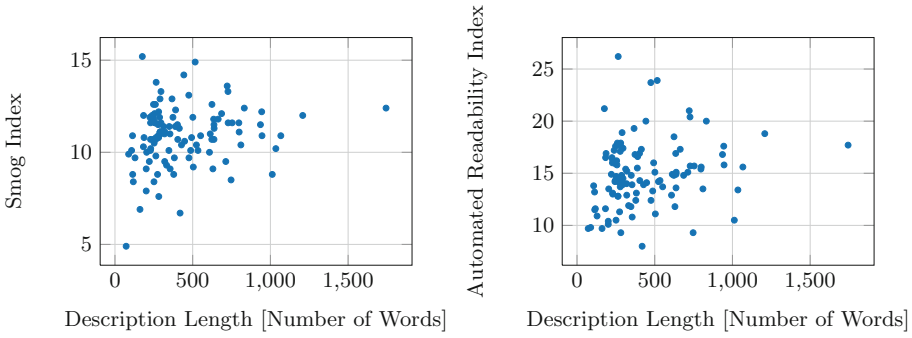


Fig. 6. Program descriptions length and readability, measured using Smog Index [16] and Automated Readability Index [22].

References

1. Algarni, A., Malaiya, Y.: Software vulnerability markets: discoverers and buyers. Int. J. Comput. Inf. Sci. Eng. **8**(3), 71–81 (2014)
2. Bacon, D., Chen, Y., Parkes, D., Rao, M.: A market-based approach to software evolution. In: 24th ACM SIGPLAN Conference Companion on Object Oriented Programming, Systems, Languages, and Applications (2009)
3. Böhme, R.: A comparison of market approaches to software vulnerability disclosure. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 298–311. Springer, Heidelberg (2006). https://doi.org/10.1007/11766155_21

4. Bozorgi, M., Saul, L., Savage, S., Voelker, G.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 105–114 (2010)
5. Bugcrowd: The state of Bug Bounty, July 2015
6. Bugcrowd: The state of Bug Bounty, June 2016
7. Clark, S., Frei, S., Blaze, M., Smith, J.: Familiarity breeds contempt: the honeymoon effect and the role of legacy code in zero-day vulnerabilities. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC), pp. 251–260 (2010)
8. Edmundson, A., Holtkamp, B., Rivera, E., Finifter, M., Mettler, A., Wagner, D.: An empirical study on the effectiveness of security code review. In: Jürjens, J., Livshits, B., Scandariato, R. (eds.) ESSoS 2013. LNCS, vol. 7781, pp. 197–212. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36563-8_14
9. Finifter, M., Akhawe, D., Wagner, D.: An empirical study of vulnerability rewards programs. In: USENIX Security Symposium (2013)
10. Flesch, R.: A new readability yardstick. J. Appl. Psychol. **1948**(32), 221–233 (1948)
11. Huang, K., Siegel, M., Madnick, S., Li, X., Feng, Z.: Poster: diversity or concentration? Hackers' strategy for working across multiple bug bounty programs. In: 37th IEEE Symposium on Security and Privacy (S&P) (2016)
12. Kuehn, A., Mueller, M.: Analyzing bug bounty programs: an institutional perspective on the economics of software vulnerabilities. In: TPRC Conference Paper (2014)
13. Laszka, A., Zhao, M., Grossklags, J.: Banishing misaligned incentives for validating reports in bug-bounty platforms. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 161–178. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_9
14. Laszka, A., Zhao, M., Grossklags, J.: Devising effective economic policies for bug-bounty platforms and security vulnerability discovery. J. Inf. Policy **7**, 372–418 (2017)
15. Maillart, T., Zhao, M., Grossklags, J., Chuang, J.: Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty markets. J. Cybersecur. **3**, 81–90 (2017)
16. Mc Laughlin, H.: SMOG grading - a new readability formula. J. Reading **12**(8), 639–646 (1969)
17. Ozment, A.: The likelihood of vulnerability rediscovery and the social utility of vulnerability hunting. In: Workshop on the Economics of Information Security (WEIS) (2005)
18. Ozment, A., Schechter, S.: Milk or wine: does software security improve with age? In: USENIX Security Symposium (2006)
19. Ransbotham, S., Mitra, S., Ramsey, J.: Are markets for vulnerabilities effective? MIS Q. **36**(1), 43–64 (2012)
20. Ratcliff, J., Metzener, D.: Pattern-matching: the gestalt approach. Dr Dobbs J. **13**(7), 46 (1988)
21. Rescorla, E.: Is finding security holes a good idea? IEEE Secur. Priv. **3**(1), 14–19 (2005)
22. Senter, R., Smith, E.: Automated readability index. Technical report, DTIC document (1967)
23. Shahzad, M., Shafiq, M., Liu, A.: A large scale exploratory analysis of software vulnerability life cycles. In: International Conference on Software Engineering (2012)

24. Zhao, M., Grossklags, J., Chen, K.: An exploratory study of white hat behaviors in a web vulnerability disclosure program. In: 2014 ACM CCS Workshop on Security Information Workers (2014)
25. Zhao, M., Grossklags, J., Liu, P.: An empirical study of web vulnerability discovery ecosystems. In: 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS) (2015)
26. Zhao, M., Laszka, A., Maillart, T., Grossklags, J.: Crowdsourced security vulnerability discovery: modeling and organizing bug-bounty programs. In: HCOMP Workshop on Mathematical Foundations of Human Computation (2016)

Why Johnny Doesn't Use Two Factor A Two-Phase Usability Study of the FIDO U2F Security Key

Sanchari Das(✉), Andrew Dingman, and L. Jean Camp

Indiana University Bloomington, Bloomington, USA
{sancdas,adingman,ljcamp}@indiana.edu

Abstract. Why do individuals choose to use (or not use) Two Factor Authentication (2FA)? We sought to answer this by implementing a two-phase study of the Yubico Security Key. We analyzed acceptability and usability of the Yubico Security Key, a 2FA hardware token implementing Fast Identity Online (FIDO). This token has notable usability attributes: tactile interaction, convenient form factor, physical resilience, and ease of use. Despite the Yubico Security Key being among best in class for usability among hardware tokens, participants in a think-aloud protocol still encountered several difficulties in usage. Based on these findings, we proposed certain design changes, some of which were adopted by Yubico. We repeated the experiment, showing that these recommendations enhanced ease of use but not necessarily acceptability. With the primary halt points mitigated, we could identify the remaining principle reasons for rejecting 2FA, like fear of losing the device and perceptions that there is no individual risk of account takeover. Our results illustrate both the importance and limits of usability on acceptability, adoption, and adherence in Two-Factor Authentication.

Keywords: Two-Factor Authentication
Hardware authentication device · Usable security · Adaptability

1 Introduction

The Yubico Security Key is an implementation of Fast Identity Online (FIDO) [21] Universal Second Factor (U2F) in a USB token form. The Security Key is designed to appeal to high-touch, low-tech users who want more secure interactions and improved ease of use from their online service providers [13]. According to Brett McDowell, Executive Director of the FIDO Alliance, "We fail if FIDO is not more usable than all the other (hardware token) options you have used before" [14].

We explore the acceptability and usability of the FIDO U2F technology, in the form of the Yubico Security Key, against these goals and metrics using a think-aloud protocol. We recruited students from STEM degree programs and tested different setup instruction sets. Our goal was to identify difficulties that

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 160–179, 2018.
https://doi.org/10.1007/978-3-662-58387-6_9

might be barriers to adoption. Usability measures if individuals can complete a set of tasks with a given technology. Acceptability addresses the experience of use, including perceived risks and benefits, and impinges on user adoption.

We asked experiment participants to configure a FIDO U2F Security Key for their Gmail account. From the analysis of the participants' experiences, we developed a series of recommendations for Yubico. Yubico adopted and implemented a subset of the recommendations. A year later, after the changes were made, we repeated the experiment to evaluate the new interaction with the security keys. There was a significant increase in usability, but we could not assert any corresponding increase in acceptability.

We detail the related literary work in Sect. 2, our experiment methods in Sect. 3, and results in Sect. 5. We further discuss on future recommendations in Sect. 7 both for Yubico Security Keys in specific and Two-Factor Authentication at large. We conclude by providing an overview of the study and giving a future direction towards a better usability, acceptability, and adaptability of the FIDO security keys in Sect. 8.

2 Related Work

Our work was primarily informed by research on usable authentication as well as influenced by research on online risk perception and risk communication. Bonneau et al. provided a framework for examining the usability of authentication technologies [4]. Before authoring this framework with Bonneau, Stajano provided a set of recommendations for any authentication system through research grounded in the Pico hardware token authentication project [22]. This earlier work demonstrated five core requirements: security, memory-less operation, scalability, loss resistance, and theft resistance.

Previous work has shown that FIDO Security Keys are easy to deploy. Lang et al. refer to the use of a Security Key as "brainless", which seems to indicate a belief that there are no halt points in Security Key adoption [13]. However, this study included neither qualitative components nor human subject experiments. In our work, we have implemented a think-aloud protocol and found numerous halt points and challenges to acceptability. A previous human-centered evaluation of 2FA also found that users perceived twice the utility from *avoiding* 2FA compared to adopting it [6]. Our results echoed this finding, with most subjects simply leaving or returning the keys. Usability of 2FA methods has been studied by Krol et al. [12], however they studied online banking customers and people often relate financial accounts as more confidential than their email accounts. The study by Krol et al. also focused on 2FA in general and approached the usability of 2FA from the steps a user has to follow in contrast to how we studied the Yubico Security Key, it's usability and acceptability.

Passwords have been heavily critiqued in academic research. Archaic recommendations such as formulaic complexity requirements of passwords and periodic password changes may be helpful, but still cannot ensure protection against password vulnerabilities. Instead, best practice guidelines are proposed such as validating newly created passwords against commonly-used or known compromised

passwords [17]. In *Understanding Password Choices*, Wash et al. showed that users tend to re-use passwords across sites, especially where they must enter passwords frequently [24]. Komanduri et al. found that users frequently have critical misunderstandings about what make passwords secure - with a tendency to overestimate the effects of additional complexity, while underestimating the impact of using common phrases [11].

Unusable password policies often result in insecure workarounds, but Inglesant and Sasse assess that the cost goes beyond insecurity and often negatively impacts the productivity of both individuals and their organizations. Their work indicated that human-centered design principles should influence policy creation which guides users to create suitably secure passwords in accordance with the usage context [9]. Through the password system, Abbott et al. showed that users can indeed be guided towards better password decisions without corresponding increase in cost [5]. Biddle et al. shows that though it was more acceptable to the users, it gradually resulted in decreased predictability of user password behavior [3]. The major alternative to FIDO in 2FA is time-based (TOTP) or hash-based (HOTP) one-time codes. However, neither TOTP or HOTP offers mutual authentication of both the user and the service [15,16]. Our study was informed by a two-phase examination of Tor by Norcie et al. [18]. Norcie et al.'s study followed the same process of a think-aloud protocol implemented in our study. Their study also analyzed the design modifications made to Tor and was in turn strongly influenced by the canonical *Why Johnny Can't Encrypt*, which examined the use of Pretty Good Privacy(PGP) [26] by asking participants to complete the tasks required for adoption and use. By observing the difficulties encountered by participants, Norcie and Whitten offered design heuristics for anonymized systems and public key systems for emails respectively. Some of the recommendations, such as focusing on the importance of the setup steps prior to operation or conveying to the user why a feature exists may be generalized to authentication systems. Our target was to provide specific solutions to enhance the adaptability of the security keys.

3 Methodology

We investigated the end user experience of configuring and using the FIDO Security Key by combining a think-aloud protocol and two surveys before and after the experiment. In a think-aloud protocol the subjects narrate their actions, providing a real-time description of their decisions, choices, or motivations. The preliminary survey was online, followed by the think-aloud protocol being implemented in a university computer laboratory. There were open questions asked after the think-aloud protocol and a final survey sent via email. We then implemented the first phase of the study and made recommendations to Yubico and Google, some of which were adopted. This experiment was repeated after a year.

Participants were recruited from an undergraduate non-technical introductory security course. We recruited the participants from the same course in the consecutive year. Participants completed a preliminary background, knowledge, and skills survey to evaluate any differences in the security and computing

expertise of the subject pool. These technical knowledge and skill inventories were implemented and calculated as done by Rajivan et al. [19]. The participants were required to be (i) at least 18 years old, (ii) have a personal Gmail account, (iii) own a laptop with the Chrome browser, and (iv) own a personal mobile phone.

The participants were consciously selected to have more security and computer expertise than the general population. Similar to the experiments on setting up access control rules [2], firewalls [20] and PGP [26], we chose a population likely to be successful.

A coin flip was used to randomly divide the participants into two groups. In one group, participants were directed to the official Yubico Security Key instructions. The other group was directed to the Security Key instructions provided by Google. The think-aloud protocol began by giving each participant a Yubico Security Key, as shown in Fig. 1. The participant was then asked to configure 2FA using the Yubico Security Key with their Gmail account while narrating the experience. Each participant was paired with one researcher who took notes, but did not offer additional guidance unless requested when the participant was unable to proceed without some guidance. After the task was complete, participants were asked to describe the benefits and importance of the Yubico Security Key using the seven open-ended questions below.

1. How could you test to confirm that your key is working?
2. If your key was lost or stolen, what would you do?
3. Based on your current understanding of the technology, could you use the same key with an account on another web site, or would you need to obtain an additional key?
4. Based on your current understanding, could you add a second key to your account?
5. Do you see any benefits from using the security key? Please explain.
6. Do you expect to continue to use your key after today? Why or why not?
7. How would you remove a key from your account if you decided to?

There were two goals for this closing interview. One was to explore the participant's reflection on the experience of configuring 2FA. The second was to ensure that we would not harm the participants by locking them out of their accounts. Each participant departed only after the researchers were certain that the research subjects were capable of removing 2FA without any assistance.

One month after the end of the think-aloud protocol, the subjects received a follow-up survey inquiring about their continued use of 2FA. The follow-up survey was sent over email.

Fig. 1. Yubico security key

3.1 Coding and Analysis

Recall that there was a preliminary survey, a think-aloud protocol, an interview after the protocol, and

a survey on continued use well after the experiment. In this section, we describe the coding and analysis of the qualitative data. The procedure was identical in both phases. We solved the few discrepancies between the codes which were discussed between the coders and the researchers.

There were two sources of qualitative data. The first source was the transcripts of the think-aloud protocol itself. These transcripts began after the Yubico Security Key was handed to the participant and ended after the enrollment into 2FA was complete. The second source consisted of the transcription of the responses to the open-ended questions from the interview immediately following the experiment.

For the enrollment task, the halt points were noted for each subject. The conversations around the halt points as well as the responses to the open-ended questions were transcribed. Three researchers trained independently in qualitative methods read through the transcripts. As standard in qualitative research, the themes were compiled into a code book. Of the recorded halt points, the cause was identified to be centered around 4 major mutually exclusive points: form factor, a setup demo, setup validation, and security valuation of the device. *Halt* points occurred when participants could not move forward alone. *Confusion* points occurred when the participants significantly slowed down due to confusion, or stopped but would have been able to continue with the registration procedure without assistance. We also noted expressions of value, where participants expressed ideas or opinions about the perceived utility of the technology, device, or installation process.

In both experiments, many participants recognized the potential value of the Security Key in theory, but not in practice for themselves. The details of the two phases are described in the following two sections followed by a discussion addressing both.

4 Experiment

4.1 Phase-I

In Phase-I, we discovered that the most significant halt point was the confusion resulting from a Yubico demonstration tool. Yubico had built a tool clearly illustrating how to register the 2FA token with a service. Participants went through the demo and believed that they had completed the installation process. No participant in the experimental group that was directed to the Yubico demo was able to realize that they needed to continue and complete the installation without researcher intervention.

Phase-I concluded with a set of recommendations about the instructions, visualization, device identification, and guidance provided to users. The details of the recommendations are described in Sect. 5.3. We repeated the experiment to test the efficacy of the adopted recommendations after Yubico implemented a subset of them. We also revisited the recommendations (in Sect. 7) from Phase-I that were not implemented to determine if those changes were still needed.

As reported in Sect. 3, our participants for both the phases were recruited from the same class to ensure that the sample was moderately security savvy. We had 27 young scholar participants in total - 6 were between 18 and 20, 16 were between 21 and 23, 4 were between 24 and 26, and 1 was over 30 years old. There were 20 male and 7 female students, a 74% to 26% split. Every student was enrolled in at least one information or computer science class, by definition. Unfortunately, due to the nature of the recorded data we lost data of 6 participants and the results in Tables 1 and 2 indicate reflect that from 21 participants.

Depending on the computer and security expertise questions answered by the participants before the in-lab experiment, the mean security expertise was 2.96 of 5 and the mean computing expertise was 4.34. We compared this with a general population survey of 593 participants where the results showed a mean security expertise (using the same calculation) of 1.7 and a mean computing expertise of 1.77 [10]. As a result, it is reasonable to assume that any halt points encountered by this population would also occur in a less technical and less educated population.

4.2 Phase-II

In Phase-II, as with Phase-I, the participants were students recruited from the same computer security course after a year. We had 34 young scholar participants in total - 1 was between 18 and 20, 29 were between 21 and 23, 2 were between 24 and 26, 1 was over 30 years old, and one chose not to answer. There were 26 male students, and 8 female students, a 76.4% to 23.6% split. Every student was enrolled in at least one information or computer science class, by definition. In Phase-II, the mean security expertise score was 2.95 of 5 and the mean computing score was 4.34. Again, it is reasonable to assert that our participants have more security and computing expertise than the general population. The differences in the mean values were not significant.

5 Results

5.1 Phase I Findings and Usability

In this section, we discuss about the various halt and confusion points where the participants found it difficult to register the Yubico Security Key.

Inserting the Device. We were able to identify several points of confusion related to device form factor. Primarily, users experienced confusion about the correct orientation of the key due to the slim design which allows it to enter the USB port both correctly or upside-down.

Finding Instructions. Once the device was successfully inserted and individuals were directed to setup the device for their account, they had trouble getting started. Over half of the users navigated to their browser settings or their

email settings first. The second time they encountered an instruction-centered challenge was when they had actually found the correct 'account settings' control panel. For successful setup, users were required to follow a non-linear path through the control panel, and at each page, a large array of options were offered. This presented many opportunities for confusion and abandonment of setup altogether for several participants.

Illumination. To activate the Security Key, either for enrollment or authentication, users must touch a capacitive button on the device. The button light would blink on insertion and at other seemingly unrelated points. Participants frequently displayed confusion over the timing of button press and the meaning of the blinking light.

Correctly Identifying the Device. Participants in the first condition found the Yubico landing page to be difficult to understand and navigate. Despite having the original packaging for the device, participants generally were not confident about which model of Yubico Security Key they were using. This was a halt point where device identification was required to receive setup instructions. The most commonly mentioned reason for choosing a particular device was color. No subjects mentioned using the images on the button to differentiate between Security Key models.

"Try Out This Key" Link. Once subjects had determined which model of key they were using, the next challenge was finding the correct setup instructions. Without exception, participants identified a link to a demo application as the most salient option for their goal of setting up their key with a Google account.

Demo Versus Reality. Many participants either believed they had completed the task after successfully authenticating to the demo, or repeated the enrollment and test cycle of the demo tool several times without progressing. After ten minutes of repeating the demonstration cycle, we considered subjects to have reached a halt point. As one participant noted, *"The web site is kinda confusing because I do not know what it wants me to do."*

Biometric Versus Touch. Many participants thought the circular touch sensor was a biometric authenticator that read their fingerprints. This has both positive and negative implications. On the positive side, this indicated awareness that interaction was necessary. It also implied, however, a higher benefit than the device actually provides, since, in reality, anyone can use it. If the token is lost, users who believe they have biometric enrollment would not realize that others could use it. One of the participant's mention: *"I guess it is more secure because they make you scan your fingerprint before you can log into your account, but to me it's a bit excessive."*

Confirmation of Operation. Participants were unable to confirm that the device was working after setup. When users were queried, "How could you test to confirm that your key is working?", a common response was the intuitive "Log out and back in". Unfortunately, since the default during setup is to trust the current computer, users never got to actually experience using their Yubico

Security Key outside the set-up process. For single-computer users, this experience could be left until weeks in the future - *"Why didn't you prompt me?! It said it would...maybe I'll just try again."*

5.2 Phase-I Acceptability

The primary drivers of acceptability were the lack of awareness of the risk and the resulting perception that there was no benefit. Here we recommend changes to increase this acceptability. Several of the participants in our experiments dropped the keys in a shared bin for leftover hardware, often used for mice or cables. This reiterates the importance of theft and loss resistance noted in related work [4,22]. Participants in the experiment did not have a clear understanding of the possible risk of account subversion. Similar lack of awareness and uncertainty of the risk of their choices has been found in privacy as well as security previously [1].

5.3 Phase-I Recommendations

In response to our results we made specific recommendations in a technical talk presented to Yubico and Google. Some of these recommendations were adopted, either as a result of our work or serendipity. Here we list our recommendations and, in the case of adoption, note the difference.

Finding Instructions. The other issue was that people had difficulty finding instructions. The current Yubico web page has vastly improved this, providing icons that link not just to the service but directly to the instructions for Security Key enrollment. We found that the service provider descriptions were easier to follow than the Yubico descriptions for each service provider. Our recommendation for Yubico to provide pointers rather than instructions for each service provider was acknowledged.

Demo Versus Reality. First time users were not easily able to identify which product they had, or which instructions they were to follow. The "Try out your YubiKey" demo was a source of much confusion. In every experiment condition where a user was directed to the Yubico instructions, they got stuck in a loop with the instructions and required guidance for the next step.

The demo does appear to serve the important goal of providing hypothetical demonstrations to prospective institutional customers. However, when this demo is included as part of the display to those who have already purchased the product, it consistently caused confusion. We recommended that this demo should not be accessible to the end user, as it was a consistent halt point. Though still accessible, the demo has been removed from the installation work flow. As a result this halt point went from confounding nearly every single subject in Phase-I to having negligible impact in Phase-II.

Correctly Identifying the Device. In our initial experiment, the instructions asked what Yubico product a user had, but provided little identification guidance. A user's best option was a product comparison table, the top of the

website. The table appeared to have been designed to assist in purchase decisions rather than configuration, with prominent price information and technical data.

A new interface offers more prominent pictures and descriptors which allows for easier identification of the device to be used. The title clearly shows the purpose, providing confidence to the subject participants that they had found the correct source for device identification.

Biometric Versus Touch. A significant change implemented in the Yubico setup instructions is the clear identification that the button is not a fingerprint reader.

Confirmation of Operation. During the experiment, participants found it challenging to confirm that a newly registered Security Key was in fact operating correctly. This confusion was caused by Google's default behavior of marking the browser as a "trusted" device. In this case, users are not required to use a second authentication factor when logging in, even when 2FA is enabled for the account.

The default browser trust defeated subjects' natural inclination to test the newly enrolled device by logging out of their account and logging back in, as there was no prompt to use the key. A subset of the experimental group did arrive at a solution, either using "incognito mode" or clearing cookies from their browser before logging in again. However, not all participants possessed the technical understanding of Google's authentication process necessary to arrive at such a solution.

5.4 Phase-I Acceptability Recommendations

Despite the fact that the chosen research pool was more educated and security aware than the general population, no participant in Phase-I decided to continue to use the token provided. We know that some Yubico security tokens were returned to researchers immediately or placed in the discarded available hardware bin after the experiment. We also piloted the study before deploying it to the participants in phase-I. In contrast, the participants in the pilot phase were all graduate students in security and all of the pilot participants continued to use the tokens.

Communicate the Intrinsic Benefit. Rational decision making theories fail to account for observed security and privacy choices, either individual or in the aggregate. Yet people consistently use a set of heuristics in making decisions, such that benefits obtained are greater than the risk. Garg argued that security systems should be designed to take advantage of these theories to encourage more adoption [7]. Applying the observations here, we recommend better risk communication that indicates that the use of the token is a benefit.

Developing appropriate feedback for users has long been recognized as a design challenge [26,27]. Thus, we recommend the addition of such feedback for users to be aware of the benefits of using the device. This may include confirmation of successful registration on first login, or occasional information about the superior security while content loads.

Communicating the Risks. Users did not understand the benefit of the device as compared to a longer, more secure password. Users who chose to return the token expressed confidence in their own security management and length of passwords. Many of the users thought the device was useful in the case of computer theft, but were dismayed to find that the device would remain trusted even when lost.

Create a Cognitive Benefit. A major impediment to users' perception of value was the continued need for passwords to authenticate when using trusted devices. During experimental sessions, several people expressed a desire for the authentication device to somehow streamline authentication compared to typical password entries. Other users were surprised that passwords were still needed after setup. Many participants felt that the second factor was an overkill, or too much of a burden in exchange for the no cognitive benefit.

West and Garg had two recommendations that address this major challenge to acceptability: reducing costs associated with security and improving rewards for good decisions [7,25]. Specifically, we recommend a visible reduction of the cognitive load of passwords in return for use of FIDO to improve acceptability. Streamlining could be achieved by not prompting the user for the full password as long as the proper FIDO token was plugged into a trusted device, or by allowing users to have a shorter, easier to use password when the device is present.

Highlight the Features. The FIDO standard is designed so that a single key can be used with multiple accounts without revealing any link between the two accounts even if service providers collude. This feature is crucial to the scalability of U2F for end users; without it they would need to obtain and manage at least one token per account. Unfortunately, only about half of our sample understood that a single U2F token could be used with multiple accounts from different service providers.

5.5 Phase-II Results

The second experimental phase consisted of running an identical protocol with a similar sample of participants. We again focused on analyzing the usability and acceptability of the two-factor authentication tool after the changes described in Sect. 5.3.

Tables 1 and 2 show a comparison of halt points and confusion points between the two phases. We observe that Phase-I had a higher percentage of halt points and confusion points when combined.

The demo and going to the incorrect settings were significant halt point in Phase-I, but in Phase-II, most of the participants did not require any intervention from the researchers for resolving this issue.

Several participants expressed confusion over whether the device would operate with Apple devices due to the implementation of the new USB-C port. While this problem can be solved by using YubiKey 4C, it is beyond the scope of the current experiment. It is worth noting that presently YubiKeys are not compatible with browsers other than Chrome or Opera. In this vein, the partici-

pants strongly recommended that YubiKeys be made compatible with all other browsers.

Phase-II include two conditions, as with Phase-I. The Google condition in Phase-II directed participants to Google Support's instructions on how to add and register the Yubico security key [23].

5.6 Phase-II Acceptability

In Phase-I of the experiment, no participant chose to continue the use of the token after registering it in our experiment. In Phase-II, most of the participants chose to keep the security keys after the experiment. However, the follow-up survey had low participation and hence, was not coded. Yet we note in Phase-II that five of ten participants reported continued use of the key on the survey. 2FA that requires pairing with a smart device is likely the only exposure to 2FA technology that many students have due to a compulsory 2FA that the University has implemented on the students to login to use University services. However, lack of participation in the follow-up study could be an indicator of lack of engagement with the token.

Communicate the Intrinsic Benefit. Confirmation of operation remains a serious issue underlying acceptability. If any artifact is not seen as working then it will not be seen to have benefits. When asked about continued use, one of the participants said, *"No, my password is secure enough and alerts are active."*

The instructions in the updated Yubico condition included information about the association of the device with other websites such as Facebook, and Salesforce. The links to the other sets of instructions also provided benefit information. Several users pointed out that multiple platforms could be linked and secured by YubiKeys.

Create a Cognitive Benefit. Google continues to require use of the full password even with Yubico Security Keys, and does not offer a decreased cognitive burden option. Thus, there was no cognitive benefit for using the device. As noted by a participant in Phase-II, the question arose, *Why is it still asking for a password?*. Remembering a password which adheres to the password rules remains a challenge if one still needs it along with the hardware tokens.

Highlight the Engineering. In the first phase, participants believed that they required different tokens for different websites. Phase-II participants indicated some awareness of the potential benefit of Security Key across different websites.

Communicating the Risks. One Phase-II participant identified risk mitigation as a reason to use the Security Key, stating that it is, *"more secure against brute force or stolen password."* In Phase-II, due to various alterations in the design, description, and specifically in the demo of the Yubico instructions, participants found it more usable and acceptable in their day-to-day life. We discussed more on how the problems in Phase-I were mitigated in Sect. 5.3, along with a discussion on further recommendations in Sect. 7 to make the device more acceptable and to make the online usage experience of users more secure.

Table 1. Percentage of participants encountering halt points: comparison

	Phase-I		Phase-II	
Halt point (stop)	Yubico	Google	Yubico	Google
Demo	72.7%	0%	0%	0%
Incorrect settings	72.7%	20.0%	19.04%	14.29%
Instruction	36.4%	20.0%	4.76%	0%
Form factor	9%	10%	4.76%	0%
Biometric	9%	0%	0%	0 %
Pressing button	9%	0%	0%	7.14%

Table 2. Percentage of participants encountering confusion points: comparison

	Phase-I		Phase-II	
Confusion point	Yubico	Google	Yubico	Google
Demo	9%	0%	0%	0%
Incorrect settings	18.2%	0%	4.76%	0%
Instruction	9%	20%	23.80%	71.43%
Form factor	9%	0%	23.80%	7.14%
Biometric	9%	0%	0%	0%
Pressing button	9%	10%	23.80%	28.57%

5.7 Comparison of Two Phases

The modification of the instructions and other changes mentioned in Sect. 5.3 made the Security Keys more usable. Table 3 list the statistically significant changes between the two studies. Figure 2 shows a stark improvement in the halt and confusion points in between the two phases.

The most important changes were the removal of the demo, the presentation of the devices so that they were easily identified, and links to the sites in which the Yubico Security Key can be used. Yubico's removal of their own instructions was a major improvement. Instead, the Yubico website redirected the user to the website the person was seeking to secure. Though the instructions still remains slightly confusing to the participants, which can be improved further.

Yubico Instructions. In Phase-I we noted that participants were not happy with the instructions especially those who received the Yubico instructions. We found that 72.7% who got the Yubico instructions were restricted by the demo, 36.4% found the instructions unclear and were stopped with the way the instructions were given, and 72.7% of the participants didn't understand which setting to go to setup their device. One of the participants also exclaimed by saying *"This is a horrible web site. I don't know what it wants from me."* In Phase-II, though some of the participants found the instructions verbose, the problems faced by them were reduced.

Biometric versus Touch. We noted that participants were more aware of how to press the golden button in Phase-II and no one faced difficulty and asked for assistance from researchers as compared to the 9% in the Phase-I. The instructions helped users in knowing about the keys and the participants expressed awareness that the button was not a biometric.

Demo. In Phase-I, the participants who received the Yubico instructions were confused by the demo setup, resulting in 72.7% of the participants failing to register the keys with their account even with the help of the researchers. Removal of the demo from the instructions in Phase-II removed the halt points found in Phase-I completely. Everyone in the second phase was able to register the keys and associate them to their Gmail account as described in Tables 1 and 2.

Table 3. Table of significance with Kruskal-Wallis test (p value greater than 0.2 are excluded)

	Phase-I	Phase-II	Yubico	Google
Halt point	Y vs. G	Y vs. G	I v. II	I v. II
Demo	0.0008	-	0.0033	-
Settings	0.0183	-	0.0033	-
Instructions	-	-	0.0213	0.0988
Form factor	-	-	-	-
Biometric	-	-	0.1671	-
Pressing button	-	0.2037	0.1671	-

Table 3 shows the results of a Kruskal–Wallis test comparing the two phases. Any p value greater than 0.05 is not significant. Those which are borderline (i.e., between 0.05 and 0.2) are included in the table as these also may be interesting for future experimental evaluations. Those not included were not distinguishable from random chance and we would not focus on them in future work.

6 Analysis

We implemented a set of correlation matrices to examine the potential interaction of the halt points and confusion points. For several people where we observed the correlation between form factor and settings in practice, this took the path of not understanding the nature of the device, in that they treated the device as a memory storage devices rather than an authentication device. Please note, the first phase of the experiment was identification of the device, so that all participants would have seen a description of these as authentication devices. Before having been directed to the settings, the participants searched their computers for this additional memory device and a few also inserted the device upside-down.

We also examined the correlation of halt points for Phase-II participants who received instructions from Google, finding only one non-zero correlation. In this case, the difficulty of finding settings was correlated with the operation of the button. This means that once the settings were identified, the participants were confused on when to engage with the pressure sensor in the enrollment phase.

Conversely, for those participants who interacted with the Yubico instructions the correlation between the confusion points of finding settings and interacting with the pressure sensor was negative (as shown in the correlation matrices in the Appendix). The only positive correlation was between difficulty in understanding the instruction and operating with the button by touching it at the correct time.

For the confusion points for participants who received the instructions from Google is also provided in the Appendix. The matrix shows an unsurprising correlation between not understanding the form factor and not being able to interact

with the button. This is unsurprising as individuals who, for example, placed the device upside down can neither see the illumination nor reach the button itself. A second positive correlation was found between difficulty understanding the instructions and interacting with the button.

We cannot conclude that the confusion and halt points are independent. The correlation of different halt and confusion factors appear inter-related.

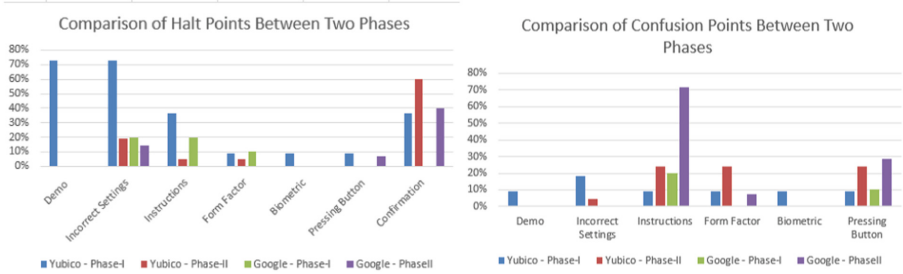


Fig. 2. Comparison of halt and confusion points between two study phases

7 Discussion and Further Suggestions

Based on student feedback, we characterize the lack of acceptability of the Security Key as lack of awareness of the risk, lack of knowledge about the benefits, and the fact that the burden of passwords is not mitigated so there is a lack of actual cognitive benefits. This lack of recognition of the intrinsic benefit appears to be a function of defaults and expertise. The members of the security lab implemented a pilot think-aloud protocol to augment experimenter training. None of the members of the lab selected the option to trust the computer. Therefore, the expertise of the employees at Google [13] and Yubico might make this difficulty effectively invisible. Literature on psychology of security illustrates that communicating security as a benefit rather than a cost can be expected to increase acceptability of a technology [7]. Behavioral economics of security indicates that presenting the safety of two factor as an asset that the participants possess, rather than having them experience it as a cost, has the potential to improve perceived value and increase long-term use [8].

As we cannot make all users experts, communication of the benefits and the creation of a cognitive benefit are most promising. Communication of benefit could occur in the form of a validation email and communicating benefit information upon enrollment. There are other possible points of interaction. For example, after a password reset email, a simple message indicating that the participant is safer could be provided. Currently, security information focuses heavily on risk avoidance information and rarely provides benefit information.

One possible form of benefit communication could be illustration of the options of different Security Keys.

A suggestion is to remind participants at first login after enrollment that the selected hardware is trusted. Explicit positive benefit messages could include initial congratulations on successful login the first time. After that a periodic reminder that "only this computer can log in without your key" with an image would provide clear indication of benefit. If any login is ever refused from a remote computer, a message to the participant indicating their successful triumph over a potential attacker would clearly identify a benefit.

Even in the second condition, many participants did not understand the benefit of the device as compared to a longer, more secure password. To address this, providers that support 2FA could include a pointer to U2F tokens when there are suspicious login attempts. In both phases, multiple participants expressed disappointment that their full password was still required after configuring the Security Key, even on trusted devices where a second factor was not needed. From a user experience perspective, pressing the single button to activate a U2F token presents a lower physical and cognitive load compared to typing a password [4,22]. From a security perspective, the authentication provided by the token is stronger than any password a normal participant is likely to choose. As an alternative to this, one can use a shorter password with a few characters along with the Yubico Security Key rather than a password phrase.

Using the security token as a primary authentication factor also offers accessibility benefits. For enterprise customers, it could ease ADA compliance with respect to authentication requirements for employees. Individuals who can be supported through voice recognition or other alternative means of entering text often still struggle with authentication, particularly when required to submit password phrases. Although still an unusual complaint, an ADA compliance issue could arise in the face of password complexity requirements.

In this experiment we studied the usability and acceptability of the FIDO U2F security key. Further studies could include a range of tokens, not only other Yubico security keys such as YubiKey 4, YubiKey 4 Nano, YubiKey 4C, and YubiKey NEO but also pico and other secure hardware. In addition, a goal of future work is to include vulnerable populations. Such populations are likely to have lower expertise but may have greater awareness of risk. We choose the undergraduates because of their increased skill in relation to computer and security and that they are early adopters of new technology but in the future we hope to expand this study for more diverse age groups including retirees.

8 Conclusion

Through a two-phase experimental study, we investigated the usability issues of the FIDO Security Key. In the first phase, we discovered there were six primary usability concerns or halt points - a confusing demo, going to incorrect account settings, confusing instructions, the hardware's form-factor, validation after setup, and participants' doubts regarding the benefit of the Security Key.

In the second phase of the experiment, the usability concerns were mostly mitigated. These included updating installation instructions, removing the demo from the setup work flow, and clarifying descriptions of the registration process. This resulted in a remarkable improvement in the usability of the device - while 33.3% of the users were not able to complete the registration process in the first phase, everyone was successful in the second phase. In the second phase, although the halt points were reduced considerably, many participants were still confused whether the key was working due to the absence of message confirmation at the end of the registration process.

The improvement of usability did not automatically result in improvements in acceptability. Participants continued to express belief in the strength of passwords alone, showing undue faith in their own security acumen. We conclude with compliments to the usability of the 2FA token, and with a warning that even the best designed hardware will not be used if the benefits are not apparent.

Acknowledgement. This research was supported in part by the National Science Foundation under CNS 1565375, Cisco Research Support #591000, and the Comcast Innovation Fund. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the the US Government, the National Science Foundation, Cisco, Comcast, or Indiana University.

A Appendix

We have provided the correlation matrices of the halt and confusion points for different sets of instructions (Yubico and Google) across the Two Phases. Due to lack of space we have used abbreviations for the halt and confusion points. The abbreviation list are as follows:

1. D: Demo
2. S: Incorrect Settings
3. I: Instructions
4. F: Form Factor
5. B: Bio-metric
6. P: Pressing Button

10.1 Phase-I

$$\begin{array}{c} \begin{array}{cccccc} D & S & I & F & B & P \end{array} \\ \begin{bmatrix} 1 & 1 & 0.04 & 0.19 & 0.19 & 0.19 \\ 1 & 1 & 0.04 & 0.19 & 0.19 & 0.19 \\ 0.04 & 0.04 & 1 & -0.24 & 0.42 & 0.42 \\ 0.19 & 0.19 & -0.24 & 1 & -0.1 & -0.1 \\ 0.19 & 0.19 & 0.42 & -0.1 & 1 & -0.1 \\ 0.19 & 0.19 & 0.42 & -0.1 & -0.1 & 1 \end{bmatrix} \begin{array}{c} D \\ S \\ I \\ F \\ B \\ P \end{array} \end{array}$$

Correlation Matrix of Halt Points for Phase-I participants who received Yubico Instructions.

$$\begin{matrix} D & S & I & F & B & P & \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.38 & -0.17 & 0 & 0 \\ 0 & 0.38 & 1 & 0.67 & 0 & 0 \\ 0 & -0.17 & 0.67 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} & \begin{matrix} Demo \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{matrix}$$

Correlation Matrix of Halt Points for Phase-I participants who received Google Instructions.

$$\begin{matrix} D & S & I & F & B & P & \\ \begin{bmatrix} 1 & -0.15 & -0.1 & -0.1 & -0.1 & -0.1 \\ -0.15 & 1 & -0.15 & -0.15 & -0.15 & -0.15 \\ -0.1 & -0.15 & 1 & -0.1 & -0.1 & -0.1 \\ -0.1 & -0.15 & -0.1 & 1 & 1 & -0.1 \\ -0.1 & -0.15 & -0.1 & 1 & 1 & -0.1 \\ -0.1 & -0.15 & -0.1 & -0.1 & -0.1 & 1 \end{bmatrix} & \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{matrix}$$

Correlation Matrix of Confusion Points for Phase-I participants who received Yubico Instructions.

$$\begin{matrix} D & S & I & F & B & P & \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -0.15 & -0.15 & -0.15 & -0.15 \\ 0 & -0.15 & 1 & -0.1 & -0.1 & -0.1 \\ 0 & -0.15 & -0.1 & 1 & 1 & -0.1 \\ 0 & -0.15 & -0.1 & 1 & 1 & -0.1 \\ 0 & -0.15 & -0.1 & -0.1 & -0.1 & 1 \end{bmatrix} & \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{matrix}$$

Correlation Matrix of Confusion Points for Phase-I participants who received Google Instructions.

10.2 Phase-II

$$\begin{array}{c} \begin{matrix} D & S & I & F & B & P \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -0.11 & 0.46 & 0 & 0 \\ 0 & -0.11 & 1 & -0.05 & 0 & 0 \\ 0 & 0.46 & -0.05 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{array}$$

Correlation Matrix of Halt Points for Phase-II participants who received Yubico Instructions.

$$\begin{array}{c} \begin{matrix} D & S & I & F & B & P \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0.68 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0.68 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{array}$$

Correlation Matrix of Halt Points for Phase-II participants who received Google Instructions.

$$\begin{array}{c} \begin{matrix} D & S & I & F & B & P \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -0.13 & 0.4 & 0 & -0.13 \\ 0 & -0.13 & 1 & -0.31 & 0 & 0.21 \\ 0 & 0.4 & -0.31 & 1 & 0 & -0.31 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -0.13 & 0.21 & -0.31 & 0 & 1 \end{bmatrix} \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{array}$$

Correlation Matrix of Confusion Points for Phase-II participants who received Yubico Instructions.

$$\begin{array}{c} \begin{matrix} D & S & I & F & B & P \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.18 & 0 & 0.4 \\ 0 & 0 & 0.18 & 1 & 0 & 0.44 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.4 & 0.44 & 0 & 1 \end{bmatrix} \begin{matrix} D \\ S \\ I \\ F \\ B \\ P \end{matrix} \end{array}$$

Correlation Matrix of Confusion Points for Phase-II participants who received Google Instructions.

References

1. Acquisti, A., Brandimarte, L., Loewenstein, G.: Privacy and human behavior in the age of information. Science **347**(6221), 509–514 (2015). http://science.sciencemag.org/content/347/6221/509.short. Accessed 04 May 2017
2. Bauer, L., et al.: A user study of policy creation in a exible access-control system. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 543–552 (2008)
3. Biddle, R., Chiasson, S., Van Oorschot, P.C.: Graphical passwords: learning from the first twelve years. ACM Comput. Surv. (CSUR) **44**(4), 19 (2012)
4. Bonneau, J., et al.: The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy (SP), May 2012, pp. 553–567. https://doi.org/10.1109/SP.2012.44, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6234436
5. Camp, L.J., Abbott, J., Chen, S.: CPasswords: leveraging episodic memory and human-centered design for better authentication. In: 2016 49th Hawaii International Conference on System Sciences (HICSS), January 2016, pp. 3656–3665. https://doi.org/10.1109/MTS.2013.2241294
6. Fagan, M., Khan, M.M.H.: Why do they do what they do?: A study of what motivates users to (not) follow computer security advice. In: Twelfth Symposium on Usable Privacy and Security (SOUPS 2016) (2016)
7. Garg, V., Camp, J.: Heuristics and biases: implications for security design. IEEE Technol. Soc. Mag. **32**(1), 73–79 (2013). https://doi.org/10.1109/MTS.2013.2241294. ISSN 0278–0097
8. Grossklags, J., Acquisti, A.: When 25 cents is too much: an experiment on willingness-to-sell and willingness-to-protect personal information. In: WEIS (2007)
9. Inglesant, P.G., Sasse, M.A.: The true cost of unusable password policies: password use in the wild. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 383–392 (2010)
10. Kelley, T., Rajivan, P., Camp, L.J.: An assessment of computer and security expertise. Technical report, March 2014
11. Komanduri, S., et al.: Of passwords and people: measuring the effect of password-composition policies. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 2595–2604 (2011)
12. Krol, K., et al.: "They brought in the horrible key ring thing!" analysing the usability of two-factor authentication in uk online banking. arXiv preprint arXiv:1501.04434 (2015)
13. Lang, J., Czeskis, A., Balfanz, D., Schilder, M., Srinivas, S.: Security keys: practical cryptographic second factors for the modern web. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 422–440. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_25. http://fc16.ifca.ai/preproceedings/25_Lang.pdf
14. McDowell, B.: Strong Authentication Canine, June 2015. https://www.youtube.com/watch?v=sdJ47NFGlgk
15. M'Raihi, D., et al.: Rfc 6238-totp: time-based one-time password algorithm (2011)

16. M'Raihi, D., et al.: RFC 4226: HOTP: an HMAC-based one-time password algorithm (2005)
17. New password guidelines say everything we thought about passwords is wrong. VentureBeat, 18 April 2017. https://venturebeat.com/2017/04/18/new-password-guidelines-say-everything-we-thought-about-passwords-is-wrong/. Accessed 04 May 2017
18. Norcie, G., et al.: Why Johnny can't blow the whistle: identifying and reducing usability issues in anonymity systems. In: Internet Society (2014). https://doi.org/10.14722/usec.2014.23022, http://www.internetsociety.org/doc/why-johnny-cant-blow-whistle-identifying-and-reducing-usability-issues-anonymity-systems. Accessed 11 May 2017. ISBN 978-1-891562-37-2
19. Rajivan, P., et al.: What can Johnny do?-Factors in an end-user expertise instrument. In: Proceedings of the Tenth International Symposium on Human Aspects of Information Security & Assurance (HAISA 2016). Lulu.com, p. 199 (2016)
20. Reeder, R.W., Maxion, R.A.: User interface dependability through goal-error prevention. In: International Conference on Dependable Systems and Networks, DSN 2005, Proceedings. IEEE, pp. 60–69 (2005)
21. Srinivas, S., et al.: Universal 2nd factor (U2F) overview. In: FIDO Alliance Proposed Standard, pp. 1–5 (2015)
22. Stajano, F.: Pico: no more passwords!. In: Christianson, B., Crispo, B., Malcolm, J., Stajano, F. (eds.) Security Protocols 2011. LNCS, vol. 7114, pp. 49–81. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25867-1_6
23. Use Security Key for 2-Step Verification - Android. https://support.google.com/accounts/answer/6103523?hl=en&ref_topic=6103521
24. Wash, R., et al.: Understanding password choices: How frequently entered passwords are re-used across websites. In: Symposium on Usable Privacy and Security (SOUPS) (2016)
25. West, R.: The psychology of security. Commun. ACM **51**(4), 34–40 (2008). http://dl.acm.org/citation.cfm?id=1330320. Accessed 05 Apr 2017
26. Whitten, A., Tygar, J.D.: Why: Johnny can't encrypt: a usability evaluation of PGP 5.0. In: USENIX Security Symposium, vol. 99 (1999)
27. Zurko, M.E., Simon, R.T.: User-centered security. In: Proceedings of the 1996 workshop on New security paradigms. ACM, pp. 27–33 (1996)

Privacy and Data Processing

High-Precision Privacy-Preserving Real-Valued Function Evaluation

Christina Boura[1,2(✉)], Ilaria Chillotti[2], Nicolas Gama[1,2], Dimitar Jetchev[1,3], Stanislav Peceny[1], and Alexander Petric[1]

[1] Inpher, New-York, USA
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
christina.boura@uvsq.fr
[3] École Polytechnique Fédérale de Lausanne, EPFL SB MATHGEOM GR-JET, Lausanne, Switzerland

Abstract. We propose a novel multi-party computation protocol for evaluating continuous real-valued functions with high numerical precision. Our method is based on approximations with Fourier series and uses at most two rounds of communication during the online phase. For the offline phase, we propose a trusted-dealer and honest-but-curious aided solution, respectively. We apply our algorithm to train a logistic regression classifier via a variant of Newton's method (known as IRLS) to compute unbalanced classification problems that detect rare events and cannot be solved using previously proposed privacy-preserving optimization algorithms (e.g., based on piecewise-linear approximations of the sigmoid function). Our protocol is efficient as it can be implemented using standard quadruple-precision floating point arithmetic. We report multiple experiments and provide a demo application that implements our algorithm for training a logistic regression model.

1 Introduction

Privacy-preserving computing allows multiple parties to evaluate a function while keeping the inputs private and revealing only the output of the function and nothing else. Recent advances in multi-party computation (MPC), homomorphic encryption, and differential privacy made these models practical. An example of such computations, with applications in medicine and finance, among others, is the training of supervised models where the input data comes from distinct secret data sources [18,24,26,27] and the evaluation of predictions using these models.

In machine learning classification problems, one trains a model on a given dataset to predict new inputs, by mapping them into discrete categories. The classical *logistic regression* model predicts a class by providing a probability associated with the prediction. The quality of the model can be measured in several

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 183–202, 2018.
https://doi.org/10.1007/978-3-662-58387-6_10

ways, the most common one being the *accuracy* that indicates the percentage of correctly predicted answers.

It appears that for a majority of the datasets (e.g., the MNIST database of handwritten digits [16] or the ARCENE dataset [15]), the classification achieves very good accuracy after only a few iterations of the gradient descent using a piecewise-linear approximation of the sigmoid function $\texttt{sigmo} : \mathbb{R} \to [0, 1]$ defined as

$$\texttt{sigmo}(x) = \frac{1}{1 + e^{-x}},$$

although the current cost function is still far from the minimum value [26]. Other approximation methods of the sigmoid function have also been proposed in the past. In [30], an approximation with low degree polynomials resulted in a more efficient but less accurate algorithm. Conversely, a higher-degree polynomial approximation applied to deep learning algorithms in [25] yielded more accurate, but less efficient algorithms (and thus, less suitable for privacy-preserving computing). In parallel, approximation solutions for privacy-preserving methods based on homomorphic encryption [2,28], [19,23] and differential privacy [1,11] have been proposed in the context of both classification algorithms and deep learning.

Nevertheless, accuracy itself is not always a sufficient measure for the quality of the model, especially if, as mentioned in [20, p. 423], our goal is to detect a rare event such as a rare disease or a fraudulent financial transaction. If, for example, one out of every one thousand transactions is fraudulent, a naïve model that classifies all transactions as honest achieves 99.9% accuracy; yet this model has no predictive capability. In such cases, measures such as *precision*, *recall* and *F1-score* allow for better estimating the quality of the model. They bound the rates of false positives or negatives relative to only the positive events rather than the whole dataset.

The techniques cited above achieve excellent accuracy for most balanced datasets, but since they rely on a rough approximation of the sigmoid function, they do not converge to the same model and thus, they provide poor scores on datasets with a very low acceptance rate. In this paper, we show how to regain this numerical precision in MPC, and to reach the same score as the plaintext regression. Our MPC approach is mostly based on additive secret shares with precomputed multiplication triplets [4]. This means that the computation is divided in two phases: an *offline* phase that can be executed before the data is shared between the players, and an *online phase* that computes the actual result. For the offline phase, we propose a first solution based on a *trusted dealer*, and then discuss a protocol where the dealer is *honest-but-curious*.

1.1 Our Contributions

Fourier Approximation of the Sigmoid Function. Evaluation of real-valued functions has been widely used in privacy-preserving computations. For instance, in order to train linear and logistic regression models, one is required to compute real-valued functions such as the square root, the exponential, the logarithm, the

sigmoid or the softmax function and use them to solve non-linear optimization problems. In order to train a logistic regression model, one needs to minimize a cost function which is expressed in terms of logarithms of the continuous sigmoid function. This minimum is typically computed via iterative methods such as the gradient descent. For datasets with low acceptance rate, it is important to get much closer to the exact minimum in order to obtain a sufficiently precise model. We thus need to significantly increase the number of iterations (naïve or stochastic gradient descent) or use faster-converging methods (e.g., IRLS [5, Sect. 4.3]). The latter require a numerical approximation of the sigmoid that is much better than what was previously achieved in an MPC context, especially when the input data is not normalized or feature-scaled. Different approaches have been considered previously such as approximation by Taylor series around a point (yielding only good approximation locally at that point) or polynomial approximation (by e.g., estimating least squares). Although better than the first one, this method is numerically unstable due to the variation of the sizes of the coefficients. An alternative method based on approximation by piecewise-linear functions has been considered as well. In MPC, this method performs well when used with garbled circuits instead of secret sharing and masking, but does not provide enough accuracy.

In our case, we approximate the sigmoid using Fourier series, an approach applied for the first time in this context. This method works well as it provides a better uniform approximation assuming that the function is sufficiently smooth (as is the case with the sigmoid). In particular, we virtually re-scale and extend the sigmoid to a periodic function that we approximate with a trigonometric polynomial which we then evaluate in a stable privacy-preserving manner. To approximate a generic function with trigonometric polynomials that can be evaluated in MPC, one either uses the Fourier series of a smooth periodic extension or finds directly the closest trigonometric polynomial by the method of least squares for the distance on the half-period. The first approach yields a super-algebraic convergence at best, whereas the second converges exponentially fast. On the other hand, the first one is numerically stable whereas the second one is not (under the standard Fourier basis). In the case of the sigmoid, we show that one can achieve both properties at the same time.

Floating-Point Representation and Masking. A typical approach to multi-party computation protocols with masking is to embed fixed-point values into finite groups and use uniform masking and secret sharing. Arithmetic circuits can then be evaluated using, e.g., precomputed multiplication triplets and following Beaver's method [4]. This idea has been successfully used in [14] and [13]. Whereas the method works well on low multiplicative depth circuits like correlations or linear regression [18], in general, the required group size increases exponentially with the multiplicative depth. In [26], this exponential growth is mitigated by a two-party rounding solution, but the technique does not extend to three or more players where an overflow in the most significant bits can occur. In this work, we introduce an alternative sharing scheme, where fixed-point values are shared directly using (possibly multibit) floating points, and present a tech-

nique to reduce the share sizes after each multiplication. This technique easily extends to an arbitrary number of players.

Significant Reduction in Communication Time. In this paper, we follow the same approach as in [26] and define dedicated triplets for high-level instructions such as large matrix multiplications, a system resolution, or an oblivious evaluation of the sigmoid. This approach is less generic than masking low-level instructions as in SPDZ, but it allows to reduce the communication and memory requirements by large factors. Masks and operations are aware of the type of vector or matrix dimensions and benefit from the vectorial nature of the high-level operations. For example, multiplying two matrices requires a single round of communication instead of up to $O(n^3)$ for coefficient-wise approaches, depending on the batching quality of the compiler. Furthermore, masking is defined per immutable variable rather than per elementary operation, so a constant matrix is masked only once during the whole algorithm. Combined with non-trivial local operations, these triplets can be used to achieve much more than just ring additions or multiplications. In a nutshell, the amount of communications is reduced as a consequence of reusing the same masks, and the number of communication rounds is reduced as a consequence of masking directly matrices and other large structures. Therefore, the total communication time becomes negligible compared to the computing cost.

New Protocol for the Honest But Curious Offline Phase Extendable to n Players. We introduce a new protocol for executing the offline phase in the honest-but-curious model that is easily extendable to a generic number n of players while remaining efficient. To achieve this, we use a broadcast channel instead of peer-to-peer communication which avoids a quadratic explosion in the number of communications. This is an important contribution, as none of the previous protocols for $n > 3$ players in this model are efficient. In [18], for instance, the authors propose a very efficient algorithm in the trusted dealer model; yet, the execution time of the oblivious transfer protocol is quite slow.

2 Notation and Preliminaries

Assume that $P_1, \ldots, P_n$ are distinct computing parties (players). We recall some basic concepts from multi-party computation that will be needed for this paper.

2.1 Secret Sharing and Masking

Let $(G, \bullet)$ be a group and let $x \in G$ be a group element.

A *secret share* of x, denoted by $[\![x]\!]_\bullet$ (by a slight abuse of notation), is a tuple $(x_1, \ldots, x_n) \in G^n$ such that $x = x_1 \bullet \cdots \bullet x_n$. If $(G, +)$ is Abelian, we call the secret shares $x_1, \ldots, x_n$ *additive secret shares*. A secret sharing scheme is computationally secure if for any two elements $x, y \in G$, strict sub-tuples of shares $[\![x]\!]_\bullet$ or $[\![y]\!]_\bullet$ are indistinguishable. If G admits a uniform distribution, an information-theoretic secure secret sharing scheme consists of drawing $x_1, \ldots, x_{n-1}$ uniformly

at random and choosing $x_n = x_{n-1}^{-1} \bullet \cdots \bullet x_1^{-1} \bullet x$. When G is not compact, the condition can be relaxed to statistical or computational indistinguishability.

A closely-related notion is the one of *group masking*. Given a subset $\mathcal{X}$ of G, the goal of masking $\mathcal{X}$ is to find a distribution $\mathcal{D}$ over G such that the distributions of $x \bullet \mathcal{D}$ for $x \in \mathcal{X}$ are all indistinguishable. Indeed, such distribution can be used to create a secret share: one can sample $\lambda \leftarrow \mathcal{D}$, and give λ^{-1} to a player and $x \bullet \lambda$ to the other. Masking can also be used to evaluate non-linear operations in clear over masked data, as soon as the result can be privately unmasked via homomorphisms (as in, e.g., the Beaver's triplet multiplication technique [4]).

2.2 Arithmetic with Secret Shares via Masking

Computing secret shares for a sum $x + y$ (or a linear combination if $(G, +)$ has a module structure) can be done non-interactively by each player by adding the corresponding shares of x and y. Computing secret shares for a product is more challenging. One way to do that is to use an idea of Beaver based on precomputed and secret shared multiplicative triplets. From a general point of view, let $(G_1, +)$, $(G_2, +)$ and $(G_3, +)$ be three abelian groups and let $\pi \colon G_1 \times G_2 \to G_3$ be a bilinear map.

Given additive secret shares $[\![x]\!]_+$ and $[\![y]\!]_+$ for two elements $x \in G_1$ and $y \in G_2$, we would like to compute secret shares for the element $\pi(x, y) \in G_3$. With Beaver's method, the players must employ precomputed single-use random triplets $([\![\lambda]\!]_+, [\![\mu]\!]_+, [\![\pi(\lambda, \mu)]\!]_+)$ for $\lambda \in G_1$ and $\mu \in G_2$, and then use them to mask and reveal $a = x + \lambda$ and $b = y + \mu$. The players then compute secret shares for $\pi(x, y)$ as follows:

- Player 1 computes $z_1 = \pi(a, b) - \pi(a, \mu_1) - \pi(\lambda_1, b) + (\pi(\lambda, \mu))_1$;
- Player i (for $i = 2, \ldots, n$) computes $z_i = -\pi(a, \mu_i) - \pi(\lambda_i, b) + (\pi(\lambda, \mu))_i$.

The computed $z_1, \ldots, z_n$ are the additive shares of $\pi(x, y)$. A given λ can be used to mask only one variable, so one triplet must be precomputed for each multiplication during the *offline phase* (i.e. before the data is made available to the players). Instantiated with the appropriate groups, this abstract scheme allows to evaluate a product in a ring, but also a vectors dot product, a matrix-vector product, or a matrix-matrix product.

2.3 MPC Evaluation of Real-Valued Continuous Functions

For various applications (e.g., logistic regression in Sect. 6), we need to compute continuous real-valued functions over secret shared data. For non-linear functions (e.g. exponential, log, power, cos, sin, sigmoid, etc.), different methods are proposed in the literature.

A straightforward approach consists of implementing a full floating point arithmetic framework [6,13], and to compile a data-oblivious algorithm that evaluates the function over floats. This is, e.g., what Sharemind and SPDZ use. However, these two generic methods lead to prohibitive running times if the floating point function has to be evaluated millions of times.

The second approach is to replace the function with an approximation that is easier to compute: for instance, [26] uses garbled circuits to evaluate fixed point comparisons and absolute values; it then replaces the sigmoid function in the logistic regression with a piecewise-linear function. Otherwise, [25] approximates the sigmoid with a polynomial of fixed degree and evaluates that polynomial with the Horner method, thus requiring a number of rounds of communications proportional to the degree.

Another method that is close to how SPDZ [14] computes inverses in a finite field is based on polynomial evaluation via multiplicative masking: using a precomputed triplet of the form $(\llbracket\lambda\rrbracket_+, \llbracket\lambda^{-1}\rrbracket_+, \ldots, \llbracket\lambda^{-p}\rrbracket_+)$, players can evaluate $P(x) = \sum_{i=0}^{p} a_p x^p$ by revealing $u = x\lambda$ and outputting the linear combination $\sum_{i=0}^{p} a_i u^i \llbracket\lambda^{-i}\rrbracket_+$.

Multiplicative masking, however, involves some leakage: in finite fields, it reveals whether x is null. The situation gets even worse in finite rings where the multiplicative orbit of x is disclosed (for instance, the rank would be revealed in a ring of matrices), and over $\mathbb{R}$, the order of magnitude of x would be revealed.

For real-valued polynomials, the leakage could be mitigated by translating and rescaling the variable x so that it falls in the range $[1, 2)$. Yet, in general, the coefficients of the polynomials that approximate the translated function explode, thus causing serious numerical issues.

2.4 Full Threshold Honest-But-Curious Protocol

Since our goal is to emphasize new functionalities, such as efficient evaluation of real-valued continuous functions and good quality logistic regression, we often consider a scenario where all players follow the protocol without introducing any errors. The players may, however, record the whole transaction history and try to learn illegitimate information about the data. During the online phase, the security model imposes that any collusion of at most $n-1$ players out of n cannot distinguish any semantic property of the data beyond the aggregated result that is legitimately and explicitly revealed. To achieve this, Beaver triplets (used to mask player's secret shares) can be generated and distributed by a single entity called the *trusted dealer*. In this case, no coalition of at most $n-1$ players should get any computational advantage on the plaintext triplet information. However, the dealer himself knows the plaintext triplets, and hence the whole data, which only makes sense on some computation outsourcing use-cases. In Sect. 5, we give an alternative *honest-but-curious* (or semi-honest) protocol to generate the same triplets, involving this time bi-directional communications with the dealer. In this case, the dealer and the players collaborate during the offline phase in order to generate the precomputed material, but none of them have access to the whole plaintext triplets. This makes sense as long as the dealer does not collude with any player, and at least one player does not collude with the other players. We leave the design of actively secure protocols for future work.

3 Statistical Masking and Secret Share Reduction

In this section, we present our masking technique for fixed-point arithmetic and provide an algorithm for the MPC evaluation of real-valued continuous functions. In particular, we show that to achieve p bits of numerical precision in MPC, it suffices to have $p + 2\tau$-bit floating points where τ is a fixed security parameter.

The secret shares we consider are real numbers. We mask these shares using floating point numbers. Yet, as there is no uniform distribution on $\mathbb{R}$, no additive masking distribution over reals can perfectly hide the arbitrary inputs. In the case when the secret shares belong to some known range of numerical precision, it is possible to carefully choose a masking distribution, depending on the precision range, so that the masked value computationally leaks no information about the input. A distribution with sufficiently large standard deviation could do the job: for the rest of the paper, we refer to this type of masking as "statistical masking". In practice, we choose a normal distribution with standard deviation $\sigma = 2^{40}$.

On the other hand, by using such masking, we observe that the sizes of the secret shares increase every time we evaluate the multiplication via Beaver's technique (Sect. 2.2). In Sect. 3.3, we address this problem by introducing a technique that allows to reduce the secret share sizes by discarding the most significant bits of each secret share (using the fact that the sum of the secret shares is still much smaller than their size).

3.1 Floating Point, Fixed Point and Interval Precision

Suppose that B is an integer and that p is a non-negative integer (the number of bits). The class of fixed-point numbers of exponent B and numerical precision p is:

$$\mathcal{C}(B, p) = \{x \in 2^{B-p} \cdot \mathbb{Z}, |x| \leq 2^B\}.$$

Each class $\mathcal{C}(B, p)$ is finite, and contains $2^{p+1}+1$ numbers. They could be rescaled and stored as $(p + 2)$-bit integers. Alternatively, the number $x \in \mathcal{C}(B, p)$ can also be represented by the floating point value x, provided that the floating point representation has at least p bits of mantissa. In this case, addition and multiplication of numbers across classes of the same numerical precision are natively mapped to floating-point arithmetic. The main arithmetic operations on these classes are:

- **Lossless Addition:** $\mathcal{C}(B_1, p_1) \times \mathcal{C}(B_2, p_2) \rightarrow \mathcal{C}(B, p)$ where $B = max(B_1, B_2) + 1$ and $p = B - \min(B_1 - p_1, B_2 - p_2)$;
- **Lossless Multiplication:** $\mathcal{C}(B_1, p_1) \times \mathcal{C}(B_2, p_2) \rightarrow \mathcal{C}(B, p)$ where $B = B_1 + B_2$ and $p = p_1 + p_2$;
- **Rounding:** $\mathcal{C}(B_1, p_1) \rightarrow \mathcal{C}(B, p)$, that maps x to its nearest element in $2^{B-p}\mathbb{Z}$.

Lossless operations require p to increase exponentially in the multiplication depth, whereas fixed precision operations maintain p constant by applying a final rounding. Finally, note that the exponent B should be incremented to

store the result of an addition, yet, B is a user-defined parameter in fixed point arithmetic. If the user forcibly chooses to keep B unchanged, any result $|x| > 2^B$ will not be representable in the output domain (we refer to this type of overflow as *plaintext overflow*.

3.2 Floating Point Representation

Given a security parameter τ, we say that a set $\mathcal{S}$ is a τ-secure *masking set* for a class $\mathcal{C}(B,p)$ if the following distinguishability game cannot be won with advantage $\geq 2^{-\tau}$: the adversary chooses two plaintexts m_0, m_1 in $\mathcal{C}(B,p)$, a challenger picks $b \in \{0,1\}$ and $\alpha \in \mathcal{S}$ uniformly at random, and sends $c = m_b + \alpha$ to the adversary. The adversary has to guess b. Note that increasing such distinguishing advantage from $2^{-\tau}$ to $\approx 1/2$ would require to give at least 2^τ samples to the attacker, so $\tau = 40$ is sufficient in practice.

Proposition 1. *The class* $\mathcal{C}(B,p,\tau) = \{\alpha \in 2^{B-p}\mathbb{Z}, |\alpha| \leq 2^{B+\tau}\}$ *is a* τ*-secure masking set for* $\mathcal{C}(B,p)$.

Proof. If $a, b \in \mathcal{C}(B,p)$ and $\mathcal{U}$ is the uniform distribution on $\mathcal{C}(B,p,\tau)$, the statistical distance between $a+\mathcal{U}$ and $b+\mathcal{U}$ is $(b-a)\cdot 2^{p-B}/\#\mathcal{C}(B,p,\tau) \leq 2^{-\tau}$. This distance upper-bounds any computational advantage. □

Again, the class $\mathcal{C}(B,p,\tau) = \mathcal{C}(B+\tau, p+\tau)$ fits in floating point numbers of $p+\tau$-bits of mantissa, so they can be used to securely mask fixed point numbers with numerical precision p. By extension, all additive shares for $\mathcal{C}(B,p)$ will be taken in $\mathcal{C}(B,p,\tau)$.

We now analyze what happens if we use Beaver's protocol to multiply two plaintexts $x \in \mathcal{C}(B_1,p)$ and $y \in \mathcal{C}(B_2,p)$. The masked values $x+\lambda$ and $y+\mu$ are bounded by $2^{B_1+\tau}$ and $2^{B_2+\tau}$ respectively. Since the mask λ is also bounded by $2^{B_1+\tau}$, and μ by $2^{B_2+\tau}$, the computed secret shares of $x \cdot y$ will be bounded by $2^{B_1+B_2+2\tau}$, So the lossless multiplication sends $\mathcal{C}(B_1,p,\tau) \times \mathcal{C}(B_2,p,\tau) \rightarrow \mathcal{C}(B,2p,2\tau)$ where $B = B_1+B_2$ instead of $\mathcal{C}(B,p,\tau)$. Reducing p is just a matter of rounding, and it is done automatically by the floating point representation. However, we still need a method to reduce τ, so that the output secret shares are bounded by $2^{B+\tau}$.

3.3 Secret Share Reduction Algorithm

The algorithm we propose depends on two auxiliary parameters: the *cutoff*, defined as $\eta = B + \tau$ so that 2^η is the desired bound in absolute value, and an auxiliary parameter $M = 2^\kappa$ larger than the number of players.

The main idea is that the initial share contains large components $z_1, \ldots, z_n$ that sum up to the small secret shared value z. Additionally, the most significant bits of the share beyond the cutoff position (say $\text{MSB}(z_i) = \lfloor z_i/2^\eta \rceil$) do not contain any information on the data, and are all safe to reveal. We also know that the MSB of the sum of the shares (i.e. MSB of the data) is null, so the sum of the MSB of the shares is very small. The share reduction algorithm simply

computes this sum, and redistributes it evenly among the players. Since the sum is guaranteed to be small, the computation is done modulo M rather than on large integers. More precisely, using the cutoff parameter η, for $i = 1, \ldots, n$, player i writes his secret share z_i of z as $z_i = u_i + 2^\eta v_i$, with $v_i \in \mathbb{Z}$ and $u_i \in [-2^{\eta-1}, 2^{\eta-1})$. Then, he broadcasts $v_i \bmod M$, so that each player computes the sum. The individual shares can optionally be re-randomized using a precomputed share $[\![\nu]\!]_+$, with $\nu = 0 \mod M$. Since $w = \sum v_i$'s is guaranteed to be between $-M/2$ and $M/2$, it can be recovered from its representation mod M. Thus, each player locally updates its share as $u_i + 2^\eta w/n$, which have by construction the same sum as the original shares, but are bounded by 2^η. This construction is summarized in Algorithm 3 in [7, Appendix B].

4 Fourier Approximation

Fourier theory allows us to approximate certain periodic functions with trigonometric polynomials. The goal of this section is two-fold: to show how to evaluate trigonometric polynomials in MPC and, at the same time, to review and show extensions of some approximation results to non-periodic functions.

4.1 Evaluation of Trigonometric Polynomials

Recall that a complex trigonometric polynomial is a finite sum of the form $t(x) = \sum_{m=-P}^{P} c_m e^{imx}$, where $c_m \in \mathbb{C}$ is equal to $a_m + ib_m$, with $a_m, b_m \in \mathbb{R}$. Each trigonometric polynomial is a periodic function with period 2π. If $c_{-m} = \overline{c_m}$ for all $m \in \mathbb{Z}$, then t is real-valued, and corresponds to the more familiar cosine decomposition $t(x) = a_0 + \sum_{m=1}^{N} a_m \cos(mx) + b_m \sin(mx)$. Here, we describe how to evaluate trigonometric polynomials in an MPC context, and explain why it is better than regular polynomials.

We suppose that, for all m, the coefficients a_m and b_m of t are publicly accessible and they are $0 \leq a_m, b_m \leq 1$. As t is 2π periodic, we can evaluate it on inputs modulo 2π. Remark that as $\mathbb{R} \mod 2\pi$ admits a uniform distribution, we can use a uniform masking: this method completely fixes the leakage issues that were related to the evaluation of classical polynomials via multiplicative masking. On the other hand, the output of the evaluation is still in $\mathbb{R}$: in this case we continue using the statistical masking described in previous sections. The inputs are secretly shared and additively masked: for sake of clarity, to distinguish the classical addition over reals from the addition modulo 2π, we temporarily denote this latter by $\oplus$. In the same way, we denote the additive secret shares with respect to the addition modulo 2π by $[\![\cdot]\!]_\oplus$. Then, the transition from $[\![\cdot]\!]_+$ to $[\![\cdot]\!]_\oplus$ can be achieved by trivially reducing the shares modulo 2π.

Then, a way to evaluate t on a secret shared input $[\![x]\!]_+ = (x_1, \ldots, x_n)$ is to convert $[\![x]\!]_+$ to $[\![x]\!]_\oplus$ and additively mask it with a shared masking $[\![\lambda]\!]_\oplus$, then reveal $x \oplus \lambda$ and rewrite our target $[\![e^{imx}]\!]_+$ as $e^{im(x\oplus\lambda)} \cdot [\![e^{im(-\lambda)}]\!]_+$. Indeed, since $x \oplus \lambda$ is revealed, the coefficient $e^{im(x\oplus\lambda)}$ can be computed in clear. Overall, the

whole trigonometric polynomial t can be evaluated in a single round of communication, given a precomputed triplet such as $([\![\lambda]\!]_{\oplus}, [\![e^{-i\lambda}]\!]_{+}, \ldots, [\![e^{-i\lambda P}]\!]_{+})$ and thanks to the fact that $x \oplus \lambda$ has been revealed.

Also, we notice that to work with complex numbers of absolute value 1 makes the method numerically stable, compared to power functions in regular polynomials. It is for this reason that the evaluation of trigonometric polynomials is a better solution in our context.

4.2 Approximating Non-periodic Functions

If one is interested in uniformly approximating (with trigonometric polynomials on a given interval, e.g. $[-\pi/2, \pi/2]$) a non-periodic function f, one cannot simply use the Fourier coefficients. Indeed, even if the function is analytic, its Fourier series need not converge uniformly near the end-points due to Gibbs phenomenon.

Approximations via C^∞-Extensions. One way to remedy this problem is to look for a periodic extension of the function to a larger interval and look at the convergence properties of the Fourier series for that extension. To obtain exponential convergence, the extension needs to be analytic too, a condition that can rarely be guaranteed. In other words, the classical Whitney extension theorem [29] will rarely yield an analytic extension that is periodic at the same time. A constructive approach for extending differentiable functions is given by Hestenes [21] and Fefferman [17] in a greater generality. The best one can hope for is to extend the function to a C^∞-function (which is not analytic). As explained in [9,10], such an extension yields a super-algebraic approximation at best that is not exponential.

Least-Square Approximations. An alternative approach for approximating a non-periodic function with a trigonometric functions is to search for these functions on a larger interval (say $[-\pi, \pi]$), such that the restriction (to the original interval) of the L^2-distance between the original function and the approximation is minimized. This method was first proposed by [8], but it was observed that the coefficients with respect to the standard Fourier basis were numerically unstable in the sense that they diverge (for the optimal solution) as one increases the number of basis functions. The method of [22] allows to remedy this problem by using a different orthonormal basis of certain half-range Chebyshev polynomials of first and second kind for which the coefficients of the optimal solution become numerically stable. In addition, one is able to calculate numerically these coefficients using a Gaussian quadrature rule. More details are given in [7, Appendix C].

Approximating the Sigmoid Function. We now restrict to the case of the sigmoid function over the interval $[-B/2, B/2]$ for some $B > 0$. We can rescale the variable to approximate $g(x) = \texttt{sigmo}(Bx/\pi)$ over $[-\pi/2, \pi/2]$. If we extend g by anti-periodicity (odd-even) to the interval $[\pi/2, 3\pi/2]$ with the mirror condition $g(x) = g(\pi - x)$, we obtain a continuous 2π-periodic piecewise C^1 function.

By Dirichlet's global theorem, the Fourier serie of g converges uniformly over $\mathbb{R}$, so for all $\varepsilon > 0$, there exists a degree N and a trigonometric polynomial g_N such that $\|g_N - g\|_\infty \leq \epsilon$. To compute $\texttt{sigmo}(t)$ over secret shared t, we first apply the affine change of variable (which is easy to evaluate in MPC), to get the corresponding $x \in [-\pi/2, \pi/2]$, and then we evaluate the trigonometric polynomial $g_N(x)$ using a Fourier triplet. This method is sufficient to get 24 bits of precision with a polynomial of only 10 terms, however asymptotically, the convergence rate is only in $\Theta(n^{-2})$ due to discontinuities in the derivative of g. In other words, approximating g with λ bits of precision requires to evaluate a trigonometric polynomial of degree $2^{\lambda/2}$. Luckily, in the special case of the sigmoid function, we can make this degree polynomial by explicitly constructing a 2π-periodic analytic function that is exponentially close to the rescaled sigmoid on the whole interval $[-\pi, \pi]$ (not the half interval). Besides, the geometric decay of the coefficients of the trigonometric polynomial ensures perfect numerical stability. The following theorem, whose proof can be found in [7, Appendix D] summarizes this construction.

Theorem 1. *Let $h_\alpha(x) = 1/(1+e^{-\alpha x}) - x/2\pi$ for $x \in (-\pi, \pi)$. For every $\varepsilon > 0$, there exists $\alpha = O(\log(1/\varepsilon))$ such that h_α is at uniform distance $\varepsilon/2$ from a 2π-periodic analytic function g. There exists $N = O(\log^2(1/\varepsilon))$ such that the Nth term of the Fourier series of g is at distance $\varepsilon/2$ of g, and thus, at distance $\leq \varepsilon$ from h_α.*

5 Honest But Curious Model

In the previous sections, we defined the shares of multiplication, power and Fourier triplets, but did not explain how to generate them. Of course, a single *trusted dealer* approved by all players (TD model) could generate and distribute all the necessary shares to the players. Since the trusted dealer knows all the masks, and thus all the data, the TD model is only legitimate for few computational outsourcing scenarios.

We now explain how to generate the same triplets efficiently in the more traditional *honest-but-curious* (HBC) model. To do so, we keep an external entity, called again the dealer, who participates in an interactive protocol to generate the triplets, but sees only masked information. Since the triplets in both the HBC and TD models are similar, the online phase is unchanged. Notice that in the HBC model, even if the dealer does not have access to the secret shares, he still has more power than the players. In fact, if one of the players wants to gain information on the secret data, he has to collude with all other players, whereas the dealer would need to collaborate with just one of them.

In what follows, we suppose that during the offline phase, a private channel exists between each player and the dealer. In the case of an HBC dealer, we also assume that an additional private broadcast channel (a channel to which the dealer has no access) exists between all the players (Fig. 5 in [7, Appendix 5]). Afterwards, the online phase only requires a public broadcast channel between the players (Fig. 5 in [7, Appendix 5]). In practice, because of the underlying

Algorithm 1. Honest but curious triplets generation for a trigonometric poly

Output: Shares $([\![\lambda]\!], [\![e^{im_1\lambda}]\!]_+, \ldots, [\![e^{im_N\lambda}]\!]_+)$.

1: Each player P_i generates λ_i, a_i (uniformly modulo 2π)
2: Each player P_i broadcasts a_i to all other players.
3: Each player computes $a = a_1 + \cdots + a_n \mod 2\pi$.
4: Each player P_i sends to the dealer $\lambda_i + a_i \mod 2\pi$.
5: The dealer computes $\lambda + a \mod 2\pi$ and $w^{(1)} = e^{im_1(\lambda+a)}, \ldots, w^{(N)} = e^{im_N(\lambda+a)}$
6: The dealer creates $[\![w^{(1)}]\!]_+, \ldots, [\![w^{(N)}]\!]_+$ and sends $w_i^{(1)}, \ldots, w_i^{(N)}$ to player P_i.
7: Each player P_i multiplies each $w_i^{(j)}$ by $e^{-im_j a}$ to get $(e^{im_j\lambda})_i$, for all $j \in [1, N]$.

encryption, private channels (e.g., SSL connections) have a lower throughput (generally ≈20 MB/s) than public channels (plain TCP connections, generally from 100 to 1000 MB/s between cloud instances).

The majority of the honest-but-curious protocols proposed in the literature present a scenario with only 2 players. In [3] and [12], the authors describe efficient HBC protocols that can be used to perform a fast MPC multiplication in a model with three players. The two schemes assume that the parties follow correctly the protocol and that two players do not collude. The scheme proposed in [12] is very complex to scale for more than three parties, while the protocol in [3] can be extended to a generic number of players, but requires a quadratic number of private channels (one for every pair of players). We propose a different protocol for generating the multiplicative triplets in the HBC scenario that is efficient for an arbitrary number n of players. In our scheme, the dealer evaluates the non-linear parts in the triplet generation, over the masked data produced by the players, then he distributes the masked shares. The mask is common to all players, and it is produced thanks to the private broadcast channel that they share. Finally, each player produces his triplet by unmasking the precomputed data received from the dealer.

In [7, Appendix E], we present in detail two algorithms: one for the generation of multiplicative Beaver's triplets (Algorithm 4) and the other for the generation of the triplets used in the computation of the power function (Algorithm 5), both in the honest-but-curious (HBC) model.

Following the same ideas, Algorithm 1 describes our triplets generation for the evaluation of a trigonometric polynomial in the honest-but-curious scenario.

6 Application to Logistic Regression

In a classification problem one is given a data set, also called a training set, that we will represent here by a matrix $X \in \mathcal{M}_{N,k}(\mathbb{R})$, and a training vector $\mathbf{y} \in \{0,1\}^N$. The data set consists of N input vectors of k features each, and the coordinate y_i of the vector $\mathbf{y}$ corresponds to the class (0 or 1) to which the i-th element of the data set belongs to. Formally, the goal is to determine a function $h_\theta : \mathbb{R}^k \to \{0,1\}$ that takes as input a vector $\mathbf{x}$, containing k features, and which outputs $h_\theta(\mathbf{x})$ predicting reasonably well y, the corresponding output value.

In logistic regression typically one uses hypothesis functions $h_\theta : \mathbb{R}^{k+1} \to [0,1]$ of the form $h_\theta(\mathbf{x}) = \texttt{sigmo}(\theta^T \mathbf{x})$, where $\theta^T \mathbf{x} = \sum_{i=0}^{k} \theta_i x_i \in \mathbb{R}$ and $x_0 = 1$. The vector θ, also called *model*, is the parameter that needs to be determined. For this, a convex *cost function* $C_{\mathbf{x},y}(\theta)$ measuring the quality of the model at a data point $(\mathbf{x}, y)$ is defined as

$$C_{\mathbf{x},y}(\theta) = -y \log h_\theta(\mathbf{x}) - (1-y) \log(1 - h_\theta(\mathbf{x})).$$

The cost for the whole dataset is thus computed as $\sum_{i=1}^{N} C_{\mathbf{x_i},y_i}(\theta)$. The overall goal is to determine a model θ whose cost function is as close to 0 as possible. A common method to achieve this is the so called *gradient descent* which consists of constantly updating the model θ as

$$\theta := \theta - \alpha \nabla C_{\mathbf{x},y}(\theta),$$

where $\nabla C_{\mathbf{x},y}(\theta)$ is the gradient of the cost function and $\alpha > 0$ is a constant called the *learning rate*. Choosing the optimal α depends largely on the quality of the dataset: if α is too large, the method may diverge, and if α is too small, a very large number of iterations are needed to reach the minimum. Unfortunately, tuning this parameter requires either to reveal information on the data, or to have access to a public fake training set, which is not always feasible in private MPC computations. This step is often silently ignored in the literature. Similarly, preprocessing techniques such as feature scaling, or orthogonalization techniques can improve the dataset, and allow to increase the learning rate significantly. But again, these techniques cannot easily be implemented when the input data is shared, and when correlation information should remain private. In this work, we choose to implement the IRLS method [5, Sect. 4.3], which does not require feature scaling, works with learning rate 1, and converges in much less iterations, provided that we have enough floating point precision. In this case, the model is updated as:

$$\theta := \theta - H(\theta)^{-1} \cdot \nabla C_{\mathbf{x},y}(\theta),$$

where $H(\theta)$ is the Hessian matrix.

6.1 Implementation and Experimental Results

We implemented an MPC proof-of-concept of the logistic regression algorithm in C++. We represented numbers in $\mathcal{C}(B,p)$ classes with 128-bit floating point numbers, and set the masking security parameter to $\tau = 40$ bits. Since a 128-bit number has 113 bits of precision, and the multiplication algorithm needs $2\tau = 80$ bits of masking, we still have 33 bits of precision that we can freely use throughout the computation. Since our benchmarks are performed on a regular x86_64 CPU, 128-bit floating point arithmetic is emulated using GCC's quadmath library, however additional speed-ups could be achieved on more recent hardware that natively supports these operations (eg. IBM's next POWER9 processor). In our proof of concept, our main focus was to improve the running time, the floating point precision, and the communication complexity of

Algorithm 2. Model training: $\mathsf{Train}(X, \mathbf{y})$

Input: A dataset $X \in \mathcal{M}_{N,k}(\mathbb{R})$ and a training vector $\mathbf{y} \in \{0,1\}^N$
Output: The model $\theta \in \mathbb{R}^k$ that minimizes $\mathsf{Cost}_{X,\mathbf{y}}(\theta)$

1: Precompute $\text{Prods}_i = X_i^T X_i$ for $i \in [0, N-1]$
2: $\theta \leftarrow [0, \ldots, 0] \in \mathbb{R}^k$
3: **for** iter $= 1$ to IRLS_ITERS **do** ▷ In practice IRLS_ITERS $= 8$
4: $\quad \mathbf{a} \leftarrow X \cdot \theta$
5: $\quad \mathbf{p} \leftarrow [\texttt{sigmo}(a_0), \ldots, \texttt{sigmo}(a_{N-1})]$
6: $\quad \text{pmp} \leftarrow [p_0(1-p_0), \ldots, p_{N-1}(1-p_{N-1})]$
7: $\quad \text{grad} \leftarrow X^T(\mathbf{p} - \mathbf{y})$
8: $\quad H \leftarrow \text{pmp} \cdot \text{Prods}$
9: $\quad \theta = \theta - H^{-1} \cdot \text{grad}$
10: **end for**
11: return θ

Model-training algorithm with the IRLS method. The algorithm is explained over the plaintext. In the MPC instantiation, each player gets a secret share for each variables. Every product is evaluated using the bilinear formula of Section 2, and the sigmoid using the Fourier method of Section 4.

the online phase, so we implemented the offline phase only for the trusted dealer scenario, leaving the honest but curious dealer variant as a future work.

We implemented the logistic regression model training described in Algorithm 2. Each iteration of the main loop evaluates the gradient (grad) and the Hessian (H) of the cost function at the current position θ, and solves the Hessian system (line 7) to find the next position. Most of the computation steps are bilinear on large matrices or vectors, and each of them is evaluated via a Beaver triplet in a single round of communication. In step 5 the sigmoid functions are approximated (in parallel) by an odd trigonometric polynomial of degree 23, which provides 20 bits of precision on the whole interval. We therefore use a vector of Fourier triplets, as described in Sect. 4. The Hessian system (step 9) is masked by two (uniformly random) orthonormal matrices on the left and the right, and revealed, so the resolution can be done in plaintext. Although this method reveals the norm of the gradient (which is predictable anyway), it hides its direction entirely, which is enough to ensure that the final model remains private. Finally, since the input data is not necessarily feature-scaled, it is essential to start from the zero position (step 2) and not a random position, because the first one is guaranteed to be in the IRLS convergence domain.

To build the MPC evaluation of Algorithm 2, we wrote a small compiler to preprocess this high level listing, unroll all for loops, and turn it into a sequence of instructions on immutable variables (which are read-only once they are affected). More importantly, the compiler associates a single additive mask λ_U to each of these immutable variables U. This solves two important problems that we saw in the previous sections: first, the masking information for huge matrices that are re-used throughout the algorithm are transmitted only once during the whole protocol (this optimization already appears in [26], and in our case, it has a

huge impact for the constant input matrix, and their precomputed products, which are re-used in all IRLS iterations). It also mitigates the attack that would retrieve information by averaging its masked distribution, because an attacker never gets two samples of the same distribution. This justifies the choice of 40 bits of security for masking.

During the offline phase, the trusted dealer generates one random mask value for each immutable variable, and secret shares these masks. For all matrix-vector or matrix-matrix products between any two immutable variables U and V (coming from lines 1, 4, 6, 7 and 8 of Algorithm 2), the trusted dealer also generates a specific multiplication triplet using the masks λ_U of U and λ_V of V. More precisely, he generates and distributes additive shares for $\lambda_U \cdot \lambda_V$ as well as integer vectors/matrices of the same dimensions as the product for the share-reduction phase. These integer coefficients are taken modulo 256 for efficiency reasons.

6.2 Results

We implemented all the described algorithms and we tested our code for two and three parties, using cloud instances on both the AWS and the Azure platforms, having Xeon E5-2666 v3 processors. In our application each instance communicates via its public IP address. Furthermore, we use the zeroMQ library to handle low-level communications between the players (peer-to-peer, broadcast, central nodes etc.).

In the results that are provided in Table 1 in Appendix A, we fixed the number of IRLS iterations to 8, which is enough to reach a perfect convergence for most datasets, and we experimentally verified that the MPC computation outputs the same model as the one with plaintext iterations. We see that for the datasets of 150000 points, the total running time of the online phase ranges from 1 to 5 min. This running time is mostly due to the use of emulated quad-float arithmetic, and this MPC computation is no more than 20 times slower than the plaintext logistic regression on the same datasets, if we implement it using the same 128-bit floats (yet, of course, the native double-precision version is much faster). More interestingly, we see that the overall size of the totality of the triplets and the amount of online communications are small: for instance, a logistic regression on 150000 points with 8 features requires only 756 MB of triplets per player, and out of it, only 205 MB of data are broadcasted during the online phase per player. This is due to the fact that a Fourier triplet is much larger than the value that is masked and exchanged. Because of this, the communication time is insignificant compared to the whole running time, even with regular WAN bandwidth.

Finally, when the input data is guaranteed to be feature-scaled, we can improve the whole time, memory and communication complexities by about 30% by performing 3 classical gradient descent iterations followed by 5 IRLS iterations instead of 8 IRLS iterations. We tested this optimization for both the plaintext and the MPC version and in Appendix A, we show the evolution of the cost function, during the logistic regression, and of the F-score (Figs. 1 and 2), depending on the method used.

We have tested our platform on datasets that were provided by the banking industry. For privacy reasons, these datasets cannot be revealed. However, the behaviour described in this paper can be reproduced by generating random data sets, for instance, with Gaussian distribution, setting the acceptance threshold to 0.5%, and adding some noise by randomly swapping a few labels. Readers interested in testing the platform should contact the authors.

Open Problems. A first important open question is the indistinguishability of the distributions after our noise reduction algorithm. On a more fundamental level, one would like to find a method of masking using the basis of half-range Chebyshev polynomials defined in the appendix as opposed to the standard Fourier basis. Such a method, together with the exponential approximation, would allow us to evaluate (in MPC) any function in $L^2([-1, 1])$.

Acknowledgements. We thank Hunter Brooks, Daniel Kressner and Marco Picasso for useful conversations on data-independent iterative optimization algorithms. We are grateful to Jordan Brandt, Alexandre Duc and Morten Dahl for various useful discussions regarding multi-party computations and privacy-preserving machine learning.

A Timings for $n = 3$ players

We present in this section a table (Table 1) summarizing the different measures we obtained during our experiments for $n = 3$ players. For this we considered datasets containing from 10000 to 1500000 points having 8, 12 or 20 features each.

Figure 1 shows the evolution of the cost function during the logistic regression as a function of the number of iterations, on a test dataset of 150000 samples, with 8 features and an acceptance rate of 0.5%. In yellow is the standard gradient descent with optimal learning rate, in red, the gradient descent using the piecewise linear approximation of the sigmoid function (as in [26]), and in green, our MPC model (based on the IRLS method). The MPC IRLS method (as well as the plaintext IRLS) method converge in less than 8 iterations, against 500 iterations for the standard gradient method. As expected, the approx method does not reach the minimal cost.

Figure 2 shows the evolution of the F-score during the same logistic regression as a function of the number of iterations. The standard gradient descent and our MPC produce the same model, with a limit F-score of 0.64. However, no positive samples are detected by the piecewise linear approximation, leading to a null F-score. However, in the three cases, the accuracy (purple) is nearly 100% from the first iteration.

Table 1. Summary of the different measures (time, amount of exchanged data and amount of precomputed data) for $n = 3$ players.

Dataset size N	# features k	Communication size (MB)	Precomputed data size (MB)	Time (sec)	Time (sec)
				Offline phase	Online phase
10000	8	13.75	50.55	20.07	6.51
10000	12	21.88	66.18	26.6	9.81
10000	20	45.97	113.1	46.26	19.83
25000	8	34.2	126.23	51.59	19.24
25000	12	54.52	165.14	68.14	24.7
25000	20	114.5	281.98	115.56	48.8
50000	8	68.53	252.35	103.41	32.89
50000	12	108.93	330.1	135.07	49.99
50000	20	228.7	563.46	229.17	103.3
100000	8	137	504.6	205.53	67.11
100000	12	217.75	659.96	269.04	99.99
100000	20	457.1	1126.41	457.33	205.28
150000	8	205.48	756.84	308.14	101.36
150000	12	326.56	989.83	343.86	152.41
150000	20	685.51	1689.36	685.74	314.4

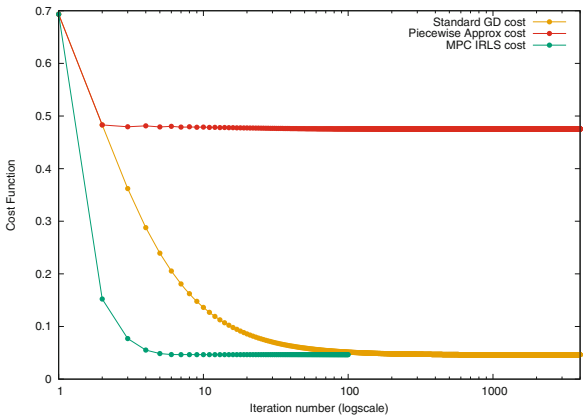


Fig. 1. Evolution of the cost function depending on the method

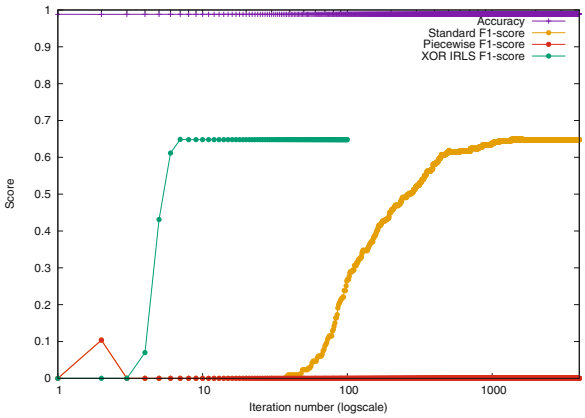


Fig. 2. Evolution of the F-score depending on the method

References

1. Abadi, M., et al.: Deep learning with differential privacy. CoRR, abs/1607.00133 (2016)
2. Aono, Y., Hayashi, T., Trieu Phong, L., Wang, L.: Privacy-preserving logistic regression with distributed data sources via homomorphic encryption. IEICE Trans. **99-D**(8):2079–2089 (2016)
3. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 805–817 (2016)
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
5. Björck, A.: Numerical Methods for Least Squares Problems. SIAM, Philadelphia (1996)
6. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88313-5_13
7. Boura, C., Chillotti, I., Gama, N., Jetchev, D., Peceny, S., Petric, A.: High-precision privacy-preserving real-valued function evaluation. Cryptology ePrint Archive, Report 2017/1234 (2017). https://eprint.iacr.org/2017/1234
8. Boyd, J.: A comparison of numerical algorithms for Fourier extension of the first, second, and third kinds. J. Comput. Phys. **178**(1), 118–160 (2002)
9. Boyd, J.: Fourier embedded domain methods: extending a function defined on an irregular region to a rectangle so that the extension is spatially periodic and c^{∞}. Appl. Math. Comput. **161**(2), 591–597 (2005)
10. Boyd, J.: Asymptotic fourier coefficients for a C infinity bell (smoothed-"top-hat") & the fourier extension problem. J. Sci. Comput. **29**(1), 1–24 (2006)

11. Chaudhuri, K., Monteleoni, C.: Privacy-preserving logistic regression. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, 8–11 December 2008, vol. 21, pp. 289–296. Curran Associates Inc. (2008)
12. Cramer, R., Damgård, I., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press, Cambridge (2015)
13. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
14. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: SPDZ Software. https://www.cs.bris.ac.uk/Research/CryptographySecurity/SPDZ/
15. Dataset, Arcene Data Set. https://archive.ics.uci.edu/ml/datasets/Arcene
16. Dataset, MNIST Database. http://yann.lecun.com/exdb/mnist/
17. Fefferman, C.: Interpolation and extrapolation of smooth functions by linear operators. Rev. Mat. Iberoamericana **21**(1), 313–348 (2005)
18. Gascón, A., et al.: Privacy-preserving distributed linear regression on high-dimensional data. Proc. Priv. Enhancing Technol. **4**, 248–267 (2017)
19. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, 19–24 June 2016, pp. 201–210 (2016)
20. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
21. Hestenes, M.R.: Extension of the range of a differentiable function. Duke Math. J. **8**, 183–192 (1941)
22. Huybrechs, D.: On the Fourier extension of nonperiodic functions. SIAM J. Numer. Anal. **47**(6), 4326–4355 (2010)
23. Jäschke, A., Armknecht, F.: Accelerating homomorphic computations on rational numbers. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 405–423. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_22
24. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 36–54. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_3
25. Livni, R., Shalev-Shwartz, S., Shamir, O.: On the computational efficiency of training neural networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems 2014, Montreal, Quebec, Canada, 8–13 December 2014, vol. 27, pp. 855–863 (2014)
26. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 19–38. IEEE Computer Society (2017)
27. Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, 19–22 May 2013, pp. 334–348. IEEE Computer Society (2013)

28. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning: revisited and enhanced. In: Batten, L., Kim, D.S., Zhang, X., Li, G. (eds.) ATIS 2017. CCIS, vol. 719, pp. 100–110. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-5421-1_9
29. Whitney, H.: Analytic extensions of differentiable functions defined in closed sets. Trans. Am. Math. Soc. **36**(1), 63–89 (1934)
30. Wu, S., Teruya, T., Kawamoto, J., Sakuma, J., Kikuchi, H.: Privacy-preservation for stochastic gradient descent application to secure logistic regression. In: The 27th Annual Conference of the Japanese Society for Artificial Intelligence, vol. 27, pp. 1–4 (2013)

Faster Unbalanced Private Set Intersection

Amanda C. Davi Resende(✉) and Diego F. Aranha

Institute of Computing, University of Campinas (UNICAMP), Campinas, Brazil
{amanda.resende,dfaranha}@ic.unicamp.br

Abstract. Protocols for Private Set Intersection (PSI) are important cryptographic primitives that perform joint operations on datasets in a privacy-preserving way. They allow two parties to compute the intersection of their private sets without revealing any additional information beyond the intersection itself. Unfortunately, PSI implementations in the literature do not usually employ the best possible cryptographic implementation techniques. This results in protocols presenting computational and communication complexities that are prohibitive, particularly in the case when one of the participants is a low-powered device and there are bandwidth restrictions. This paper builds on modern cryptographic engineering techniques and proposes optimizations for a promising one-way PSI protocol based on public-key cryptography. For the case when one of the parties holds a set much smaller than the other (a realistic assumption in many scenarios) we show that our improvements and optimizations yield a protocol that outperforms the communication complexity and the run time of previous proposals by around one thousand times.

Keywords: Cuckoo filter · Private set intersection · Unbalanced PSI
Software implementation

1 Introduction

Private Set Intersection (PSI) is a special case of secure multiparty computation (MPC) where two parties perform joint operations on datasets while preserving privacy. They have been used in several applications such as genetic testing of fully-sequenced human genomes [4], private contact discovery [7], relationship path discovery in social networks [28], botnet detection [29] and proximity testing [32].

PSI protocols allow two parties storing a set of private data such as lists of patients, criminal suspects or telephone contacts to compute the intersection of their sets without revealing any additional information beyond the intersection to one or both parties. These protocols can be divided into one-way PSI, i.e., only one of the parties learns the intersection; or mutual PSI (mPSI), in which both parties learn the intersection. The focus of this work is one-way PSI protocols. For more information about mPSI, the reader is invited to check [6,8,24].

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 203–221, 2018.
https://doi.org/10.1007/978-3-662-58387-6_11

PSI protocols can also be classified based on their set sizes. In the literature, Chen *et al.* [7] defined the PSI setting as symmetric when the sets have approximately the same size, and asymmetric when one of the sets is substantially smaller than the other. We propose a new terminology to prevent confusion with the type of primitive being used (symmetric or asymmetric): *balanced* for sets with approximately the same size and *unbalanced* for the opposite scenario[1].

However, even with several PSI protocols proposed in the literature, most real-word applications use naive solutions (as later detailed in Sect. 2.2). A reason for this is that some solutions are efficient when performing operations on small datasets, but become impractical for large sets. They may also be efficient in terms of execution time, but when performed in constrained environments with low bandwidth became impractical as they transmit too much data.

Protocols proposed and implemented in several papers by Pinkas *et al.* [36, 38, 39] are efficient in terms of computation (by using mostly symmetric operations), but need to transmit a lot of data, while other works based on public-key cryptography [4, 7, 19, 27] need to transmit fewer data, but require less efficient operations. Thus, the choice of the protocol depends on the PSI setting, network bandwidth, storage space, security properties, among other factors.

1.1 Our Contributions

Several of the PSI implementations available in the literature makes no use of modern and efficient techniques for the implementation of cryptographic protocols, mainly based on public-key cryptography. We aim at filling this gap by showing that the protocol previously proposed by Baldi *et al.* in [4] can be optimized as to reduce its communication and running time by a factor of at least three. Our implementation is available online at http://github.com/amandadavi7/PSI. In more detail, the main contributions of this paper are:

- **Improvements on the one-way PSI protocol based on public-key cryptography by Baldi et al. [4]:** We show that the version of the protocol secure against semi-honest adversaries becomes an efficient and practical one-way PSI for the unbalanced setting after our optimizations[2]. Moreover, it satisfies a desired forward secrecy property on the client side, usually more vulnerable than the server, which guarantees that elements exchanged in the past will remain confidential even if long-term secrets (keys) are exposed.
- **We propose Cuckoo filters to reduce the amount of data to be exchanged by the protocol and stored by the client:** Cuckoo filters present many advantages: (i) they require less storage space than other similar approaches, like Bloom filter and Cuckoo hashing, for a false positive rate (FPR) less than 3% [11]; (ii) they allow the delete operation (important in

[1] Throughout this paper, the client set is always the smaller one.

[2] In constrained scenarios, like 1 Mbps of network bandwidth, our optimized protocol remains a good choice for balanced one-way PSI. See Table 4 in the full version of this paper [40].

some applications); and (iii) the lookup operation is performed in linear time in the number of entries per bucket. To the best of our knowledge, this is the first time that a Cuckoo filter is employed in PSI protocols, where normally a Bloom filter is used.

- **We provide an efficient software implementation of the protocols using the Galbraith-Lin-Scott binary elliptic curve (GLS-254) with point compression:** To the best of our knowledge, this is the first time that a state-of-the-art implementation of elliptic curves is used to instantiate PSI protocols that rely on this type of operation. Our implementation of the GLS-254 curve takes around 50,000 cycles to compute an exponentiation, which is 24× faster than the 283-bit Koblitz curve implementation used, for example, in the PSI protocol presented in [37].
- **Experimental comparison:** We implemented the original version [4] and our optimized protocol (both using the GLS-254 curve) and compared them with the most promising PSI protocols in the literature, showing the results of the (offline) preprocessing phase (when it is possible) and the online phase. Our results show that with our optimizations, this protocol is efficient even when used in bandwidth restricted scenarios.

1.2 Application to Private Contact Discovery

In the private contact discovery problem, a user signs up to a messaging application such as WhatsApp, Signal or Telegram, and would like to discover which contacts in his/her address book are also registered. However, the user is not willing to reveal his entire list of contacts. In this setting, the user typically has a set with a few hundred contacts, while the messaging service can have from a few million to a few billion users, characterizing the unbalanced setting.

Because of the sheer number of entries in the social network server's side, secure messaging applications such as TextSecure/Signal[3] and Secret[4], currently employ naive approaches (see Sect. 2.2) to "solve" the private contact discovery problem, since they have both better run time and communication complexity when compared to state-of-the-art secure protocols. Signal is also experimenting with the Intel SGX, a trusted execution environment, in order to improve the security of private contact discovery[5].

At the cost of tolerating a small FPR, our optimized protocol provides a secure solution that works in this realistic scenario, being potentially useful for social networks with millions of users.

Organization. This paper is organized as follows. In Sect. 2, we define notation and terminology used during the development of this work, a classification of PSI protocols into categories and a brief overview of the main protocols in each

[3] https://signal.org/blog/contact-discovery/.
[4] https://medium.com/@davidbyttow/demystifying-secret-12ab82fda29f#.5433o6e8h.
[5] https://signal.org/blog/private-contact-discovery/.

class. In Sects. 3 and 4, the basic protocol is presented and the optimizations are proposed, respectively. In Sect. 5 we describe the experimental results and compare them with the most promising protocols from the literature. Finally, in Sect. 6 we present our conclusions.

2 Related Work for PSI Protocols

We start by formalizing the notation used throughout the paper and other relevant definitions.

2.1 Notation and Terminology

- P_1 and P_2 are the participating parties in the protocols, where P_1 is the server and P_2 the client, except when referring to server-aided (third party) PSI protocols. X and Y are the respective input sets of P_1 and P_2, with size $n_1 = |X|$ and $n_2 = |Y|$. The set X is denoted by $\{x_1, x_2, ..., x_{n_1}\}$ and the set Y by $\{y_1, y_2, ..., y_{n_2}\}$ where each element has bit-length σ.
- For a set S, the notation $x \xleftarrow{R} S$ indicates that x was sampled from S with uniform distribution.
- The operation $a \stackrel{?}{=} b$ denotes the comparison whether a is equal or not to b.
- $\kappa = 128$ is the symmetric security parameter.
- $\rho = 40$ is the statistical security parameter (hashing failure).
- $\varphi = 256$ is the size of the representation of a point in the GLS-254 binary elliptic curve when using point compression (number of bits to store one x-coordinate and two trace bits).
- $\eta = 30$ is the hash collision parameter, i.e., the probability of a hash collision occurring is $< 2^{-30}$.
- $\mathbb{G}$ is a multiplicative group of prime order q.
- $H : \{0,1\}^* \rightarrow \{0,1\}^l$, $H_1 : \{0,1\}^\sigma \rightarrow \mathbb{G}$, $H_2 : \mathbb{G} \rightarrow \{0,1\}^l$ are hash functions modeled as random oracles in the security analysis. In some cases, the output length is defined as $l = \rho + \log n_1 + \log n_2$ bits ($l' = \lceil l/8 \rceil$ bytes), as suggested by Pinkas *et al.* [39], instead of $2 \cdot \kappa$. This produces the collision probability $2^{-\eta}$, which is suitable for most applications.
- For the Cuckoo filter, we also define v as the fingerprint length (in bits), w as the load factor ($0 \leq w \leq 1$), b as the number of entries per buckets, m as the number of buckets, $\epsilon_{max} = 1 - (1 - \frac{1}{2^v})^{2b}$ as the upper bound on the false positive rate (FPR) and ϵ as the observed FPR, both given in %.

2.2 Classification and Related Work of PSI Protocols

Many one-way PSI protocols have been proposed in the open research literature [7,9,15,18,21,36,38,39]. They are constructed based on several primitives such as Bloom filters [5], Homomorphic Encryption [12,16], Oblivious Pseudorandom Function (OPRF) [14], Unpredictable Function [21], Oblivious Transfer (OT) [25,31], Oblivious Polynomial Evaluation (OPE) [30], Cuckoo hashing [35],

Garbled Circuits (GC) [41,42], among others. Following Pinkas *et al.* [36,38,39], PSI protocols can be classified into: naive hashing (or naive solution), server-aided PSI (or third-party based PSI), PSI based on generic protocols (or circuit-based PSI), OT-based PSI and PSI based on public-key cryptography.

Naive Hashing. Both P_1 and P_2 use a hash function H to compute the hash of their elements. P_1 then computes $x'_i = H(x_i)$ while P_2 computes $y'_j = H(y_j)$, where $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$. After, P_1 sends values x'_i to P_2 which computes the intersection by checking if $y'_j \stackrel{?}{=} x'_i$ for all values of i and j. This approach is very efficient in both run time and communication. However, if the hash function inputs were taken from a low-entropy domain $\mathbb{D}$, P_2 can discover all elements of P_1 by performing a brute-force attack. One solution could be to choose $\mathbb{D}$ with high entropy when possible. This would prevent the problem, but consecutive executions of the protocol would still leak repeated elements and would not guarantee forward secrecy since P_2 can verify if a specific element $z \in \mathbb{D}$ was part of the P_1 set, by just checking if $H(z) \stackrel{?}{=} x'_i$. Nonetheless, this naive protocol is employed by messaging applications for the private contact discovery problem, and social networks like Facebook[6] and Twitter[7] to measure advertisement conversion rates.

Server-Aided PSI. Several works in the literature [8,9,22] have employed a third party, in this case, called as server, to achieve better performance in PSI protocols. The server can be semi-honest (cannot deviate from protocol, learns only by observing communication between parties), covert (if it deviates from protocol, it is detected with some probability by an honest party) or malicious (can arbitrarily deviate from the protocol). In [22], Kamara *et al.* present a semi-honest server protocol that takes 10 min with 12 GB of communication and 100 threads to evaluate 2 sets of 1 billion of elements each. However, such protocols are secure only if the third party does not collude with any of the other parties, thus having a different security model from conventional protocols.

PSI Based on Generic Protocols. Generic secure computation uses arithmetic or Boolean circuits to securely evaluate functions, among them, the set intersection. In [18], Huang *et al.* presented several of these protocols using Boolean circuits, all of them constructed using Yao's garbled circuits [41,42]. The simplest protocol described in [18] involves the comparison of each element from P_1 with each from P_2. This approach is known as Pairwise-Comparison (PWC) and involves $O(n^2)$ comparisons, which does not scale well for large sets. Another more efficient approach presented in [18], the Sort-Compare-Shuffle (SCS) circuit, is more efficient, with complexity $O(n \log n)$. The major advantage of this type of protocol is that they can be easily adapted to any other features that PSI

[6] https://www.wired.com/2014/12/oracle-buys-data-collection-company-datalogix/.
[7] https://support.twitter.com/articles/20170410.

protocols may require, such as revealing only the intersection size or whether the size is larger or smaller than a threshold. However, despite the improvements in recent years, they still have a very high run time compared to others.

OT-Based PSI. This category of protocols is the most recent and, up to date the most promising, mainly because of the large performance improvements from OT extensions. The first protocol was proposed in 2013 by Dong *et al.* [9], combining Bloom filters and OT [20] in their construction.

In 2014, Pinkas *et al.* [38] presented improvements to [9] and also proposed a new and more efficient protocol combining OT and hashing. In 2015, they have shown [36] that their previously proposal [38] could be improved by using the permutation-based hashing technique [2], since it reduces the size of each element stored in the bins, which until then was the main overhead of the protocol. In 2016, Pinkas *et al.* [39] presented improvements for their earlier protocols, where the complexity no longer depends on the size of each element. This solution is the state of the art for balanced one-way PSI protocols and, depending on the scenario (network bandwidth), also for unbalanced PSI protocols with security against semi-honest adversaries. By using only symmetric operations in almost all of its construction, the solution is extremely efficient[8].

Public-Key Cryptography Based PSI. Meadows [27] and Huberman *et al.* [19] proposed the earliest PSI approaches based on public-key cryptography, even before the PSI problem was formally defined in [15]. Both were based on the commutative properties of the Diffie-Hellman (DH) key exchange. Later Jarecki *et al.* [21] presented a PSI protocol secure against malicious adversaries based on a Parallel Oblivious Unpredictable Function (POUF). In [4], Baldi *et al.* relaxed the security of [21] to semi-honest adversaries. Another PSI protocol was proposed by Chen *et al.* [7] and is based on the one presented by Pinkas *et al.* [39], but instead of performing OPRF (via OT) operations, it uses the Fan-Vercauteren (FV) leveled Fully Homomorphic Encryption (FHE) scheme [12]. This change considerably decreases the amount of data to be transmitted in the unbalanced setting. Therefore, depending on the setting and the network bandwidth, Chen *et al.* [7] is faster than [39]. The good performance is however restricted to 32-bit elements due to limitations in the parameters of the FHE scheme. The protocol proposed in [4] was used to obtain the results in this paper.

The most recent work was presented by Kiss *et al.* [23]. They independently noted that in some PSI protocols the server can perform operations on its data only once and send the result to the client, which will use them in future executions to compute the intersection. They proposed using a Bloom filter (or a counting Bloom filter) to decrease the amount of data to be transmitted or stored by the client. These observations are very important in an unbalanced

[8] The protocol presented in [39] uses asymmetric operations [31] to generate the OT bases. However, the cost of these operations is negligible when the number of elements evaluated is substantially greater than the value of κ.

setting since all operations and communication are only performed considering the smaller client set. In terms of security, there is an important limitation in their approach: in the protocol closest to our optimized proposal (DH-based PSI [19,27]), the client and server reuse the same keys across all executions, which does not provide forward secrecy. In terms of performance, during the preprocessing phase alone (the setup phase, as in the paper), the server should send $n_1\varphi$ bits to the client and the server and the client compute n_1 exponentiations. For example, if $n_1 = 2^{24}$ it will be necessary to transmit 568 MB, and to perform 2^{24} exponentiations on the server and client side.

One efficient way to instantiate PSI protocols based on public-key cryptography is to use elliptic curves. The exponentiation on elliptic curves becomes a scalar multiplication operation, but we will keep the exponentiation notation throughout the paper for compatibility with other works.

3 The Basic Protocol

Jarecki and Lui [21] presented a one-way PSI protocol secure against malicious adversaries based on the hardness of the One-More-Gap-Diffie-Hellman (OMGDH) problem and a Zero-Knowledge Proof (ZKP). Later, Baldi *et al.* [4] relaxed the security of this protocol to be secure against semi-honest adversaries, by removing the ZKP. This protocol is shown in Fig. 1 and works as follows: for each element $x_i \in X$, the server computes the hash $H_1(x_i)$, the exponentiation $H_1(x_i)^\alpha$ with the same exponent for all the elements, and again computes

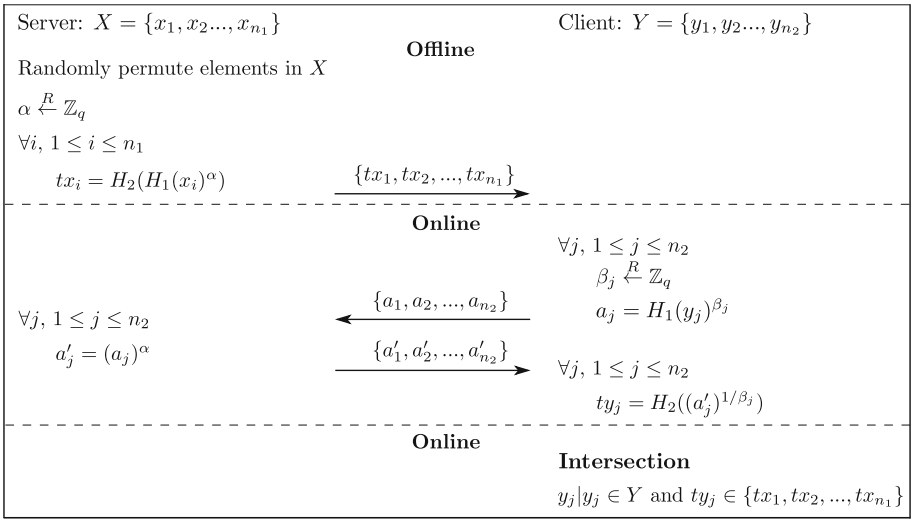


Fig. 1. Basic PSI protocol proposed in [4] that relaxes the security of the [21] to be secure against semi-honest adversaries. H_1 and H_2 are hash functions modeled as random oracles. [4, Adapted].

the hash $tx_i = H_2(H_1(x_i)^\alpha)$, sending values tx_i to the client. For each element $y_j \in Y$, the client computes the hash $H_1(y_j)$, the exponentiation $a_j = H_1(y_j)^{\beta_j}$ with ephemeral exponents β_j and sends values a_j to the server. The server computes $a'_j = (a_j)^\alpha$ for each a_j using the same α used previously and sends values a'_j to the client. The client then calculates $ty_j = H_2((a'_j)^{1/\beta_j})$, by "removing" the exponents that were applied earlier. Finally, the client computes the intersection by checking if $ty_j \in \{tx_1, tx_2, ..., tx_{n_1}\}$. The long-term secret is the server key α.

4 Optimizations

We propose a few modifications to the protocol presented in Sect. 3 that drastically improve its performance during the preprocessing and online phase.

(i) The offline phase is executed just once and the results stored in a database. We significantly reduce the size of the database by using a Cuckoo filter [11].
(ii) We implement the protocol based on the GLS-254 elliptic curve, which improves its computational performance.

Below these improvements are described in detail.

4.1 Generating the Database

As it can be seen in Fig. 1, the protocol is divided into two parts: offline and online. The offline part is executed without the need of any communication from the server to the client, except for any possible negotiation to define the initial parameters such as the group $\mathbb{G}$ and its order q. Thus, the server can mask all elements using α and the hash functions H_1 and H_2 ($tx_i = H_2(H_1(x_i)^\alpha)$) before receiving connections. Because of this feature, the offline part can be performed only once, where the server would compute the mask of each element and send them to the client, which would store them for use in each execution of the protocol. Therefore, only the online part needs to be used. The resulting protocol is very efficient when used in unbalanced PSI setting because in the online part all operations are performed only on the client elements ($3n_2$ asymmetric cryptographic operations and $2n_2\varphi$ bits are transmitted).

4.2 Reducing the Database Size

The database size increases as the server set grows. For example, assuming each masked server element has l bits and $\rho = 40$, as defined in Sect. 2.1, with $n_2 = 2^8$ and $n_1 = 2^{24}$, each masked element would have $l = 72$ bits. Since the server has 2^{24} elements, all server masked elements would occupy 144 MB. However, if the scale changes from a few million to a few billion, as is the case of a large messaging application with approximately 2^{30} users, the server masked elements would need 10 GB of space. Downloading and storing this data on devices with low memory resources, such as mobile devices, or with constrained network connection (low

bandwidth or/and high latency) can be prohibitive. To reduce the size of the data, techniques have been used previously in the literature such as Bloom filters and their variants [5,13], and Cuckoo hashing [35].

We take a different approach and propose to use Cuckoo filters [11]. They have clear advantages over Bloom filters and Cuckoo hashing, since they allow the delete operation (essential in private contact discovery), besides the insertion and lookup operations, using significantly less space than the Bloom filter variants and the Cuckoo hashing by storing only the element's fingerprint. For the two examples given above, a Cuckoo filter would use 48 MB and 3 GB, respectively[9]. For a reader not familiar with the concept and looking for more detail, we invite you to read [11,40]. To the best of our knowledge, this is the first application of Cuckoo filters to the problem of PSI.

4.3 Efficient Software Implementation of GLS-254 Elliptic Curve

Our implementation of ECC is based on the latest version of the GLS-254 software [33] available in SUPERCOP[10]. The binary GLS curve is a particularly efficient choice for our target platform due to its native support to binary field arithmetic, the lambda coordinate system [34] and the GLS endomorphism for fast scalar multiplications [17], achieving the current speed record for this operation. The code is structured in three layers: an efficient vectorized implementation of binary field arithmetic targeting Intel vector instruction sets; a regular window-based method for variable-base scalar multiplication implemented in constant time; a thin protocol layer implementing the DH key exchange. The exponentiations in our protocol were heavily based on the two last layers, while hashing and point compression were directly implemented over the field arithmetic available in the first layer.

The approach selected for hashing was a combination of the SHA256 hash function with the binary Shallue-van de Woestijne well-bounded encoding algorithm [1]. Elements are first hashed to a binary field element $u \in \mathbb{F}_{2^m}$ using SHA256, and then the encoding outputs the lambda coordinates (x, λ) of a point over the binary elliptic curve. This approach requires only a single inversion, a quadratic equation solution and some cheaper field operations, and provides better statistical properties than popular try-and-increment heuristics. Point compression adapts a rather classical technique [26]. The λ coordinate defined over a quadratic extension $\mathbb{F}_{2^{2m}}[s]/(s^2 + s + 1)$ as $(\lambda_0 + \lambda_1 s)$ is compressed to a pair of trace values $(Tr(\lambda_0), Tr(\lambda_1))$, which can later be used to solve a quadratic equation and disambiguate among the four possible solutions. In total, 256 bits are used by concatenating the 254 bits of the x-coordinate with the two trace bits. Decompression again requires a field inversion, solving a quadratic equation and some cheaper binary field operations. Our entire code runs in constant time for side-channel resistance, including the quadratic solver [1].

[9] These values may change if the FPR changes. Here we set $\epsilon_{max} = 0.009155\%$.
[10] https://bench.cr.yp.to.

4.4 Our Optimized Protocol

Figure 2 presents our optimized protocol based on Baldi *et al.* [4]. In the first part (offline), the server generates a Cuckoo filter, from his/her masked elements, using the insertion operation (*CF.Insert*), and sends the filter (*CF*) to the client. The online part is divided in two: in the first, client and server interact in order to mask the elements of the client. In the second, the client with his/her masked elements, checks if each one of them belongs to the filter, through the lookup operation (*CF.Check*), thus computing the set intersection. The last part of the protocol is the step of updating the filter. The server has a set of elements Z that he/she would like to insert, such as new users of a messaging application; or to delete, in the case where some users no longer use the service. In both cases, the server masks each element $z_k \in Z$ using α and sends them to the client. Along with these values, a variable is also sent to tell the client what is the type of update, if it is an insertion or a deletion.

In the insertion operation, the user must first check the load factor w of the filter. If it is greater than 0.95, the user must request the server to generate a new filter using all the elements, as in the first part of the protocol. Otherwise, the client inserts the element in the filter *CF*. The value 0.95 was set in [11] to be the highest w for the filter to have high space and lookup efficiency. After that, it is hard to insert elements without errors. Therefore, when w is greater than 0.95, a new and larger filter must be generated. In the case of deletion, the element is removed from the filter without the need to generate a new one.

4.5 Correctness and Security Guarantees with Our Modifications

The correctness guarantees of the protocol with the modifications shown above follow from the correctness of the original protocol by Jarecki and Lui [21] and Baldi *et al.* [4] and the correctness of the Cuckoo filter (up to false positives) [10,11]. This happens due to the fact that the same messages are exchanged and the same computational steps are performed by both parties, with the only difference that the server's masked elements are encoded in a Cuckoo filter.

The FPR of the Cuckoo filter can be as small as the application requires considering the cost of increasing the filter size. We have 2 different FPR: ϵ_{max} and ϵ. The first is an upper bound and does not take into account the load factor of the filter, and the second is the observed measure that takes into account the load factor (for more details see the full version of this paper [40]).

The security guarantees of the modified protocol also follow from the security of the original protocol, which is based on the hardness of the OMGDH problem. We apply only one modification to the protocol that does not change the security: we replace the transmission of the server's masked elements to the client, in the offline phase, by sending a Cuckoo filter which encodes the masked elements.

However, sending the Cuckoo filter does not reveal any more information than sending the masked elements. By assumption, there is an algorithm $\mathcal{A}$ that the attacker can use on the modified protocol (with the Cuckoo filter) and breaks security with non-negligible probability γ. It is possible to devise the

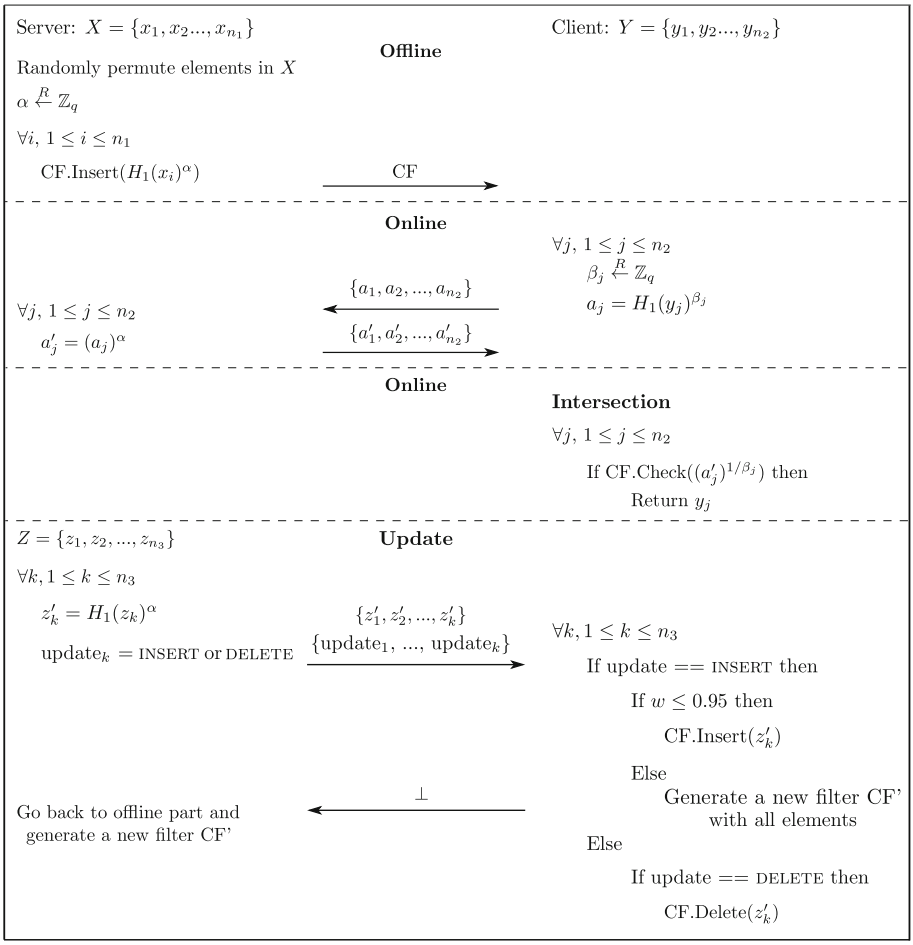


Fig. 2. Our optimized protocol combining the PSI protocol of Baldi *et al.* [4] with Cuckoo filter [11]. CF is a Cuckoo filter, *CF.Insert* is the insertion operation, *CF.Check* is the lookup operation and *CF.Delete* is the deletion operation.

algorithm $\mathcal{A}'$ with which the attacker can use to break the original protocol and works as follows: first $\mathcal{A}'$ runs the original protocol and keeps the raw set of masked elements received from the server. Then, the attacker encodes the masked elements as a Cuckoo filter and feeds $\mathcal{A}$ with it. Therefore, $\mathcal{A}$ observes the same view as in a run of the modified protocol and thus can break security with probability γ. Thus, $\mathcal{A}'$ breaks the security with the same probability.

We have preserved a weak notion of forward secrecy on the client side, as provided in the original version [4]. This weak notion ensures that elements exchanged in the past will remain confidential even if long-term keys are exposed. In the case of private contact discovery, where the client set is always almost the

same, compromising the client once reveals almost all the contacts used in previous executions, so forward secrecy does not provide much advantage. However, when the client set changes from one execution to another, as in the case of other applications, having forward secrecy is important. One such application is malware detection, where the client set may store networking data collected during a time interval. Modifying the protocol to relax forward secrecy allows precomputation in the client side which potentially improves its performance.

5 Implementation and Experimental Evaluation

We ran our experiments in a computer equipped with an Intel Haswell i7-4770K quadcore CPU with 3.4 GHz and 16 GB of RAM and Turbo Boost turned off. All tests were performed using only this machine, and network bandwidth and latency were simulated using the Linux command `tc`. For the Local Area Network (LAN) setting, the two parties (client and server) are connected via local host with 10 Gbps of bandwidth and a 0.2 ms Round-Trip Time (RTT). In addition to the LAN, we also considered three Wide Area Network (WAN) settings with 100 Mbps, 10 Mbps and 1 Mbps of bandwidth, each with an 80 ms RTT. These settings follow what was proposed by Chen *et al.* [7].

We evaluate the performance of the PSI protocols in the unbalanced setting, (in the full version [40] we also show the performance in the balanced setting) where $n_2 \in \{5535, 11041\}$ and $n_1 \in \{2^{16}, 2^{20}, 2^{24}\}$, as proposed by Chen *et al.* [7]. The size of each element was set to be $\sigma = 32$ bits, but this does not impact the performance of our protocol due to hashing. The output length of the hash function used in Baldi *et al.* [4] is l, as defined in Sect. 2.1. The run time of each protocol was measured from the beginning of the execution until the client computes the intersection. Each protocol was executed 10 times and the run times were computed as the average of these executions as done in [7,39].

5.1 Implementation

The implementation of OT+Hashing [39] was obtained from Pinkas *et al.* [39], available at https://github.com/encryptogroup/PSI. They used OpenSSL (v.1.0.1e) for the symmetric cryptographic primitives, the implementation of [3] and the code available at https://github.com/encryptogroup/OTExtension for the OT extension, and the MIRACL library (v.5.6.1) for ECC. According to our benchmarking, an exponentiation within their codebase takes 1.2 million cycles[11], which indicates a misconfigured version of MIRACL. Up to now, there is no implementation available for the Chen *et al.* [7] protocol, but we tried to reproduce the benchmarking scenarios as close as possible to their work.

We implemented our optimized protocol and the original [4] using the software provided by Pinkas *et al.* [39] replacing the NIST K-283 curve, available in MIRACL, with the GLS-254 curve. Our implementation of the GLS-254 curve

[11] Average of 2^{20} exponentiations performed on our Haswell machine.

takes around 50,000 cycles to compute an exponentiation, which is 24× faster than [39]. We note, however, that the K-283 curve is a more conservative choice of parameters. We used the Cuckoo filter implementation of Fan *et al.* [11] available at https://github.com/efficient/cuckoofilter. Our implementation is available at http://github.com/amandadavi7/PSI.

All protocols were implemented using C and C++ programming languages and executed using the same hardware. The same libraries were used to perform the cryptographic operations, except for the OTs in OT+Hashing [39] which still use the Koblitz curve. This does not impact the run time of the protocol since the cost of this operation is negligible when the number of elements is large.

5.2 Preprocessing

To improve performance, some PSI protocols can be divided into two phases (online and offline) without impact on security. The offline part of the protocol can be executed only once and reused in future executions. In the protocol proposed by Chen *et al.* [7], the server precomputes some values to facilitate the underlying FHE multiplications. In this case, only the server will use the precomputed data and no transfer to the client is required.

Unlike Chen *et al.* [7], in the basic protocol presented in Sect. 3, the server can preprocess the encryption/masking of all elements and send them to the client, which must store and reuse this data in all subsequent executions to compute the intersection, as shown in Sect. 4.1. Beyond using the precomputing allowed in the basic protocol, our approach also inserts each encrypted element into a Cuckoo filter, according to Sect. 4.2, in order to reduce the data that must be transmitted to the client and that should be stored.

Table 1 presents the preprocessing and data transmission time using the network settings defined in the beginning of Sect. 5. The run times of Chen *et al.* were obtained from their own paper [7], and since some parameters of the FHE can generate more efficient processing depending on the configuration, the preprocessing column may have two different values (we separate them with the symbol *) that will be used in the next section.

We note that the run times for the protocol proposed by Baldi *et al.* in [4] here presented, are for an implementation based on the binary elliptic curve GLS-254 and not for the original implementation proposed in [4] which works over a 1024-bit prime number. The improvements would be way more drastic had the original implementation proposed in [4] been used for comparison.

It is interesting to note that the preprocessing run times of our optimized protocol and our implementation of the original protocol are practically the same since we perform the same operations. However, by employing a Cuckoo filter to reduce the amount of data to be transmitted, our optimized version transmits up to 3.3× less data than [4] and is accordingly 3.3× faster.

Table 1. Preprocessing and transmission time for PSI protocols. The WAN setting has 80 ms RTT and the LAN 0.02 ms RTT. For the filter in our optimized protocol we have $v = 16$, $b = 3$, $w = 0.66$ and $\epsilon_{max} = 0.009155\%$. More details about Cuckoo filter is given in the full version of this paper [40]. Chen *et al.* [7] does not have transmission time, since only the server will use the precomputed data. Best values marked in bold.

					Transmission time (s)			
			Comm.	Preprocessing	LAN	WAN		
Protocol	n_1	n_2	Size (MB)	Time (s)	10 Gbps	100 Mbps	10 Mbps	1 Mbps
Chen *et al.* [7]	2^{24}	11041	-	**70.90, *76.80**	-	-	-	-
		5535	-	**64.10, *71.20**	-	-	-	-
	2^{20}	11041	-	**6.40**	-	-	-	-
		5535	-	**4.30**	-	-	-	-
	2^{16}	11041	-	**1.00**	-	-	-	-
		5535	-	**0.70**	-	-	-	-
Baldi *et al.* [4]	2^{24}	11041	160.00	334.17	0.13	15.73	136.32	1,345.55
		5535						
	2^{20}	11041	10.00	20.91	0.01	1.10	8.38	84.40
		5535						
	2^{16}	11041	0.56	1.31	0.01	0.19	0.53	5.09
		5535						
Our protocol	2^{24}	11041	**48.00**	333.62	**0.06**	**4.82**	**40.71**	**403.68**
		5535						
	2^{20}	11041	**3.00**	20.78	**0.00**	**0.60**	**2.55**	**25.63**
		5535						
	2^{16}	11041	**0.19**	1.30	**0.00**	**0.01**	**0.19**	**1.56**
		5535						

5.3 Comparison to Others PSI Protocols

The performance evaluation of the protocols was performed in the unbalanced setting. Since the code for Chen *et al.* [7] was not made available, we obtained results for this protocol and OT+Hashing [39] from [7]. As shown in Sect. 2.2, PSI protocols can be classified into five categories. Because the naive hashing and server-aided approaches have different security notions from the others, they are not included. PSI based on generic protocols are out of scope because they have limitations in run time and memory. Among the two remaining categories, OT-based PSI and PSI based on public-key, we will analyze the best protocol in each category comparing the results with our optimized proposal.

Table 2 shows the run time (in seconds) and the communication (in MBs) of the unbalanced scenario considering both the LAN and WAN settings. We have analyzed the best protocol for the OT-based PSI, the two best protocols for PSI based on public-key and we compare them with our optimized proposal.

Amongst the public-key protocols, our optimized proposal and the Baldi *et al.* [4] have the same communication cost ($2n_2\varphi$ bits) and, regarding run time, our approach is slightly better by employing the Cuckoo filter in the server database, what makes the final computation of the intersection more efficient, since the filter is already constructed and the lookup is done in $O(b)$ per element. Because of this, we omitted the figures related to Baldi *et al.*

protocol [4] in Table 2. In addition, comparing with the Chen[12] *et al.* protocol [7], our optimized approach transmits up to 59× less data and is up to 76× faster with 10 Gbps, for $n_2 = 5535$ and $n_1 = 2^{24}$. Comparing our optimized protocol with OT+Hashing [39], our approach transmits up to 1,413× less data and is up to 74× faster with 10 Gbps of bandwidth and 946× faster with 1 Mbps, for $n_2 = 5535$ and $n_1 = 2^{24}$.

Table 2. Run time and communication for unbalanced PSI protocols. In the communication column, the Chen *et al.* [7] may have two values due to different parameters used in the FHE system. For more information, see [7]. The results of OT+Hashing [39] and Chen *et al.* [7] were obtained from [7]. Best values marked in bold.

					Transmission time (s)			
		Parameters		Comm.	LAN	WAN		
Type	Protocol	n_1	n_2	Size (MB)	10 Gbps	100 Mbps	10 Mbps	1 Mbps
OT	OT+Hashing [39]	2^{24}	11041	480.90	40.50	88.00	449.50	4,084.80
			5535	480.40	40.10	87.90	449.20	4,080.60
		2^{20}	11041	30.90	3.30	7.00	29.80	263.70
			5535	30.40	3.10	6.80	29.00	260.00
		2^{16}	11041	2.60	0.70	1.50	3.30	21.60
			5535	2.10	0.70	1.40	2.90	19.80
Public key	Chen *et al.* [7]	2^{24}	11041	23.20, *21.10	44.50	46.90	63.50	*214.00
			5535	20.10, *12.50	41.10	43.10	*49.10	*139.90
		2^{20}	11041	11.50	6.40	7.60	15.80	99.00
			5535	5.60	4.30	4.90	9.00	49.30
		2^{16}	11041	4.10	2.00	2.40	5.40	35.00
			5535	2.60	1.10	1.30	3.20	21.80
	Our optimized protocol	2^{24}	11041	**0.67**	**0.87**	**1.52**	**1.86**	**7.81**
			5535	**0.34**	**0.54**	**1.04**	**1.21**	**4.31**
		2^{20}	11041	**0.67**	**0.67**	**1.31**	**1.65**	**7.59**
			5535	**0.34**	**0.34**	**0.83**	**1.00**	**3.97**
		2^{16}	11041	**0.67**	**0.66**	**1.29**	**1.64**	**7.57**
			5535	**0.34**	**0.33**	**0.82**	**0.99**	**3.93**

Our optimized approach performs well in unbalanced scenarios because our operations depend only on the client set size, with $2n_2\varphi$ bits transmitted and $3n_2$ exponentiations. Although exponentiations are considered an expensive operation, when performed a small number of times and with an efficient elliptic curve implementation, a curve-based protocol becomes competitive with the others.

In a recent paper, Kiss *et al.* [23] present many PSI protocols, where the closest to our proposal is ECC-DH-PSI [19,27]. In the preprocessing stage, the server needs to compute n_1 exponentiations like in our protocol, but the client also needs to compute n_1 exponentiations. In some applications, such as private contact discovery, this amount of exponentiations in the client side could be

[12] In the communication column of Table 2, the protocol [7] can have 2 different values, because according to the networking setting it is better that operations take more time and generate less data than taking less time but producing more data. This trade-off can be changed in FHE by adjusting the system parameters.

prohibitive, because typically the client has a resource constrained device. Considering $n_1 = 2^{20}$ and according to [23], the preprocessing takes 1,325 s while our proposal takes 21 s (using a 1 Gbps network), which is 63× faster. Moreover, the server sends $n_1\varphi$ ($\varphi = 284$ in their case), that adds up to 35.5 MB, while our proposal just sends a 2.125 MB filter for $\epsilon = 0.05\%$ ($v = 16$, $w = 0.94$ and $b = 17$) and a 5 MB filter for $\epsilon = 1.6 \times 10^{-7}\%$ ($v = 32$, $w = 0.8$ and $b = 5$)[13]. This is 16.7× and 7.1× less data to be transmitted, respectively.

In the online phase the amount of data to be transmitted is asymptotically the same, $2n_2\varphi$, but concretely Kiss *et al.* [23] use $\varphi = 284$ bits for the K-283 curve with compression, and we have $\varphi = 256$ bits for the GLS-254 curve. Considering the number of exponentiations, their approach needs to compute $2n_2$ operations while our protocol computes $3n_2$. This advantage happens because the ECC-DH-PSI from [23] does not provide forward secrecy on the client side and reuse the same key across all protocol executions.

In order to reduce the amount of data to be stored by the client, Kiss *et al.* [23] use a Bloom filter, while our optimized approach employs a Cuckoo filter. The Cuckoo filter allows deletions while the traditional Bloom filter does not and uses 30% less space for the same FPR [10]. While counting Bloom filters do allow deletions, this happens at the cost of using 3-4× more space.

In summary, our protocol provides an efficient preprocessing phase, forward secrecy on the client side and a filter that needs less storage space. The ECC-DH-PSI protocol from [23] has an asymptotically faster online phase, but the performance improvement is small in the unbalanced setting when n_2 is small. Moreover, their protocol does not provide any forward secrecy to clients and the preprocessing phase is expensive and can be prohibitive on mobile devices.

6 Conclusions

Private set intersection is an important cryptographic primitive to allow two parties to perform joint operations on their private sets without revealing additional information beyond the intersection. Despite many protocols available in the literature, few of them provide solutions that are efficient in both run time and data transmission. In most approaches, the computational cost is based on both the server and client set sizes, giving no advantages in the unbalanced setting.

We show that the protocol of Baldi *et al.* [4] based on public-key cryptography, with our optimizations, becomes an efficient, practical and simple one-way PSI protocol for unbalanced sets that ensures forward secrecy on the client side. Additionally, we implemented the protocol using the GLS-254 binary elliptic curve with point compression using techniques considered state of the art, that allow a better comparison with the other proposed approaches.

Our optimized protocol with this implementation provides an interesting trade-off between preprocessing and the online phase of the protocol, where for

[13] The filter in the client side from [23] has $\epsilon = 0.1\%$ and $\epsilon = 10^{-7}\%$, respectively.

$n_1 = 2^{24}$ the preprocessing takes less than six minutes of computing time (recall that this phase needs to be done only once) and the online phase for $n_2 = 11041$ takes less than 8 seconds even with 1 Mbps bandwidth. The client needs to store only 48 MB of information for this configuration. We believe that our improved protocol is a practical alternative for the solutions currently in place for privacy-preserving contact discovery in existing social networks.

Acknowledgements. This work was in part supported by the Intel/FAPESP grant 14/50704-7, project "Secure Execution of Cryptographic Algorithms". We thank Anderson Nascimento and Fabian Monrose for discussion and comments on an earlier version.

References

1. Aranha, D.F., Fouque, P.-A., Qian, C., Tibouchi, M., Zapalowicz, J.-C.: Binary elligator squared. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 20–37. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_2
2. Arbitman, Y., Naor, M., Segev, G.: Backyard cuckoo hashing: constant worst-case operations with a succinct representation. In: FOCS, pp. 787–796. IEEE Computer Society (2010)
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM Conference on Computer and Communications Security, pp. 535–548. ACM (2013)
4. Baldi, P., Baronio, R., Cristofaro, E.D., Gasti, P., Tsudik, G.: Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In: ACM Conference on Computer and Communications Security, pp. 691–702. ACM (2011)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
6. Camenisch, J., Zaverucha, G.M.: Private intersection of certified sets. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 108–127. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03549-4_7
7. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS, pp. 1243–1255. ACM (2017)
8. Debnath, S.K., Dutta, R.: Towards fair mutual private set intersection with linear complexity. Secur. Commun. Netw. **9**(11), 1589–1612 (2016)
9. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: ACM Conference on Computer and Communications Security, pp. 789–800. ACM (2013)
10. Eppstein, D.: Cuckoo filter: simplification and analysis. In: SWAT. LIPIcs, vol. 53, pp. 8:1–8:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
11. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.: Cuckoo filter: practically better than bloom. In: CoNEXT, pp. 75–88. ACM (2014)
12. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive (2012)
13. Fan, L., Cao, P., Almeida, J.M., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000)
14. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_17

15. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_1
16. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178. ACM (2009)
17. Hankerson, D., Karabina, K., Menezes, A.: Analyzing the Galbraith-Lin-Scott point multiplication method for elliptic curves over binary fields. IEEE Trans. Comput. **58**(10), 1411–1420 (2009)
18. Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: NDSS. The Internet Society (2012)
19. Huberman, B.A., Franklin, M.K., Hogg, T.: Enhancing privacy and trust in electronic communities. In: EC, pp. 78–86 (1999)
20. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
21. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 418–435. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_26
22. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 195–215. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_13
23. Kiss, A., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile application. PoPETs **2017**(4), 97–117 (2017)
24. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_15
25. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_4
26. Lopez, J., Dahab, R.: New point compression algorithms for binary curves. In: IEEE Information Theory Workshop - ITW 2006, pp. 126–130, March 2006. https://doi.org/10.1109/ITW.2006.1633795
27. Meadows, C.A.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: IEEE Symposium on Security and Privacy, pp. 134–137. IEEE Computer Society (1986)
28. Mezzour, G., Perrig, A., Gligor, V., Papadimitratos, P.: Privacy-preserving relationship path discovery in social networks. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 189–208. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_13
29. Nagaraja, S., Mittal, P., Hong, C., Caesar, M., Borisov, N.: BotGrep: finding P2P bots with structured graph analysis. In: USENIX Security Symposium, pp. 95–110. USENIX Association (2010)
30. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC, pp. 245–254. ACM (1999)
31. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: SODA, pp. 448–457. ACM/SIAM (2001)
32. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS. The Internet Society (2011)

33. Oliveira, T., Aranha, D.F., Hernandez, J.L., Rodríguez-Henríquez, F.: Improving the performance of the GLS254. CHES Rump Session (2016)
34. Oliveira, T., López, J., Aranha, D.F., Rodríguez-Henríquez, F.: Two is the fastest prime: lambda coordinates for binary elliptic curves. J. Cryptogr. Eng. **4**(1), 3–17 (2014)
35. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) ESA 2001. LNCS, vol. 2161, pp. 121–133. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_10
36. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: USENIX Security Symposium, pp. 515–530. USENIX Association (2015)
37. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15
38. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security Symposium, pp. 797–812. USENIX Association (2014)
39. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. **21**(2), 7:1–7:35 (2018)
40. Resende, A.C.D., Aranha, D.F.: Faster Unbalanced Private Set Intersection. IACR Cryptology ePrint Archive (2017). https://eprint.iacr.org/2017/677
41. Yao, A.C.: Protocols for secure computations (Extended Abstract). In: FOCS, pp. 160–164. IEEE Computer Society (1982)
42. Yao, A.C.: How to generate and exchange secrets (Extended Abstract). In: FOCS, pp. 162–167. IEEE Computer Society (1986)

SWiM: Secure Wildcard Pattern Matching from OT Extension

Vladimir Kolesnikov[1], Mike Rosulek[2(✉)], and Ni Trieu[2]

[1] Georgia Institute of Technology, Atlanta, Georgia
kolesnikov@gatech.edu
[2] Oregon State University, Corvallis, USA
{rosulekm,trieun}@eecs.oregonstate.edu

Abstract. Suppose a server holds a long text string and a receiver holds a short pattern string. Secure pattern matching allows the receiver to learn the locations in the long text where the pattern appears, while leaking nothing else to either party besides the length of their inputs. In this work we consider secure *wildcard* pattern matching (WPM), where the receiver's pattern is allowed to contain wildcards that match to any character.

We present SWiM, a simple and fast protocol for WPM that is heavily based on oblivious transfer (OT) extension. As such, the protocol requires only a small constant number of public-key operations and otherwise uses only very fast symmetric-key primitives. SWiM is secure against semi-honest adversaries. We implemented a prototype of our protocol to demonstrate its practicality. We can perform WPM on a DNA text (4-character alphabet) of length 10^5 and pattern of length 10^3 in just over 2 s, which is over two orders of magnitude faster than the state-of-the-art scheme of Baron et al. (SCN 2012).

1 Introduction

Secure two-party computation allows mutually untrusted parties to perform a computation on their private inputs without revealing any additional information except for the result itself. Over the last few years, secure two-party computation has been extensively studied and has become practical for a variety of applications. Two adversarial models are usually considered. In the semi-honest model, the adversary is assumed to follow the protocol, while trying to learn information from the protocol transcript. In the malicious model, the adversary can follow an arbitrary polynomial-time strategy. We consider the semi-honest model in this work.

Pattern matching is a basic problem in secure computation. It has been extensively studied in the past decade, e.g., [HL08, BEDM+12, DF13, DCFT13, FHV13, YSK+13, HT14, CS15, YSK+14, WJW+15, WZX17]. Pattern matching is frequently used in text processing, database search [GHS10, CS15], network

V. Kolesnikov—Work done while the author was at Bell Labs.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 222–240, 2018.
https://doi.org/10.1007/978-3-662-58387-6_12

security [NN10], DNA analysis [OPJM10], and other practical algorithms. The most commonly considered variant of secure pattern matching, which we will call exact PM, is the setting where a server with input a text $x \in \Sigma^n$ (over some alphabet Σ) interacts with a receiver with input a pattern $p \in \Sigma^m$ (for $m < n$). The receiver learns where the pattern occurs as a substring of the server's text without revealing any additional information. There are several important variants of pattern matching, including approximate pattern matching and outsourced pattern matching, which we discuss in Sect. 1.2.

In this work, we focus on secure pattern matching with *wildcards*, which we will call WPM. In this variant, the receiver's pattern can include wildcard characters that can match any character in the data, hence $p \in (\Sigma \cup \{\star\})^m$. With wildcards, the security requirements are more demanding: the server should not learn which positions of p contain wildcards, and in the case of a match the receiver should not learn the text character that matches a wildcard character in the pattern (unless this could be inferred from the presence or absence of an overlapping match).

Allowing wildcards in a pattern matching functionality has been well studied in the absence of a security requirement [CH02, CWZ+06, CLI07, CEPR09, SOF10, Tha11, BGVV14, BI14, SSSS15, AWY15], and is motivated by the goal of providing the facility of searching with errors/unknowns. Privacy issues arise in searching on sensitive data and secure pattern matching with wildcards has applications, e.g., in computational genetics and DNA analysis. Indeed, consider the case of a hospital or biomedical research center holding patient genomic data, and a researcher holding a specific cancer marker sequence with some errors. The researcher wishes to know the frequency and positions of the gene occurrences in the database. Due to the genome's highly sensitive nature, the hospital is required keep genomic data private, while the researcher needs to protect specific genome sequence he is working on. The abundance of WPM applications, such as privacy-preserving DNA matching described above, is our main motivation for improving the state-of-the-art in secure wildcard pattern matching.

1.1 Pattern Matching with Wildcards

In this section, we discuss directions and related work that achieves, or can be naturally used to achieve, the WPM functionality in the semi-honest setting.

Circuit Based. Generic secure computation protocols [Yao86, GMW87], allowing evaluation of arbitrary functions, have seen tremendous performance improvements in the last decade. Modern garbled circuit (GC) protocols evaluate two million AND gates per second on a 1 Gbps LAN. Several garbled circuits for Pattern Matching and its variants were studied in [JKS08, KM10]. The best protocol using this technique were proposed by Katz and Malka [KM10]. The authors showed how to modify Yao's garbled circuit to solve Pattern Matching where the size of circuit is linear in the (*a priori* upper bound on the) number of occurrences of the pattern in text. While it is possible to extend circuit-based protocol [KM10] to allow wildcards, it would still require a bound

on the number of matches to be provided *a priori* for the circuit construction. When such bound is high or simply unknown, their protocol suffers corresponding performance penalty. The work [KM10] does not provide implementation or experimental results.

Homomorphic Encryption Based. To our knowledge, Hazay and Toft [HT10] were the first to explicitly consider wildcard secure pattern matching. The core idea of their protocol is that the receiver provides the wildcard positions to the server in an encrypted form, and the substrings of the server's text are obliviously modified so as to match the pattern at those positions. Later, Vergnaud [Ver11] improved the work of [HT10] by employing Fast Fourier Transform. Both works rely on the fact that if a pattern bit p_i is equal to a text bit t_i, then $(t_i - p_i)^2$ equals 0, and otherwise it is equal to 1. The work of Vergnaud [Ver11] requires $O((n+m)\kappa^2)$ communication and $O(n \log m)$ computational cost in both semi-honest and malicious settings, where κ is computational security parameters. As [HT10, Ver11] do not provide the experimental results, we do not compare execution times with their work.

In 2012, Baron et al. [BEDM+12] proposed an efficient pattern matching protocol called 5PM, for 5ecure Pattern Matching. 5PM works with character (non-binary) wildcards, and was the first to provide and accompanying implementation. The protocol is based on an insecure pattern matching algorithm proposed by Hoffmann [HHD11]. To obtain a secure pattern matching, 5PM modifies the algorithm [HHD11] to work with basic linear operations, which allowed instantiation with additive homomorphic encryption. 5PM requires $O(n\kappa)$ communication and $O(n+m)$ computational costs in semi-honest setting. In Sect. 5 we compare our performance to that of 5PM and report $2-499\times$ performance improvement even on medium-size instances.

Yasuda et al. [YSK+14] extend the exact pattern matching protocol of [YSK+13] to support wildcards. The security of [YSK+14] is based on the polynomial LWE assumption. Their scheme operates by blocks, limited by the lattice dimension; for larger inputs $\boldsymbol{x}$, inefficiency is introduced either by using a larger lattice, or by the difficulty and cost of handling boundaries of blocks. In [YSK+14], the authors do not present the performance comparison with 5PM protocol, but indirectly this can be calculated. Yasuda et al. [YSK+14] mention that their protocol only $4-5\times$ slower than the protocol [YSK+13], which does not allow wildcards. In addition, [YSK+13] estimated that their work is about $10\times$ faster than 5PM when using much stronger hardware than 5PM ([YSK+13] experiments were performed on Intel Xeon X3480 3.07 GHz machine with 16 GB RAM, while 5PM [BEDM+12] used Intel dual quad-core 2.93 GHz machine with 8 GB RAM). Putting all together, we conclude that [YSK+14] is approximately $2-2.5\times$ faster than 5PM. In contrast, our protocol is $2-499\times$ times faster than 5PM, while running on weak commodity hardware (same at 5PM, cf. Sect. 5.1); this translates into the corresponding improvement over [YSK+14] as well. Further, our approach is simpler and easier to implement.

We mention recent work of Saha and Koshiba [SK17], which improves on the work of [YSK+14] by proposing a new packing method that efficiently addresses

continuous wildcards occurring in the pattern (e.g., pattern 10⋆⋆⋆⋆01⋆⋆⋆110 has $k = 3$ sub-patterns: 10, 01, and 110). The main idea of their packing method is to let the receiver break down the pattern into k sub-patterns and have the parties perform the traditional pattern matching on these patterns. This solution is about $k\times$ faster than previous work [YSK+14]. However, it reveals significant information about the pattern, especially for larger k.

1.2 Variants of Pattern Matching

For completeness, we briefly discuss work on several additional variants of secure pattern matching.

Exact Pattern Patching. To our knowledge, Troncoso-Pastoriza et al. [TPKC07] were the first to consider secure pattern matching. Their protocol is based on oblivious automaton evaluation. The protocol [TPKC07] requires $O(nm)$ communication and computational cost. Several follow-up works [Fri09, MNSS12] improved the computational cost and reduced the round complexity. Another line of work [HL08, GHS10] is based on oblivious pseudorandom functions (OPRF), and obtains security in the malicious setting using $O(nm)$ communication and computational cost with $O(m)$ rounds. De Cristofaro et al. [DCFT13] consider a secure and efficient pattern matching protocol which hides the length of the pattern.

Approximate/Fuzzy Pattern Matching. The functionality of this problem is to find the text positions matches approximately (rather than exactly). This problem can be solved by determining whether the Hamming distance between each text substring and the pattern is less than a threshold t. Hazay and Toft [HT10, HT14] proposed a malicious-secure solution with $O(nt)$ communication and $O(nm)$ computation costs.

Outsourcing Pattern Matching. In this setting, parties outsource their encrypted data and computation to an untrusted server, while maintaining data privacy. The main goal here is to minimize the communication and computational overhead of the parties by relying on the powerful resources of the untrusted server. The first work that considered secure pattern matching in the cloud setting can be traced back to Faust et al. [FHV13]. Other follow-up works are [WJW+15]. Recently, Wei et al. [WZX17] proposed an efficient solution by combining a secret sharing scheme and oblivious transfer which requires $O(\kappa)$ computation and $O(mn)$ communication costs. Outsourcing pattern matching can be viewed as substring searchable encryption which are studied in [CS15, ÇCL+17].

2 Overview of Our Results and Techniques

In this work we present SWiM (**S**ecure **W**ildcard Pattern **M**atching), a protocol for WPM based on two fast cryptographic tools: oblivious transfer and secure

Table 1. Communication (bits) and computation (number of exponentiations) complexities of WPM protocols in the semi-honest setting, where n is length of text,m is length of pattern; and λ and κ are the statistical and computational security parameters, respectively. $\lambda = 40$ and $\kappa = 128$ in our protocols, while κ is in the range 1024–2048 in [Ver11, BEDM+12] protocols (due to their use of public-key primitives).

Protocol	Computation		Communication		Rounds		Security model
	Online	Offline	Online	Offline	Online	Offline	
[Ver11]	$\mathcal{O}(n \log m)$		$\mathcal{O}((m+n)\kappa^2)$		$O(1)$		Semi-honest & malicious
[BEDM+12]	$\mathcal{O}(m+n)$		$\mathcal{O}(n\kappa)$		2		Semi-honest
Ours	0	$\mathcal{O}(\kappa)$	$\mathcal{O}(m+(\lambda+\kappa)n)$	$\mathcal{O}(nm)$	2	2	Semi-honest

string equality test (given two strings of equal lengths, without wildcards, determine whether they are equal). Thanks to recent optimizations in oblivious transfer protocols [Bea96, IKNP03, KK13, ALSZ13], it is possible to realize a large number of OT instances with amortized cost of only a few μs. Kolesnikov et al. [KKRT16] give a protocol for secure string equality test based on techniques for efficient OT. With their protocol, one can perform many private equality tests with amortized cost of 5 µs.

Overview of Techniques. Suppose the sender holds a string $\boldsymbol{x} \in \{0,1\}^*$ and the receiver holds a pattern $\boldsymbol{p} \in \{0,1,\star\}^*$.

As a very simple warm up, consider the case that $|\boldsymbol{x}| = |\boldsymbol{p}| = 1$. The receiver will first encode its pattern $\boldsymbol{p} \in \{0,1,\star\}$ as a pair of bits $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}})$ ("p-star & p-bar"), using the following encoding:

$\boldsymbol{p}$	$\overset{\star}{\boldsymbol{p}}$	$\bar{\boldsymbol{p}}$
$\star$	1	0
1	0	1
0	0	0

(1)

The significance of this encoding is the following:

$$\boldsymbol{x} \text{ matches pattern } \boldsymbol{p} \iff \boldsymbol{x} = \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x} \oplus \bar{\boldsymbol{p}} \tag{2}$$

Indeed, if $\boldsymbol{p} = \star$, then $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}}) = (1,0)$, so the RHS of (2) simplifies to $\boldsymbol{x}$ and the two sides equal (regardless of $\boldsymbol{x}$). On the other hand, if $\boldsymbol{p} \neq \star$, then $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}}) = (0, \boldsymbol{p})$, so the RHS simplifies to $\boldsymbol{p}$ and the two sides equal if and only if $\boldsymbol{p} = \boldsymbol{x}$.

Our next trick is to blindly evaluate Eq. (2) using a single OT evaluation. The parties invoke an instance of 1-out-of-2 bit-OT, where the sender gives inputs $(\boldsymbol{k}, \boldsymbol{k} \oplus \boldsymbol{x})$, and the receiver gives input $\overset{\star}{\boldsymbol{p}}$. Here $\boldsymbol{k}$ is a random bit chosen by the sender. Note that the receiver's output from this OT is $\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}$.

Now, adding $\boldsymbol{k}$ to both sides of the equation in (2), we have that $\boldsymbol{x}$ matches pattern $\boldsymbol{p}$, if and only if $\boldsymbol{k} \oplus \boldsymbol{x} = (\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}) \oplus \bar{\boldsymbol{p}}$. Importantly, the LHS of this equation is known to the sender, while the RHS is known to the receiver. At the same time, the random mask $\boldsymbol{k}$ hides all information about $\boldsymbol{x}$ from the receiver.

We can summarize the above gadget as follows: using a single OT of bits, the sender and receiver each compute a bit which is *the same bit if and only if* $\boldsymbol{x}$ *matches pattern (possibly wildcard)* $\boldsymbol{p}$.

This technique can be easily extended to the case of WPM with $|\boldsymbol{x}| = |\boldsymbol{p}| = n$ by simply doing the above gadget n times, bit-by-bit. After doing so, each party will hold an n-bit string (without wildcards); these two strings will be equal if and only if $\boldsymbol{x}$ matches the pattern $\boldsymbol{p}$. An example is given in Fig. 1 (we simply extend the notation $\oplus$ and $\cdot$ to bit-vectors). In short, we have reduced the problem of WPM with $|\boldsymbol{x}| = |\boldsymbol{p}|$ to the problem of secure (exact, no wildcards) equality test of strings. We complete the wildcard pattern matching by actually testing the equality of these strings, using the efficient protocol of Kolesnikov et al. [KKRT16].

The security of this protocol (in the semi-honest model) is easy to understand: the only new information is that the receiver obtains output $\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}$, which leaks no information about the sender's input $\boldsymbol{x}$ since $\boldsymbol{k}$ is uniform. Now consider extending this approach to the general case of WPM with $|\boldsymbol{x}| > |\boldsymbol{p}|$. The idea is the natural one: for each $i \in \{1, |\boldsymbol{x}| - |\boldsymbol{p}| + 1\}$ simply perform the above approach on the substring $x[i \dots i + |\boldsymbol{p}| - 1]$ and $\boldsymbol{p}$. Unpacking the abstractions reveals room for optimizations, as follows. While the previous constructions were presented in terms of OT of *bits,* the OT of strings is significantly more efficient in practice. We observe that in each subprotocol, the receiver's OT choice bits are always the same $\overset{\star}{\boldsymbol{p}}$, allowing corresponding OT instances to be combined easily. Hence instead of $|\boldsymbol{p}|(|\boldsymbol{x}| - |\boldsymbol{p}| + 1)$ instances of bit-OT, we can use $|\boldsymbol{p}|$ instances of string-OT, with strings of length $|\boldsymbol{x}| - |\boldsymbol{p}| + 1$. This optimization actually reduces costs by a multiplicative factor of the security parameter. The details are given in Sect. 4.

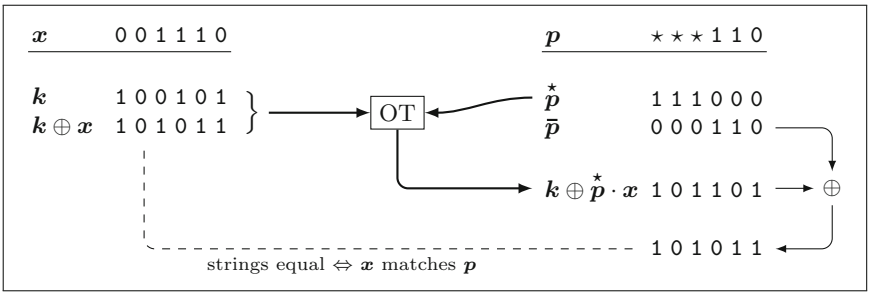


Fig. 1. Illustration of the main idea behind our protocol: using oblivious transfer and private string equality test to perform private string equality with wildcards.

In Sect. 4.1 we present additional optimizations and extensions, such as moving almost all of the cost to the offline, amortization and efficient handling of non-binary alphabets.

Efficiency. SWiM requires only $O(\kappa)$ public-key operations (all in the offline phase). In terms of communication, our protocol requires $O(mn)$ in the offline phase, but only $O(m + (\lambda + \kappa)n)$ in online phase. Here, κ, λ are the computational and statistical security parameters, respectively. As noted previously, all constants under the big-O are small, as we use fast optimized building blocks. We describe the performance of representative Secure Wildcard Pattern Matching protocols in Table 1.

We note that SWiM is efficient *concretely.* This is because we carefully optimize both computation and communication. Further, we use algorithmically- and implementation-optimized building blocks, namely the OT extension of [ALSZ13] and private equality test of [KKRT16]. In particular, the [KKRT16] equality test is *independent* of the length of the players' inputs.

This significantly improves over the state-of-the-art secure wildcard pattern matching protocol of [BEDM+12]. In Sect. 5, we report in detail on implementation and evaluation, and find that SWiM is a $2 - 499\times$ faster than 5PM, and continues to scale well on larger instances. 5PM considers WPM instances on DNA text of length up to 10^5 and pattern of length up to 10^3. These larger instances require only 1.96 s in our protocol, in comparison with 304.53 s with 1024-bit key and 978.94 s with 2048-bit key using [BEDM+12].

3 Preliminaries

3.1 Notation

Throughout the paper we use the following notation: The length of the text is n, while the length of the pattern is m. Wildcard is denoted by $\star$. The computational and statistical security parameters are denoted by κ, λ, respectively. $[m]$ to denote a set $\{1, \ldots, m\}$.

The notation OT_m denotes a 1-out-of-2 OT where the string is m bits long. We denote vectors in bold $\boldsymbol{a}$, and matrices in capitals A. For the vector, we let $\boldsymbol{a}_{[i,j]}$ denote the sub-vector of $\boldsymbol{a}$ from i-th bit to j-th bit, and a_i denote the i-th bit of vector $\boldsymbol{a}$. Given vectors $\boldsymbol{a} = a_1 \| \cdots \| a_n$ and $\boldsymbol{b} = b_1 \| \cdots \| b_n$, we define $\oplus$ and $\cdot$ operations as follows. We use the notation $\boldsymbol{a} \oplus \boldsymbol{b}$ to denote the vector $(a_1 \oplus b_1) \| \cdots \| (a_n \oplus b_n)$. Similarly, the notation $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the vector $(a_1 \cdot b_1) \| \cdots \| (a_n \cdot b_n)$. Let $c \in \{0, 1\}$, then $c \cdot \boldsymbol{a}$ denotes the vector $(c \cdot a_1) \| \cdots \| (c \cdot a_n)$. For a matrix A, we let $\mathbf{a}_i$ denote the i-th row of A, $\mathbf{a}^j$ denote the j-th column of A; A_i^j denote the entry of A at the i-th row and the j-th column.

Consider an alphabet Σ. We define a **pattern matching relation** $\preceq$ via the following rules: (1) $a \preceq a$ for $a \in \Sigma$; (2) $\star \preceq a$ for $a \in \Sigma$. We extend the notation to vectors as $\boldsymbol{x} \preceq \boldsymbol{y} \Leftrightarrow (\forall i) x_i \preceq y_i$. If $\boldsymbol{p} \preceq \boldsymbol{x}$ we say that $\boldsymbol{x}$ **matches the pattern** $\boldsymbol{p}$.

3.2 Oblivious Transfer

Oblivious Transfer (OT) is a ubiquitous cryptographic primitive, and necessary for secure computation, which was introduced by Rabin [Rab05]. In OT, a

sender with two input strings (x_0, x_1) interacts with a receiver who has a input choice bit b. In a privacy-preserving way, the receiver learns x_b without learning anything about x_{1-b}, while the sender learns nothing about b. Rabin's protocol requires expensive public key cryptography. Ishai et al. [IKNP03] proposed OT extension, an efficient protocol that evaluates a small number of expensive OTs, from which a large number of OTs can be performed using only cheap symmetric-key operations. OT extension, to which we sometimes refer as IKNP, has become a core building block in many aspects of secure computation such as Garble Circuit, Private Set Intersection [PSZ14,KKRT16,KMP+17], Hamming Distance [BCP13]. We describe the ideal functionality for OT in Fig. 2.

Despite the wide use of OT, there are very few improvement of IKNP OT protocol in semi-honest setting. In 2013, Kolesnikov and Kumaresan [KK13] proposed an generalization of IKNP OT extension for short secrets, which brought $O(\log(\kappa))$ factor performance improvement in communication and computation, where κ is security parameter. They also proposed an IKNP optimization,

PARAMETERS: A bit length m, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for pair-input $(\boldsymbol{x}_0, \boldsymbol{x}_1) \subseteq \{0,1\}^m$ from the sender $\mathcal{S}$
- Wait for bit-input $b \in \{0,1\}$ from the receiver $\mathcal{R}$
- Give output $\boldsymbol{x}_b$ to the receiver $\mathcal{R}$.

Fig. 2. Oblivious Transfer functionality OT_m.

PARAMETERS: Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for input $\boldsymbol{x}_0 \in \{0,1\}^*$ from the sender $\mathcal{S}$.
- Wait for input $\boldsymbol{x}_1 \in \{0,1\}^*$ from the receiver $\mathcal{R}$.
- Give the receiver $\mathcal{R}$ output 1 if $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and 0 otherwise.

Fig. 3. The Private Equality ideal functionality $\mathcal{F}_{\mathsf{peqt}}$

PARAMETERS: A text length n, a pattern length m, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for text $\boldsymbol{x} \in \{0,1\}^n$ from the sender $\mathcal{S}$
- Wait for pattern $\boldsymbol{p} \in \{0,1,\star\}^m$ from the receiver $\mathcal{R}$
- Give the receiver $\mathcal{R}$ output $\{i \in [n-m+1] \mid \boldsymbol{p} \preceq \boldsymbol{x}_{[i,i+m-1]}\}$ (see Section 3.1 for notation)

Fig. 4. Wildcard Pattern Matching functionality $\mathcal{F}_{\mathsf{wpm}}^{n,m}$.

saving on the auxiliary matrix transfer. Later same year, Asharov et al. [ALSZ13] proposed several IKNP optimizations (one of which was the optimization independently discovered by [KK13]). Importantly, [ALSZ13] also provided optimized implementation of (improved) IKNP OT protocol.

[ALSZ13] also presented optimizations for a useful variant of OT. In Correlated OT (COT), the sender's OT inputs x_0, x_1 are chosen randomly subject to $x_0 \oplus x_1 = \Delta$, where Δ is chosen by the sender (possibly a different Δ for each OT instance). In this case, it is possible to let the protocol itself "choose" the value x_0 randomly. Doing so reduces the bandwidth requirement by approximately half. It is easy to see that we require only this weaker variant of OT for pattern matching, hence our implementation takes advantage of this optimization.

3.3 Private Equality Test

Definition. A **Private Equality Test (PEQT)** is a 2-party protocol in which the sender with input string $\boldsymbol{x}_0$ interacts with a receiver with input string $\boldsymbol{x}_1$ in the following way. The receiver learns a bit indicating whether $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and nothing else, while the sender learns nothing about $\boldsymbol{x}_1$. We describe the ideal functionality for an PEQT in Fig. 3.

To our knowledge, PEQT was first introduced in 1996 by Fagin, Naor, and Winkler [FNW96]. Follow-up works [NP99, BST01, Lip03] improved the efficiency of PEQT, while still relying on expensive public-key operations. PEQT is heavily used in two-party private set intersection (PSI) protocols [FNP04]. Recently, Kolesnikov et al. [KKRT16], in the context of PSI proposed an efficient PEQT, which was constructed by applying novel encodings inside the OT extension matrix. Their protocol, cast as a variant of Oblivious PRF, executes many PEQT instances by using only cheap symmetric cryptographic operations, apart from base OTs. Concretely, the amortized cost of each PEQT instance with unbounded input domain $\{0,1\}^*$ is only a few symmetric-key operations and 488 bits in communication. We heavily rely on the high-performing PEQT protocol of [KKRT16] in this work.

4 SWiM: The Main Construction

We present SWiM, our main construction for the WPM functionality in Fig. 4. It closely follows and formalizes the high-level overview presented in Sect. 2. For readability, we present SWiM for binary alphabet $\Sigma = \{0,1\}$. In Sect. 4.1 we show how to easily extend it to an arbitrary Σ. We first run OT extension with the chosen inputs defined in Fig. 2, which will allow the receiver to compute $\boldsymbol{\alpha} = \boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x} \oplus \bar{\boldsymbol{p}}$. Recall, as discussed in Sect. 2, $\boldsymbol{x}$ matches $\boldsymbol{p}$, iff $\boldsymbol{\alpha}$ equals to $\boldsymbol{k} \oplus \boldsymbol{x}$ held by the sender. This equality is efficiently checked in bulk by calling instances of Private Equality Test defined in Fig. 3, with the result delivered to the receiver and output. The SWiM protocol is presented in Fig. 5 and is proven secure against semi-honest adversaries.

PARAMETERS:

1. Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
2. A length n of text, a length m of pattern. Define $n' = n - m + 1$
3. A repetition encoding $C : \{0,1\} \to \{0,1\}^{n'}$ defined by $C(a) = a^{n'}$ for $a \in \{0,1\}$.
4. Ideal OT and $\mathcal{F}_{\mathsf{peqt}}$ primitives defined in Figure 2 and Figure 3, respectively.

INPUT OF $\mathcal{S}$: a text $\boldsymbol{x} \in \{0,1\}^n$

INPUT OF $\mathcal{R}$: a pattern $\boldsymbol{p} \in \{0,1,\star\}^m$ encoded into $\bar{\boldsymbol{p}}, \overset{\star}{\boldsymbol{p}} \in \{0,1\}^m$, as described in Section 2.

PROTOCOL:

1. **[Random Keys]** $\mathcal{S}$ chooses $\{\boldsymbol{k}_i\}_{i\in[m]} \leftarrow \{0,1\}^{n'}$ at random
2. **[OT]** For each $i \in [m]$, $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathsf{OT}_{n'}$-functionality
   (a) $\mathcal{R}$ acts as *receiver* with a input-bit $\overset{\star}{p}_i$.
   (b) $\mathcal{S}$ acts as *sender* with a ordered pair input $(\boldsymbol{k}_i, \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]})$
   (c) $\mathcal{R}$ receives output $\boldsymbol{q}_i$
3. **[Matrix Form]**
   (a) $\mathcal{S}$ forms $m \times n'$ matrix T such that the i-th row of T is the vector $\boldsymbol{t}_i = \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]}$
   (b) $\mathcal{R}$ forms $m \times n'$ matrix U such that the i-th row of U is the vector $\boldsymbol{u}_i = \boldsymbol{q}_i \oplus C(\bar{p}_i)$.
4. **[PEQ]**
   (a) For each $i \in [n']$, $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{peqt}}$-functionality:
      - $\mathcal{S}$ acts as *sender* with input $\boldsymbol{t}^i$ as the i-th column of T
      - $\mathcal{R}$ acts as *receiver* with input $\boldsymbol{u}^i$ as the i-th column of U
   (b) $\mathcal{R}$ outputs $\{i \in [n'] \mid i\text{th instance of } \mathcal{F}_{\mathsf{peqt}} \text{ outputs } 1\}$

Fig. 5. SWiM: **S**ecure **Wi**ldcard Pattern **M**atching Protocol for $\Sigma = \{0,1\}$.

Correctness. The main observation of OT-extension is that the receiver obtains output $\boldsymbol{q}_i$ such that:

$$\boldsymbol{q}_i = \boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]} = \begin{cases} \boldsymbol{k}_i, & \text{if } \overset{\star}{p}_i = 0 \\ \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]}, & \text{if } \overset{\star}{p}_i = 1 \end{cases}$$

Therefore, the i-th row of U is equal to $\boldsymbol{u}_i = \boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]} \oplus C(\bar{p}_i)$. Let K denote the $m \times n'$ matrix such that the i-th row of K is the vector $\boldsymbol{k}_i$. When viewing the matrices U and T column-wise, we see that the receiver holds $\boldsymbol{u}^i = \boldsymbol{k}^i \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}_{[i,i+m-1]} \oplus \bar{\boldsymbol{p}}$ while the sender holds $\boldsymbol{t}^i = \boldsymbol{k}^i \oplus \boldsymbol{x}_{[i,i+m-1]}$. Following the high-level idea described Sect. 2, and specifically the pattern match test of Eq. 2, it is clear that the pattern matches the text $\boldsymbol{x}$ at the i-th position if and only if $\boldsymbol{u}^i = \boldsymbol{t}^i$.

Theorem 1. *The SWiM protocol in Fig. 5 securely computes the WPM functionality (Fig. 4) in semi-honest setting, given the ideal OT and $\mathcal{F}_{\mathsf{peqt}}$ primitives defined Figs. 2 and 3, respectively.*

Proof. The proof of security of our construction is based on the fact that the OT and $\mathcal{F}_{\mathsf{peqt}}$ are secure.

Simulating $\mathcal{S}$. It is easy to argue that the view of the sender $\mathcal{S}$ can be perfectly simulated since the semi-honest $\mathcal{S}$ receives nothing from the protocol.

Simulating $\mathcal{R}$. The view of the receiver $\mathcal{R}$ consists of two kinds of messages: (1) output of the form $\boldsymbol{q}_i$ from the OT primitive in Step 2c, which is equal to $\boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}$ and hence information-theoretically hides $\boldsymbol{x}$; (2) outputs of $\mathcal{F}_{\mathsf{peqt}}$ in step 4b, which correspond exactly to the WPM protocol output itself. Hence both can be perfectly simulated.

Cost. Using OT extension, some initial "base OT" instances are required. These base OTs consist of $O(\kappa^2)$ communication and $O(\kappa)$ exponentiations. Thereafter, any number of OTs can be obtained with communication and computation proportional only to total size of parties' inputs. The computation consists of only *symmetric-key* operations. In our case, there are m OT instances, each on strings of length n', so $O(n'm)$ total communication and symmetric-key operations.

The $\mathcal{F}_{\mathsf{peqt}}$ protocol of [KKRT16] has a statistical security parameter which we denote λ. Specifically, the protocol allows for a false positive (output 1 for input strings which are different) with probability $2^{-\lambda}$. The protocol also uses OT extension, but the base OTs can be shared/reused from the base OTs mentioned above. The amortized cost of an equality test is $448 + \lambda$ bits of communication (using typical parameters) and a constant number of symmetric-key operations.

4.1 Additions, Optimizations

Online/Offline Phase. We briefly describe how the protocol can be modified so that most of the cost can be incurred in an offline phase, before the parties' inputs are known.

First, we can run all OTs in Step 2 of the protocol before the receiver's input $\boldsymbol{p}$ is known, by taking advantage of a well-known technique of Beaver [Bea95]. The following modifications are required: First, the receiver uses a random $\boldsymbol{\pi} \in \{0,1\}^m$ (rather than $\overset{\star}{\boldsymbol{p}}$) as its OT choice bits in Step 2 (note that $\bar{\boldsymbol{p}}$ is not used until Step 3). Later, upon learning $\boldsymbol{p}$, the receiver sends $\boldsymbol{\delta} = \boldsymbol{p} \oplus \boldsymbol{\pi}$ to the sender. The sender sets $\boldsymbol{k}'_i = \boldsymbol{k}_i \oplus \delta_i \cdot \boldsymbol{x}_{[i,i+n'-1]}$. It is easy to see that the receiver holds

$$\boldsymbol{q}_i = \boldsymbol{k}_i \oplus \pi_i \cdot \boldsymbol{x}_{[i,i+n'-1]} = \boldsymbol{k}_i \oplus (\delta_i \oplus \overset{\star}{p}_i) \cdot \boldsymbol{x}_{[i,i+n'-1]} = \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}.$$

In other words, $\boldsymbol{k}'_i$ and $\boldsymbol{q}_i$ satisfy the appropriate condition, now with respect to the receiver's true input $\boldsymbol{p}$. The rest of the protocol continues as usual, with $\boldsymbol{k}'_i$ instead of $\boldsymbol{k}_i$.

There is also a standard Beaver technique for preprocessing OTs before the sender's OT input is known. Applying here naively would require the sender to send online correction strings of total length $O(|\boldsymbol{p}||\boldsymbol{x}|)$ since that is the combined length of all the sender's OT inputs.

Instead, we propose the following technique that is similar in spirit but takes advantage of the fact that the sender's OT inputs are derived from a single $\boldsymbol{x}$ value. The parties run step 1, but with the sender using a random $\boldsymbol{\chi} \in \{0,1\}^n$

instead of the true input $\boldsymbol{x}$ (which is not yet known). After the online phase described above, the sender will have $\boldsymbol{k}'_i$ strings and the receiver will have $\boldsymbol{q}_i = \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\chi}_{[i,i+n'-1]}$. As the sender learns its input $\boldsymbol{x}$, it sends $\boldsymbol{\gamma} = \boldsymbol{x} \oplus \boldsymbol{\chi}$ to the receiver. The receiver can compute

$$\begin{aligned}\boldsymbol{q}'_i \overset{\text{def}}{=} \boldsymbol{q}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\gamma}_{[i,i+n'-1]} &= (\boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\chi}_{[i,i+n'-1]}) \oplus \overset{\star}{p}_i \cdot \boldsymbol{\gamma}_{[i,i+n'-1]} \\ &= \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot (\boldsymbol{\chi}_{[i,i+n'-1]} \oplus \boldsymbol{\gamma}_{[i,i+n'-1]}) \\ &= \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}\end{aligned}$$

In other words, $\boldsymbol{k}'_i$ and $\boldsymbol{q}'_i$ satisfy the appropriate condition, now with respect to the sender's true input $\boldsymbol{x}$. The protocol can proceed, using $\boldsymbol{q}'_i$ instead of $\boldsymbol{q}_i$.

By having precomputation, we are able to shift the bulk of the $O(nm)$ communication to the offline phase. In the online phase, each party only sends a "correction string" whose length is proportional to its input size, followed by the equality tests. Similarly to the standard Beaver's technique, it is easy to see that the resulting protocol is secure, namely that the separation of the offline and online phases can be simulated.

Amortization. In certain multiple-execution scenarios, the cost of our protocol can be further significantly reduced by reusing the OT/PEQT outputs.

First, notice that in SWiM (Fig. 5), the OT step is independent of the non-wildcard characters of the pattern string (i.e., independent of $\bar{\boldsymbol{p}}$). Therefore, if the *positions of wildcards* in the receiver's pattern (i.e., $\overset{\star}{\boldsymbol{p}}$) are the same across several executions, OT in subsequent executions can be implemented as length extension of the OT in the first execution. Further, if additionally the sender's text is the same across the executions (and the only variation is the non-$\star$ pattern), then only the equality tests need to be run in the subsequent executions.

Further, in the PEQT protocol of [KKRT16], the receiver can check his input for equality against a polynomial number sender's inputs at the cost λ per check (vs $4\kappa + \lambda$ for full KKRT PEQT). Indeed, on the KKRT BaRK-OPRF output $(R, S) \leftarrow (F_k(x), k)$, KKRT sender $\mathcal{S}$ can send to receiver $\mathcal{R}$ a set of $\{F_k(y_i)\}$, and R will determine $x = y_i \iff F_k(x) = F_k(y_i)$.

To use this in the amortization, we let the WPM sender play the role of PEQT's receiver. We note that this amortization will reveal whether the WPM receiver has used the same pattern in different instances. Additionally, PEQT receiver learns the comparison output, and so will the WPM sender. Both restrictions may be acceptable in certain scenarios.

Non-binary Alphabets. The protocol extends naturally to alphabets Σ beyond $\Sigma = \{0, 1\}$. Without loss of generality let $\Sigma = \mathbb{Z}_b$ for some b. The receiver holds a pattern $\boldsymbol{p} \in (\Sigma \cup \{\star\})^m$ and will encode the pattern into $\overset{\star}{\boldsymbol{p}} \in \{0,1\}^m$ and $\bar{\boldsymbol{p}} \in \Sigma^m$, as follows:

$\boldsymbol{p}_i$	$\overset{\star}{\boldsymbol{p}}_i$	$\bar{\boldsymbol{p}}_i$
$\star$	1	0
$a \neq \star$	0	a

Consider the corresponding amendment to SWiM (Fig. 5), where the parties hold strings of length m and n, both over the alphabet Σ. The parties still perform m 1-out-of-2 OT, using $\overset{\star}{\boldsymbol{p}}$ as the receiver's choice bits. All other vectors ($\boldsymbol{k}$, $\boldsymbol{q}$, etc.) become vectors over Σ, and the $\oplus$ operation is replaced by component-wise addition mod $|\Sigma|$. Note that the "$\cdot$" operation in the protocol is only used between a *binary* vector $\overset{\star}{\boldsymbol{p}}$ and a Σ-vector, so its meaning can still be taken as component-wise multiplication. Finally, the KKRT PEQT can be naturally amended to support equality tests of non-binary strings, e.g. by translating the strings into binary.

5 SWiM Implementation and Performance

Our SWiM implementation uses code from [KKRT16, Rin, WMK16]. All running times are reported as the average over 10 trials. Our complete implementation is available on https://github.com/osu-crypto/PatternMatching.

5.1 Experimental Performance: Comparison with Prior Work

We compare our prototype to the state-of-art WPM protocols [BEDM+12, YSK+14]. While the implementations [BEDM+12, YSK+14] are not publicly available, [BEDM+12] reports experimental numbers. Further, as we discussed in Sect. 1.1, [YSK+14] numbers can be indirectly estimated to be around $2-2.5\times$ faster than 5PM. We give detailed comparisons to 5PM protocol [BEDM+12]; comparison to other works can be appropriately derived.

Runtime Comparison. For the most direct comparison, we matched the test system's computational performance to that of [BEDM+12], as reported in their Table 13. Since 5PM [BEDM+12] experiments were performed on Intel dual quad-core 2.93 GHz Linux machine with 8 GB RAM, we evaluate our protocol on a virtual Linux machine with 8GB RAM and 2 cores (the host machine is Intel Core i7 2.60 GHz with 12 GB RAM). Table 2 presents the running time of our protocol compared with 5PM [BEDM+12]. For our protocol, we report both the total running time and the online time. We use $\lceil \log(\Sigma) \rceil$ bits to encode the text and pattern alphabet into binary alphabet.

When comparing the two protocols, we find that the total running time of SWiM is significantly less than that of the prior works, requiring 1.96 s to perform a wildcard pattern matching with 4-symbol alphabet for text size $n = 10^5$ and pattern size $m = 10^3$. This is a $155\times$ improvement in running time compared to 5PM [BEDM+12] which used 1024-bit key length. When considering 5PM [BEDM+12] with 2048-bit key length (which better corresponds to our security level), our improvement is $499\times$.

SWiM is optimized for the typical use case, where the length of the text is greater than that of the pattern. If this doesn't hold (indeed, an unusual setting for the motivating examples we consider), our performance improvement is moderate. For instance when $m = n = 10^3$, our protocol requires 0.61 s. Using the same parameters, the protocol of [BEDM+12] results in an execution time of 1.39 s. The moderate 2× improvement is due to the constant-cost overheads of OT extension and PEQT, which do not pay off without amortization in a larger execution. Even in these cases, our protocol achieves great improvement in the online phase (e.g., running in just 3ms for $m = n = 10^3$).

Bandwidth Comparison. We calculate the bandwidth requirements of our protocol on the range of the length text $n \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}\}$ and the length pattern $m \in \{2^8, 2^{10}, 2^{12}, 2^{14}\}$, for the binary alphabet. For comparison, we calculate the communication cost of 5PM [BEDM+12], for the same parameters. 5PM bandwidth requirements is independent on the length of pattern, and is roughly $(n + 2)\kappa$ bits. 5PM protocol relies on public-key operations, and needs 1024–2048-bit key lengths.

Table 4 reports the communication overhead of the protocols. Our protocol requires less communication for smaller pattern sizes. Concretely, for $n = 2^{22}$ and $m = 2^8$, our protocol requires 392.1 MB of communication, a 1.37 to 2.7× improvement compared to 5PM [BEDM+12]. Increasing the pattern length to $m = 2^{12}$ the communication cost of 5PM protocol (at a great performance penalty!) becomes preferable to ours, since their bandwidth is independent of the length of pattern. Note, the bulk of the communication cost in our protocol is OT extension in the offline phase.

We note that Table 4 does not show off SWiM algorithmic improvement for non-binary alphabet, which reduces the number of OT calls. For larger Σ, we (but not other approaches, to our knowledge) get factor $\approx \log |\Sigma|$ bandwidth reduction in the *offline* phase over the simple mapping of Σ to a binary alphabet.

5.2 SWiM Performance at Scale: Experiments and Discussion

To understand the scalability of SWiM, we evaluate it on the range of the text/pattern lengths $n \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}, 2^{24}\}$, $m \in \{2^8, 2^{10}, 2^{12}, 2^{14}\}$, for the binary alphabet. We report SWiM detailed performance results in Table 3, showing total running time and online time in both LAN and WAN settings.

This set of experiments was ran on a larger machine (a single server with 2x 36-core Intel Xeon 2.30 GHz CPU and 256 GB of RAM), whose resources were carefully limited by us to provide a good understanding of the performance. Specifically, we ran each party *single threaded*, both on the same machine, communicating via `localhost` network. We simulated a network connection using the Linux `tc` command. We configured LAN setting with 0.02ms round-trip latency, 10 Gbps network bandwidth, and WAN setting with a simulated 40ms round-trip latency, 400 Mbps network bandwidth.

The step of forming the matrices in SWiM is relatively costly. We push it into the preprocessing phase, which will include creating OT matrices and performing

Table 2. 5PM vs SWiM. Comparison of 5PM and SWiM of the total runtime (in seconds) for wildcard pattern matching of length n, the pattern of length m, and the alphabets of sizes 4 (DNA). In SWiM, the online time is presented in parenthesis. Best results marked in bold. SWiM experiment ran on Intel Core i7 2.60 GHz with 8 GB RAM. 5PM timings reported on comparable hardware.

Protocol	Bit key length	Pattern length m	Text length n		
			10^3	10^4	10^5
5PM	1024	10	0.42	4.08	40.43
		10^2	0.67	6.81	64.76
		10^3	0.39	29.15	304.53
	2048	10	1.50	14.18	140.52
		10^2	2.27	22.37	216.27
		10^3	1.39	92.29	978.94
SWiM	128	10	**0.29 (0.006)**	**0.36 (0.03)**	**0.76 (0.32)**
		10^2	**0.37 (0.005)**	**0.62 (0.09)**	**1.82 (0.49)**
		10^3	**0.61 (0.003)**	**0.73 (0.04)**	**1.96 (0.39)**

Table 3. SWiM scaling. Total running time and online time (in parenthesis) in second of SWiM for the text of length n, the pattern of length m, binary alphabet. The results mentioned in the discussion is marked in bold. Experiment ran sender and receiver *single-threaded* on 2x 36-core Intel Xeon 2.30 GHz CPU and 256 GB of RAM.

Setting	Pattern length m	Text length n				
		2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
LAN	2^8	**0.21 (0.04)**	0.40 (0.15)	0.94 (0.48)	4.07 (2.78)	16.11 (11.38)
	2^{10}	0.24 (0.03)	0.48 (0.12)	1.41 (0.57)	5.21 (2.38)	20.61 (10.00)
	2^{12}	0.37 (0.03)	0.97 (0.17)	3.40 (0.78)	12.92 (3.34)	**51.88 (14.44)**
	2^{14}	1.02 (0.07)	3.91 (0.37)	15.14 (1.66)	**60.10 (6.46)**	246.24 (43.51)
WAN	2^8	**1.04 (0.40)**	1.90 (1.02)	5.10 (3.10)	17.84 (12.04)	70.45 (48.43)
	2^{10}	1.28 (0.40)	2.81 (0.95)	8.62 (3.04)	31.29 (12.00)	127.92 (48.08)
	2^{12}	2.28 (0.36)	6.46 (0.96)	21.61 (3.17)	84.52 (12.48)	**363.06 (50.15)**
	2^{14}	6.16 (0.34)	22.24 (1.07)	85.98 (3.87)	318.23 (15.45)	1,382.03 (65.86)

Table 4. Bandwidth calculation of communication (in MB) for wildcard pattern matching of text length n, pattern length m, binary alphabet. In SWiM, the online communication cost is presented in parenthesis. Compared to 5PM, best results marked in bold.

Protocol	Bit key length	Pattern length m	Text length n			
			2^{16}	2^{18}	2^{20}	2^{22}
5PM	1024	$\{2^8, 2^{10}, 2^{12}, 2^{14}\}$	8.4	33.5	134.2	536.9
	2048	$\{2^8, 2^{10}, 2^{12}, 2^{14}\}$	16.8	67.1	268.4	1073.7
SWiM	128	2^8	**7.6(3.9)**	**25.9(16.1)**	**99.2(64.1)**	**392.1(256.4)**
		2^{10}	**13.7(3.9)**	**50.9(15.9)**	**199.7(64.1)**	**794.6(256.3)**
		2^{12}	36.7(3.7)	149.5(15.8)	600.2(63.8)	2403.1(256.1)
		2^{14}	105.2(2.9)	519.9(15.1)	2178.6(63.1)	8813.3(255.4)

the matrix transposition. Our experiments show that the offline phase takes 60−90% of the total running time. For instance, with text size $n = 2^{22}$ and pattern size $m = 2^{14}$ our overall running time is 60.10 s with an offline phase of 53.64 s, a 89% of the overall cost.

We find that SWiM scales well in the experiments. For text size $n = 2^{16}$ and pattern size $m = 2^8$, our protocol takes only 0.21 s in which 0.04 s is for online time. When increasing the lengths to $n = 2^{24}$ and $m = 2^{12}$, we see that our protocol requires roughly 52 s in total.

When evaluating our implementation in the WAN setting, we still have a fast online phase due to the fact that OTs can be precomputed in the offline phase. For $n = 2^{24}$ and $m = 2^{12}$, we obtain an overall running time of 363.06 s and an online time of 50.15 s which contains only 13% of the total cost. For the small text and pattern, the protocol requires only a few seconds. With $n = 2^{16}$ and $m = 2^8$, our protocol takes an overall running time of 1.04 s with the online phase requiring 0.4 s.

Acknowledgments. The first author was supported by Office of Naval Research (ONR) contract number N00014-14-C-0113. The second and third authors were supported by NSF awards #1149647 and #1617197.

References

[ALSZ13] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM CCS, vol. 13 (2013)

[AWY15] Abboud, A., Williams, R.R., Yu, H.: More applications of the polynomial method to algorithm design. In: Indyk, P. (ed) 26th SODA, pp. 218–230. ACM-SIAM, January 2015

[BCP13] Bringer, J., Chabanne, H., Patey, A.: SHADE: secure hamming distance computation from oblivious transfer. In: Adams, A.A., Brenner, M., Smith, M. (eds.) FC 2013. LNCS, vol. 7862, pp. 164–176. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41320-9_11

[Bea95] Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_8

[Bea96] Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: 28th ACM STOC (1996)

[BEDM+12] Baron, J., El Defrawy, K., Minkovich, K., Ostrovsky, R., Tressler, E.: 5PM: secure pattern matching. In: SCN 2012 (2012)

[BGVV14] Bille, P., Gørtz, I.L., Vildhøj, H.W., Vind, S.: String indexing for patterns with wildcards. Theor. Comput. Syst. **55**, 41 (2014)

[BI14] Barton, C., Iliopoulos, C.S.: On the average-case complexity of pattern matching with wildcards. CoRR, abs/1407.0950 (2014)

[BST01] Boudot, F., Schoenmakers, B., Traoréa, J.: A fair and efficient solution to the socialist millionaires' problem. Discrete Appl. Math. **111**, 23–36 (2001)

[ÇCL+17] Çetin, G.S., Chen, H., Laine, K., Lauter, K., Rindal, P., Xia, Y.: Private queries on encrypted genomic data. BMC Med. Genomics **10**, 45 (2017)

[CEPR09] Clifford, R., Efremenko, K., Porat, E., Rothschild, A.: From coding theory to efficient pattern matching. In: 20th SODA (2009)
[CH02] Cole, R., Hariharan, R.: Verifying candidate matches in sparse and wildcard matching. In: 34th ACM STOC (2002)
[CLI07] Simple deterministic wildcard matching. Inf. Process. Lett. (2007)
[CS15] Chase, M., Shen, E.: Substring-searchable symmetric encryption. In: PoPETs (2015)
[CWZ+06] Chen, G., Wu, X., Zhu, X., Arslan, A.N., He, Y.: Efficient string matching with wildcards and length constraints. Knowl. Inf. Syst. **10**, 399–419 (2006)
[DCFT13] De Cristofaro, E., Faber, S., Tsudik, G.: Secure genomic testing with size- and position-hiding private substring matching. In: WPES 2013 (2013)
[DF13] El Defrawy, K., Faber, S.: Blindfolded data search via secure pattern matching. Computer **46**, 68–75 (2013)
[FHV13] Faust, S., Hazay, C., Venturi, D.: Outsourced pattern matching. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 545–556. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_48
[FNP04] Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_1
[FNW96] Fagin, R., Naor, M., Winkler, P.: Comparing information without leaking it. Commun. ACM **39**, 77–85 (1996)
[Fri09] Frikken, K.B.: Practical private DNA string searching and matching through efficient oblivious automata evaluation. In: Gudes, E., Vaidya, J. (eds.) DBSec 2009. LNCS, vol. 5645, pp. 81–94. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03007-9_6
[GHS10] Gennaro, R., Hazay, C., Sorensen, J.S.: Text search protocols with simulation based security. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 332–350. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_20
[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: 19th ACM STOC (1987)
[HHD11] Hoffmann, H., Howard, M.D., Daily, M.J.: Fast pattern matching with time-delay neural networks. In: The 2011 International Joint Conference on Neural Networks, pp. 2424–2429, July 2011
[HL08] Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_10
[HT10] Hazay, C., Toft, T.: Computationally secure pattern matching in the presence of malicious adversaries. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 195–212. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_12
[HT14] Hazay, C., Toft, T.: Computationally secure pattern matching in the presence of malicious adversaries. J. Cryptol. **27**, 358–395 (2014)
[IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9

[JKS08] Jha, S., Kruger, L., Shmatikov, V.: Towards practical privacy for genomic computation. In: 2008 IEEE Symposium on Security and Privacy (2008)

[KK13] Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_4

[KKRT16] Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: ACM CCS 16 (2016)

[KM10] Katz, J., Malka, L.: Secure text processing with applications to private DNA matching. In: ACM CCS, vol. 10 (2010)

[KMP+17] Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: ACM CCS, vol. 17 (2017)

[Lip03] Lipmaa, H.: Verifiable homomorphic oblivious transfer and private equality test. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 416–433. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40061-5_27

[MNSS12] Mohassel, P., Niksefat, S., Sadeghian, S., Sadeghiyan, B.: An efficient protocol for oblivious DFA evaluation and applications. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 398–415. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27954-6_25

[NN10] Namjoshi, K., Narlikar, G.: Robust and fast pattern matching for intrusion detection. In: Proceedings of the 29th Conference on Information Communications, INFOCOM 2010 (2010)

[NP99] Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, STOC 1999 (1999)

[OPJM10] Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: SCiFI - a system for secure face identification. In: 2010 IEEE Symposium on Security and Privacy (2010)

[PSZ14] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium (USENIX Security 14) (2014)

[Rab05] Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005)

[Rin] Rindal, P.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe

[SK17] Saha, T.K., Koshiba, T.: An enhancement of privacy-preserving wildcards pattern matching. In: Cuppens, F., Wang, L., Cuppens-Boulahia, N., Tawbi, N., Garcia-Alfaro, J. (eds.) FPS 2016. LNCS, vol. 10128, pp. 145–160. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51966-1_10

[SOF10] Pattern matching with don't cares and few errors. J. Comput. Syst. Sci. (2010)

[SSSS15] Saikkonen, R., Sippu, S., Soisalon-Soininen, E.: Experimental analysis of an online dictionary matching algorithm for regular expressions with gaps. In: Bampis, E. (ed.) SEA 2015. LNCS, vol. 9125, pp. 327–338. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20086-6_25

[Tha11] Thachuk, C.: Succincter Text Indexing with Wildcards (2011)

[TPKC07] Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient DNA searching through oblivious automata. In: ACM CCS 07 (2007)

[Ver11] Vergnaud, D.: Efficient and secure generalized pattern matching via fast Fourier transform. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 41–58. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21969-6_3

[WJW+15] Wang, D., Jia, X., Wang, C., Yang, K., Fu, S., Xu, M.: Generalized pattern matching string search on encrypted data in cloud systems. In: INFOCOM (2015)

[WMK16] Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: efficient MultiParty computation toolkit (2016). https://github.com/emp-toolkit

[WZX17] Wei, X., Zhao, M., Xu, Q.: Efficient and secure outsourced approximate pattern matching protocol. Soft Comput. **22**, 1195–1187 (2017)

[Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS (1986)

[YSK+13] Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: Secure pattern matching using somewhat homomorphic encryption. In: CCSW 2013 (2013)

[YSK+14] Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: Privacy-preserving wildcards pattern matching using symmetric somewhat homomorphic encryption. In: Susilo, W., Mu, Y. (eds.) ACISP 2014. LNCS, vol. 8544, pp. 338–353. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08344-5_22

Attacks

Attacks Against GSMA's M2M Remote Provisioning (Short Paper)

Maxime Meyer[1,2(✉)], Elizabeth A. Quaglia[2], and Ben Smyth[3]

[1] Vade Secure Technology Inc., Paris, France
maxime.meyer@telecom-bretagne.eu
[2] Information Security Group - Royal Holloway, University of London, London, UK
[3] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg City, Luxembourg

Abstract. GSMA is developing and standardizing specifications for embedded SIM cards with remote provisioning, called eUICCs, which are expected to revolutionize the cellular network subscription model. We study GSMA's "Remote Provisioning Architecture for Embedded UICC" specification, which focuses on M2M devices, and we analyze the security of remote provisioning. Our analysis reveals weaknesses in the specification that would result in eUICCs being vulnerable to attacks: we demonstrate how a network adversary can exhaust an eUICC's memory, and we identify three classes of attacks by malicious insiders that prevent service. We disclosed our findings to GSMA; GSMA confirmed the validity of these attacks and acknowledged their potential to disrupt the cellular industry. We propose fixes, which GSMA is incorporating into its specification. Thus, we improve security of next generation telecommunication networks.

1 Introduction

Machine to Machine (M2M) devices (i.e., machines communicating together without human intervention) are ubiquitous. Some of these devices communicate using cellular networks. To access such networks, a device authenticates using an embedded SIM, which is issued by a Mobile Network Operator (MNO). Authentication is established with the AKA protocol [4]. AKA algorithms and keys are embedded in SIMs, which physically ensures their confidentiality and integrity. Limitations of SIMs include being neither re-programmable (hence, restricted to a *single* subscription during their lifetime) nor remotely personalizable (hence, installation requires physical access).

ETSI [5] specified requirements for re-programmable and remotely personalizable embedded SIMs to overcome the aforementioned limitations. Following ETSI's specification, industrial researchers, e.g., [2,6,16], and academic researchers, e.g., [19], proposed remote provisioning schemes. Moreover, building upon ETSI's specification and GlobalPlatform's smart card standard [7], GSMA

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 243–252, 2018.
https://doi.org/10.1007/978-3-662-58387-6_13

released a specification for a next generation SIM, namely, an *embedded UICC*, which supports multiple operators simultaneously. Profiles are remotely provisioned and installed into eUICCs. For M2M devices, remote provisioning protocols and management mechanisms are described by GSMA's *"Remote Provisioning Architecture for Embedded UICC"* specification [11][1]. (We adopt GSMA's terminology for consistency with their specification.)

Unlike SIMs, which are restricted to a single subscription from a single operator, eUICCs support multiple subscriptions and multiple operators. Subscriptions are defined by *Profiles* (P_{MNO} in Fig. 1), which encapsulate subscription data. Each profile is stored inside a separate *Security Domain* (ISD-P in Fig. 1) and an interface (ISD-R in Fig. 1) is defined for communication between ISD-Ps and remote entities. Authentication of that communication is managed by an application (ECASD in Fig. 1). Internal communication between eUICC components exploits the underlying GlobalPlatform framework, which is reliant on the Application Protocol Data Unit (APDU) message format.

Remote provisioning is a core aspect of GSMA's specification. Motivated by business cases [10, Sect. 3], GSMA introduces Subscription Managers which act as intermediaries between operators and eUICCs. Subscription managers are separated into two roles: Data Preparation roles (SM-DPs) oversee eUICC profile formatting and installation, they may be controlled by an individual operator; and Secure Routing roles (SM-SRs) oversee remote eUICC management, they may be operated by an independent organization, e.g., a regulator.

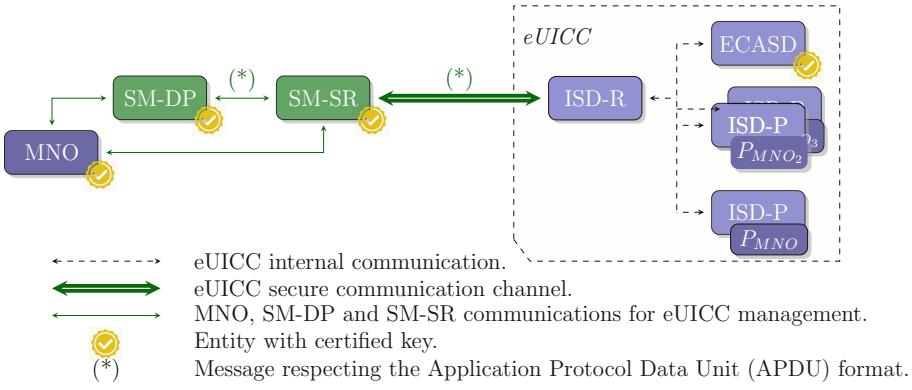


Fig. 1. Remote provisioning interfaces and communication channels, adapted from [17]

GSMA is promoting their specification for standardization [9,22]. Any weakness or flaw in the specification or subsequent standard could have disastrous consequences on the secure deployment of eUICCs. As such, security of remote

[1] Meyer, Quaglia and Smyth provide a detailed introduction to GSMA's specification [17].

provisioning must be analyzed. Indeed, finding and fixing specification flaws is paramount, because the cost of fixing problems increases exponentially once production commences.

Contribution and Structure. We study version 3.1 of GSMA's M2M remote provisioning specification and present the first analysis of remote provisioning. Our analysis reveals flaws which would make eUICCs vulnerable to attacks and we present fixes to eliminate those flaws. We proceed as follows: Sect. 2 describes creation of a profile and its associated security domain. Sect. 3 presents a memory exhaustion attack against eUICCs. The attack works by dropping an acknowledgement message sent during ISD-P creation, which causes the creation of an empty and undeletable ISD-P, and can be repeated to exhaust an eUICC's memory. Section 4 presents attacks by malicious insiders that exploit remote management messages to prevent operators from installing new profiles on eUICCs. Section 5 shows how a malicious operator can lock an eUICC to their profile and block other operators. Section 6 documents our disclosure of the aforementioned attacks to GSMA and explains how GSMA is revising its specification.

2 Preliminaries

2.1 Profile Download and Installation

GSMA's eUICC specification defines a remote provisioning procedure, called `Download&Install`, which transmits profiles from an operator to an eUICC, and installs them. The procedure can be summarized as follows (see Fig. 2):

1. An operator initiates the process with a `DownloadProfile` request to an SM-DP containing a profile description (e.g., profile size, network capabilities).
2. The SM-DP uses the `GetEIS` function to obtain data about the target eUICC, including mutable (e.g., remaining memory, SM-SR identifier, installed profiles description) and immutable (e.g., production date, platform version) information about the eUICC. The SM-DP checks the validity of the profile description against the characteristics of the eUICC and creates a profile according to the operator's profile description.
3. The SM-DP makes a `CreateISDP` request to the SM-SR responsible for the target eUICC (procedure `CreateISDP` detailed in Sect. 2.2). The SM-SR receives the request (labelled 3a in Fig. 2) and creates an ISD-P on the eUICC to hold the profile (labelled 3b).
4. The SM-DP establishes a secure channel with the ISD-P (labelled 4a), and sends the profile to the ISD-P over that channel (labelled 4b).
5. The ISD-P installs the profile, and relays acknowledgments to the operator.

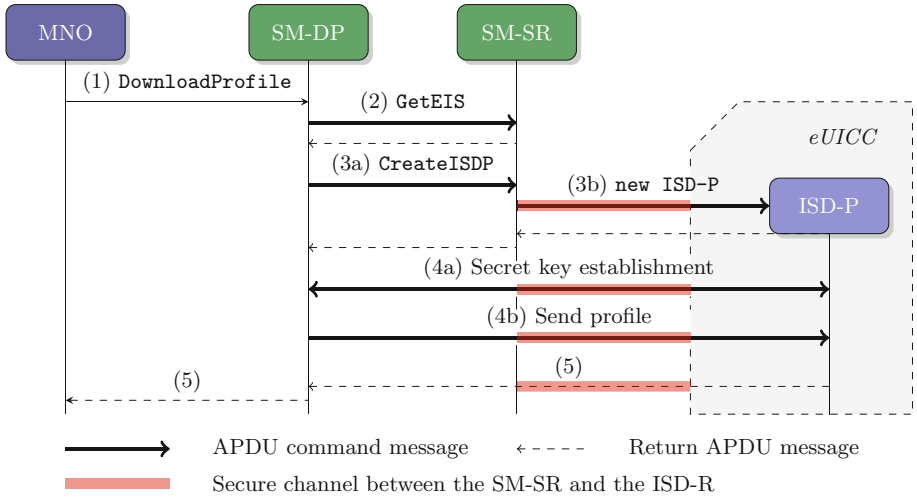


Fig. 2. Profile download and installation flow (high level)

2.2 ISD-P Creation

ISD-P creation (Step 3 in Sect. 2.1) precedes the upload of the profile onto the eUICC. At the end of this phase, the profile container's unique application identifier (`ISD-P AID`) has been set, and memory has been reserved for the future profile onto the eUICC. The creation proceeds as follows (see Fig. 3):

1. The SM-DP sends a `CreateISDP` request to the SM-SR containing the following payload: the identifiers of the target eUICC and the operator that requested profile creation, along with memory requirements.
2. The SM-SR establishes a secure channel with the eUICC, via its ISD-R interface.
3. The SM-SR instructs the ISD-R to create an ISD-P, specifying its creation parameters (e.g., ISD-P identifier (AID), profile size, etc.).
4. The command is processed by the smart card (labelled 4a in Fig. 3) which creates the ISD-P (4b) and returns (4c).
5. The ISD-R reports the success of the ISD-P creation to the SM-SR.
6. The SM-SR updates the eUICC Information Set (EIS) file.
7. Finally, the SM-SR returns the ISD-P identifier to the SM-DP.

3 Memory Exhaustion Attack by Network Adversary

We analyzed the security of GSMA's remote provisioning protocol [11] by considering potential adversaries and their motivations. In this section, we consider a network adversary, i.e., an adversary that is able to read, modify and delete messages sent over wireless networks. In practice, such an adversary can intercept a signal simply by being close enough to the signal transmitter or receiver.

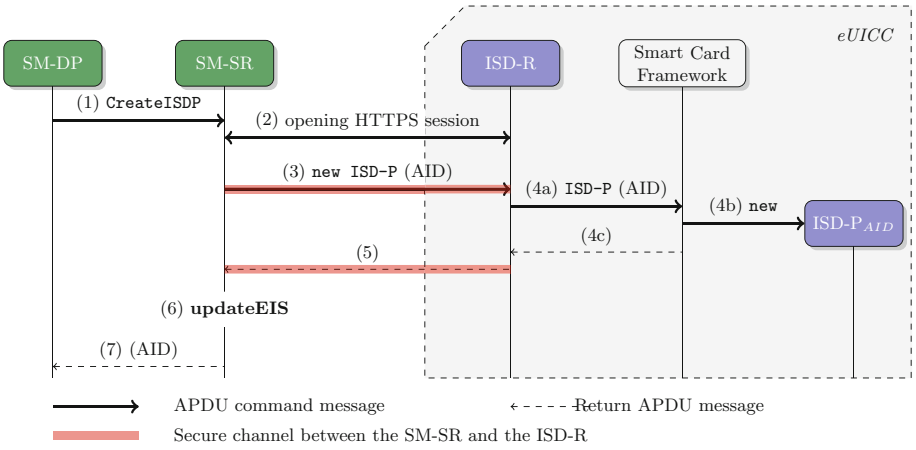


Fig. 3. ISD-P creation flow

3.1 Memory Exhaustion Attack

It is possible to launch an attack that fills part of an eUICC's memory with an empty ISD-P by exploiting error handling during ISD-P creation, and the ISD-P deletion mechanism. Moreover, the eUICC's memory could be exhausted by repeating the attack. Indeed, an adversary could drop the ISD-R's response to the SM-SR (see (5) in Fig. 3) as is common in denial of service attacks [26,27] (The adversary can identify and drop the response, even when it is encrypted, using truncation attacks [3,23]). As a result of the dropped message, the SM-SR, unaware of the ISP-D creation status, cannot update the `EIS` file, and after waiting some time, sends a timeout response message to the SM-DP. The ISD-P created remains on the card, neither associated to an SM-DP nor operator, thus the ISD-P is *orphaned* and memory space on the eUICC has been reserved for its profile.

Recovery from this attack by deleting the created ISD-P is not possible, because deletion of an ISD-P and its profile is restricted to operators or SM-DPs and requires the ISD-P's identifier. Restricting deletion to operators and SM-DPs was motivated by a operator's requirements. An exceptional procedure called *Master Delete* is defined in the specification to allow an SM-SR to delete a profile correctly installed on an eUICC for which the operator owning the profile has given its approval for deletion and for which the subscription period is elapsed [11, Sect 3.10]. Consequently, the master delete procedure cannot be applied. In fact, there is no mechanism defined in the specifications for the SM-SR or the eUICC to delete orphaned ISD-Ps. If the attack is repeated to exhaust the eUICC's memory, only profiles existing on the eUICC before the attack can be used later.

This attack causes financial loss, as it prevents operators from providing service. Moreover, recovery is impossible and any trace of the attack is minimal.

An operator, with a profile on an eUICC, could, for example, collude with a network adversary and deliberately fill the eUICC memory.

Countermeasure. The attack can be prevented by creating a mechanism on the card to manage ISD-P creation. Once an ISD-P is created, the mechanism awaits the next logical instruction of the `DownloadProfile` process, i.e., the APDU command for the key exchange between the SM-DP and the ISD-P. If the awaited instruction is not received on time, the ISD-P is automatically deleted by this mechanism and a notification is sent to the SM-SR. As such, even if the notification is dropped too, the orphaned ISD-P is deleted. This mechanism could be implemented as an extension of the GlobalPlatform framework.

4 Payload Exploitation by Malicious Insiders

GSMA does not formally define the trust model between the different network entities. In this section, we exploit this and we present two attacks that can be performed by a malicious entity, behaving as a *malicious insider* [14,21]. Such an adversary is dishonest, perhaps due to greed. Malicious entities have the capabilities of their honest counterparts, plus they will try to modify the protocol or the messages sent without being suspected of being dishonest as they could face sanctions otherwise [8,24], resulting in what is considered as low cost attacks (see [1] for further information on such attacks).

4.1 Undersizing Memory Attack

A malicious SM-SR could, after receiving a `GetEIS` request from the SM-DP (Sect. 2.1), return the EIS file with the value of field `remainingMemory` set to a random value under the minimal size of a profile (This cannot be detected because the field is not signed). By doing so, the SM-SR prevents an SM-DP from creating an ISD-P required for uploading a new profile on the eUICC, because the `Download&Install` process would halt, and the eUICC would be considered by the SM-DP as unable to receive a new profile. Therefore, an SM-SR can deny operators from installing profiles on an eUICC.

This attack is feasible as mutable fields from the EIS file, including the `remainingMemory` field, are not signed.[2] Such an attack is likely to be detected if the eUICC has been recently created but, for an old eUICC, the SM-SR will likely probably not be detected.

Countermeasure. This attack can be prevented by having eUICCs sign the values sent back to the SM-SR during an `AuditEIS` request. Such a request updates the value of an eUICC's mutable characteristics present in the EIS file, ensuring the SM-DP of their integrity. To prevent replay attacks, the signature should also contain a timestamp.

[2] Immutable characteristics of eUICCs, set at manufacture time, are signed by the manufacturer and stored, with other mutable information, into the *EIS* file. The *EIS* file is issued by the manufacturer to the first SM-SR responsible for the eUICC.

4.2 Inflated Profile Attack

A malicious operator could request, during the `DownloadProfile` protocol (Sect. 2.1), a profile almost exhausting the eUICC's remaining memory (The operator can learn how much memory is available from using the getEIS function). This prevents other operators from installing profiles on the eUICC. This attack can similarly be initiated by an SM-DP during the `createISDP` request.

Countermeasure. This attack can be avoided by defining a profile size upper bound. During a profile creation, the operator's profile request would be verified by the subscription managers.[3]

5 Locking Profile Attacks by Network Operators

5.1 Profile Policy Rules and eUICC Lock

GSMA's specification defines policy rules for managing the life cycle of profiles. These rules are initialized by the operator during profile creation, and are stored inside each profile. An unsynchronized copy of these rules, maintained by the operator, is in the EIS file stored by the SM-SR. They specify whether a profile can be disabled, can be deleted, or should be deleted once it is disabled. A profile's rules can only be modified when the profile is enabled by the operator owning the profile.

The policy rule `CannotBeDisabled` locks an eUICC to a profile, forcing the device to connect to a specific network. It is a feature of the existing subscription model [25], typically used to subsidize subscriptions. However, contrarily to eUICCs, for 4G networks, once unlocked, a device cannot be locked again.

At a high level, rule `CannotBeDisabled` is either *true* or *false* for the enabled profile. We show that this rule introduces a weakness that can result in an eUICC being locked to an undesirable operator's profile, without regard for the initial value of the rule. We demonstrate that a malicious operator can launch an attack when rule `CannotBeDisabled` is false (Sect. 5.2.1) and that an opportunistic operator can take advantage of its position when the rule is true (Sect. 5.2.2).

5.2 Locking Profile Attacks

5.2.1 Rule `CannotBeDisabled` Is *False*

Suppose all of an eUICC's profiles have set the policy rule `CannotBeDisabled` to *false*. Further suppose a malicious operator is interested in blocking other operators' profiles. For this, the malicious operator installs its profile (Sect. 2.1), enables it and sets policy rule `CannotBeDisabled` to *true*. This disables the enabled profile, which is possible given its policy rules. The eUICC is locked to the malicious operator's profile which cannot be disabled and, consequently, cannot be deleted. Thus, even upon receiving a notification from the SM-SR file, the operator owning the previously enabled profile will be unable to re-enable it. The following examples present scenarios whereby eUICCs might be locked:

[3] The SM-SR should perform the check to prevent a similar attack by the SM-DP.

- **Cyberwarfare.** Assuming conflict between countries, one country could use a national operator to remotely attack the other country's eUICCs [20].
- **Hackers.** Hackers might steal valid certificates, as previously observed [15, 18]. Thus, it is feasible for hackers to pose as insiders.
- **Supply chain attack.** Assuming devices are powered-on once manufactured, and then shipped to their destination, and further assuming that devices are passing along the border of a country where operators have an aggressive market strategy, one operator could install a profile on all devices inside the container. Such attacks could also occur while devices are in production or in storage.

5.2.2 Rule `CannotBeDisabled` Is *True*

An issue might arise when a subscriber wants the operator to unlock devices. Indeed, device owners are likely to initiate the remote unlocking of eUICCs. This setting, where the client ask the operator to unlock devices, is problematic in the presence of an opportunistic operator. Such an operator can delay the unlocking process, thus preventing other operator's from enabling their profile on the locked eUICC. Furthermore, the locking profile cannot be deleted without the operator's approval.

Countermeasure. We present several countermeasures that can be combined, if desired. First, a mechanism to automatically unlock the eUICC, once a lock expires. Secondly, specifying an upper bound on the locking period (e.g., two years), to prevent abuse. Finally, permitting locking only once during the life of an eUICC. This can be achieved by using a counter set to a specific value once a lock is used on a profile.

6 GSMA Response

We reported our findings to GSMA under their Coordinated Vulnerability Disclosure Programme. GSMA's experts investigated our findings, acknowledged our attacks and confirmed their ability to impact the mobile industry. GSMA publicly recognized our work and contribution by adding our names to their Hall of Fame. Moreover, GSMA is working with us to incorporate our fixes into their specification. So far, GSMA has released an updated specification [13], which includes the fix described in Sect. 3 (see Sect. 3.1.1_(7) of the updated specification). They have also released a document detailing non-technical trust model and dependencies between the different parties needed for remote provisioning [12], which covers attacks initiated by malicious insiders by claiming that certification will solve the problem. Furthermore, GSMA is still integrating technical countermeasures, including our suggestions, to appear in the next releases of the specification.

7 Conclusion

GSMA is striving towards standardization of remotely provisioned, embedded SIMs. Its efforts have resulted in specifications for remote provisioning, in particular, for M2M devices. This evolution towards next generation telecommunications is exciting, but not without risk. Indeed, we have studied release 3.1 of GSMA's specification and discovered that the proposed evolution is insecure. More issues might well exist and it is crucial that the specification is studied further to ensure security of next generation telecommunications, ideally resulting in formal security proofs.

Acknowledgments. This work was largely conducted at Huawei's Mathematical and Algorithmic Sciences Lab in France.

References

1. Anderson, R., Kuhn, M.: Low cost attacks on tamper resistant devices. In: Christianson, B., Crispo, B., Lomas, M., Roe, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 125–136. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0028165
2. Berard, X., Gachon, D.: Method for remotely delivering a full subscription profile to a UICC over IP, November 2013. US Patent App. 13/991,846
3. Berbecaru, D., Lioy, A.: On the robustness of applications based on the SSL and TLS security protocols. In: Lopez, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 248–264. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73408-6_18
4. Blom, R., Norrman, K., Naslund, M., Rommer, S., Sahlin, B.: Security in the evolved packet system. Technical report, February 2010
5. ETSI: Smart Cards; Embedded UICC; Requirements Specification (V. 12.0.0). Technical Specification 103 383, September 2013
6. Girard, P., Proust, P.: Method for managing content on a secure element connected to an equipment, November 2013. US Patent App. 13/991,823
7. GlobalPlatform: Card Specification (V. 2.3). Technical Specification GPC_SPE_034, October 2015
8. Gow, D.: Telefónica hit by record €152m anti-trust fine, July 2007. goo.gl/Dvx6kk. Accessed 6 Dec 2016
9. GSMA: GSMA announces mobile industry initiative to create a global remote provisioning specification for consumer devices, March 2015
10. GSMA: Business Process for Remote SIM Provisioning in M2M (V.1.0). Technical Specification CLP.05, February 2015
11. GSMA: Remote Provisioning Architecture for Embedded UICC (V. 3.1). Technical Specification SGP.02, May 2016
12. GSMA: M2M IoT Trust Model (V. 1.0). Technical Specification SGP.15, November 2017
13. GSMA: Remote Provisioning Architecture for Embedded UICC (V. 3.2). Technical Specification SGP.02, June 2017
14. Jiang, S., Smith, S., Minami, K.: Securing web servers against insider attack. In: Annual Computer Security Applications Conference, pp. 265–276. IEEE (2001)

15. Langley, A.: Further improving digital certificate security, December 2013. goo.gl/kRHHD6. Accessed 16 Jan 2017
16. Merrien, L., Berard, X., Gachon, D.: Method for transmitting a SIM application of a first terminal to a second terminal, May 2014. US Patent App. 13/991,542
17. Meyer, M., Quaglia, E.A., Smyth, B.: Overview of GSMA remote provisioning specification (2017). https://bensmyth.com/publications/2017-eUICC-overview/
18. Microsoft: Fraudulent Digital Certificates Could Allow Spoofing, August 2011. goo.gl/bLbSQM. Accessed 16 Jan 2017
19. Park, J., Baek, K., Kang, C.: Secure profile provisioning architecture for embedded UICC. In: International Conference on Availability, Reliability and Security, pp. 297–303. IEEE (2013)
20. Schneier, B.: Cyberwar, June 2007. goo.gl/SJW3oU. Accessed 12 Oct 2016
21. Schultz, E.E.: A framework for understanding and predicting insider attacks. Comput. Secur. **21**(6), 526–531 (2002)
22. Sierra Wireless: The eUICC opportunity: harness the power of IoT eSIMS. White paper (2017)
23. Smyth, B., Pironti, A.: Truncating TLS connections to violate beliefs in web applications. In: USENIX Workshop on Offensive Technologies. USENIX Association (2013). see also INRIA tech. rep. hal-01102013 (2015)
24. Thomas, D.: France hits orange with & €350m antitrust fine, December 2015. goo.gl/B8z1Xf. Accessed 6 Dec 2016
25. Vermeulen, J.: Why it is legal for FNB to SIM-lock its smartphones, September 2016. https://goo.gl/xbX5zn. Accessed 16 Jan 2017
26. Wood, A.D., Stankovic, J.A.: Denial of service in sensor networks. Computer **35**(10), 54–62 (2002)
27. Xie, L., Zhu, S.: Message dropping attacks in overlay networks: attack detection and attacker identification. ACM Trans. Inf. Syst. Secur. **11**(3), 15 (2008)

Rescuing LoRaWAN 1.0

Gildas Avoine[1,2,3] and Loïc Ferreira[2,4](✉)

[1] INSA Rennes, CNRS, Rennes, France
[2] IRISA UMR 6074, Rennes, France
[3] Institut Universitaire de France, Paris, France
[4] Orange Labs, Applied Cryptography Group, Caen, France
loic.ferreira@orange.com

Abstract. LoRaWAN is a worldwide deployed IoT security protocol. We provide an extensive analysis of the version 1.0, which is the currently deployed version, and we show that it suffers from several weaknesses. We introduce several attacks, including practical ones, that breach the network availability, data integrity, and data confidentiality, and target either an end-device or the backend system.

Based on the inner weaknesses of the protocol, these attacks do not lean on potential implementation or hardware bugs. Likewise they do not entail a physical access to the targeted equipment and are independent from the means used to physically protect secret parameters.

Finally we propose practical recommendations aiming at thwarting the attacks, while at the same time being compliant with the specification, and keeping the interoperability between patched and unmodified equipment.

1 Introduction

1.1 Context

With the arrival of the Internet of Things, several communication protocols have been proposed, with technical specifics suited to the intended use case. For instance, the Bluetooth wireless protocol [3] is designed for short distance communication. Technologies such as ZigBee [30] or Z-Wave [29] afford medium range distance communication, and aim at reducing the energy needed by the nodes to set up and maintain a mesh network.

As for long range distance communication (several kilometers), proposals have been made, such as LoRa. LoRa, developed by Semtech company, aims at setting up a Low-Power Wide-Area Network (LPWAN) based on a long-range, low-rate, wireless technology. It is somewhat similar to a cellular technology (2G/3G/4G mobile systems) but optimised for IoT/M2M. LoRa does not require a spectrum license because it uses free (although regulated) frequency bands (e.g., 863-870 MHz in Europe, 902-928 MHz in the USA, 779-787 MHz in China) [14]. A LoRa end-device, with an autonomous power-supply, is supposed to communicate through several kilometers in an urban area, and to have a lifespan up to eight or ten years.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 253–271, 2018.
https://doi.org/10.1007/978-3-662-58387-6_14

LoRaWAN is a protocol that aims at securing the Medium Access Control layer of a LoRa network. It is designed by the LoRa Alliance, which is an association that gathers more than 400 members (telecom operators, semiconductor manufacturers, digital security companies, hardware manufacturers, network suppliers, etc.).

Public and private LoRaWAN networks are deployed in more than 50 countries worldwide [23] by telecom operators (SK Telecom, FastNet, ZTE, KPN, Orange, Proximus, etc.), private providers (e.g., LORIOT.io [15]), and private initiatives (e.g., The Things Network [27]). Several nationwide networks are already deployed in Europe (France, Netherlands) [7], Asia (South Korea) [16], Africa (South Africa) [1], Oceania (New Zealand) [24], providing coverage to at least half of the population. Trials are launched in Japan [4], the USA (starting with a hundred cities) [10], China (the expected coverage extend to 100 million homes and 300 million people) [22], India (the first phase network aims at covering 400 million people across the country) [9].

The services provided by LoRa end-devices are numerous, from performing measurements (humidity, temperature, water leak, etc.), up to achieving more sensitive purposes such as triggering an alarm/help message, detecting an intrusion, or allowing to remotely switch on and off another equipment. The data sent by the end-device may also have to remain confidential (e.g., geolocation of valuable assets sent by a tracker).

In this paper we focus on the version 1.0.2 of the LoRaWAN specification released in 2016, which is the version currently deployed worldwide, and whose official name is now 1.0.

1.2 Protocol Overview

A LoRaWAN network corresponds to a star-of-stars topology: a set of end-devices communicates with several gateways, which relay the data to a Network Server (NS) in the backend. In turn, the NS delivers the data to one or more Application Servers (AS), which own the corresponding end-devices, optionally through intermediary servers such as an MQTT server (see Fig. 1). The security mechanisms are based on a symmetric key `AppKey` (the root key) shared between an end-device and the NS. From this key, distinct per end-device, two session keys are computed: the application session key `AppSKey` guarantees the data confidentiality between the end-device and the AS; the network session key `NwkSKey` guarantees the data integrity between the end-device and the NS (it is worth noting that data integrity is not provided end-to-end between the end-device and the AS[1]). When a frame is exchanged exclusively between an end-device and the NS, both data confidentiality and data integrity are provided by the network session key `NwkSKey`. An application payload is always encrypted. Moreover, if no payload is carried, the frame is only authenticated. Encryption is done with AES [19] in CTR mode [5,20], and data integrity is provided with AES in CMAC mode [21,25]. An end-device may establish an "activation" (namely a session) with the

[1] As acknowledged by the specification ([26], Sect. 6.1.4).

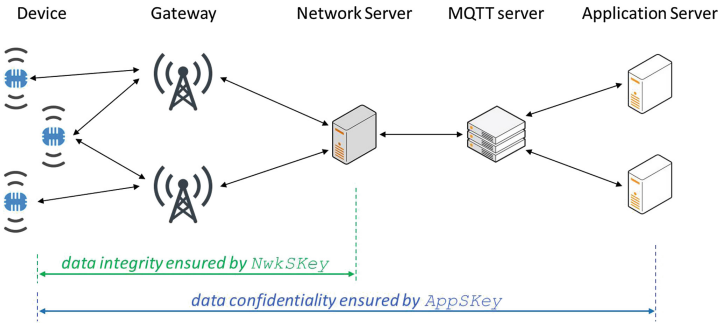


Fig. 1. LoRaWAN network (simplified view)

NS through two ways. The pre-personalization (Activation By Personalization, ABP) consists in setting two session keys (and other parameters but not the `AppKey` root key) into the end-device before its deployment. An ABP end-device is then able to communicate with the NS (and its AS) but not to renew the "session" keys. The other possibility (Over The Air Activation, OTAA) consists in provisioning the end-device with an `AppKey` root key and other parameters, allowing for key exchanges with the NS through the radio interface once it is deployed. In this paper we focus on OTAA end-devices.

1.3 Paper Outline

The LoRaWAN protocol is detailed in Sect. 2. Theoretical and practical attacks against LoRaWAN are described in Sect. 3. Section 4 describe recommendations that thwart the attacks. Section 5 summarises previous comments and analysis on the protocol. Section 6 deals with the responsible disclosure. We finally conclude in Sect. 7.

2 The LoRaWAN Protocol

The technical features described in this section are based on [26].

2.1 Key Exchange

The key exchange done over the air is triggered when the end-device sends a Join Request message. The NS then responds with a Join Accept message. The (unencrypted) Join Request message includes two static identifiers (the end-device's `DevEUI` and the AS' `AppEUI`), and a pseudo-random value `DevNonce` generated by the end-device. The message is protected with a 4-byte CMAC authentication tag (called `MIC`) computed with the 128-bit (static) root key `AppKey`. The Join Accept response from the NS contains the (static) identifier of the latter (`NetID`), a pseudo-random value generated by the NS (`AppNonce`), a value used

as the end-device short address (DevAddr), and several (optional) radio parameters. The Join Accept message is protected with a CMAC authentication tag, and encrypted with AES (both operations made with the root key AppKey).[2] Two 128-bit session keys are then computed:

$$\texttt{NwkSKey} = \mathsf{AES}(\texttt{AppKey}, \texttt{0x01}\|\texttt{data})$$
$$\texttt{AppSKey} = \mathsf{AES}(\texttt{AppKey}, \texttt{0x02}\|\texttt{data})$$

with $\texttt{data} = \texttt{AppNonce}\ (3)\|\texttt{NetID}\ (3)\|\texttt{DevNonce}\ (2)\|\texttt{0x00}\cdots\texttt{00}\ (7)$.

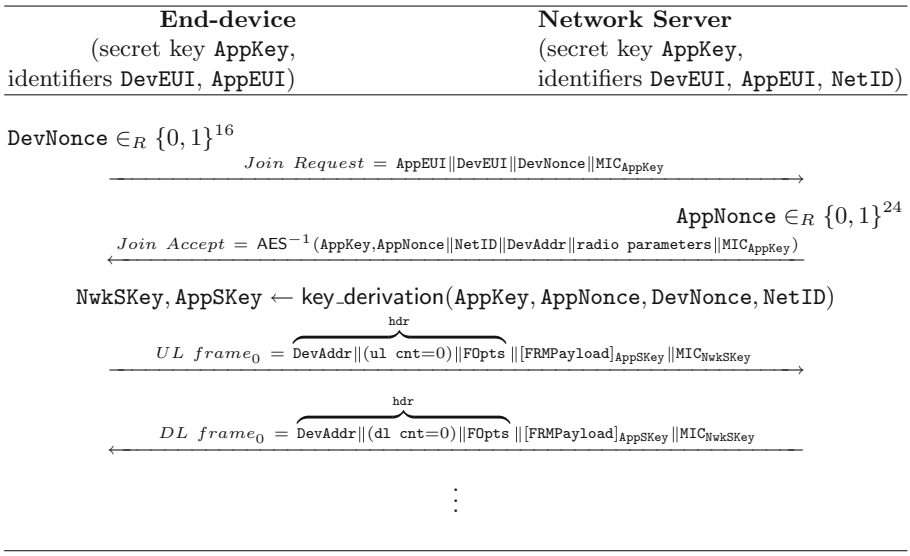


Fig. 2. LoRaWAN activation (simplified scheme).

Thus the session keys depend mostly on a secret and static value (the root key AppKey), and two pseudo-random values, respectively 2-byte and 3-byte long. Once the Join Request and Join Accept messages are exchanged, the end-device, the NS, and the AS are able to communicate. After the NS computes the session keys, it transmits the application session key AppSKey to the AS. It is worth noting that the AS is not involved in the key agreement, which is handled by the NS. The NS must keep the previous session keys, and the corresponding security parameters, until it receives a (valid) frame protected by the new security parameters. The security mechanisms between NS and AS are out of the LoRaWAN scope. Figure 2 depicts an activation.

[2] More precisely the AES decryption function is used to protect the Join Accept message, because the end-device implements the encryption function only.

2.2 Data Encryption and Authentication

The frame payload FRMPayload is encrypted in CTR mode. From block counters

$$A_i = \texttt{0x01}\ (1)\|\texttt{0x00}\cdots\texttt{00}\ (4)\|\texttt{dir}\ (1)\|\texttt{DevAddr}\ (4)\|\texttt{cnt}\ (4)\|\texttt{0x00}\ (1)\|i\ (1)$$

a secret keystream $S_i = \mathsf{AES}(K, A_i)$ is produced with $K \in \{\texttt{AppSKey}, \texttt{NwkSKey}\}$, and used to mask the payload[3]: $[\texttt{FRMPayload}] = (S_0\| \cdots \|S_{n-1}) \oplus \texttt{FRMPayload}$.

dir specifies the direction (uplink = 0x00, downlink = 0x01). cnt is the frame counter (of 16 or 32 bits), initialised to 0 when the session starts, and monotonically increased when a (valid) frame is sent or received. Two different counters are used depending on the frame's direction. DevAddr is the end-device address (within a given LoRa network) chosen by the NS and sent in the Join Accept message, and it remains constant during the entire session. To compute DevAddr, seven bits are chosen from the NS' unique identifier NetID: $\mathsf{msb}_7(\texttt{DevAddr}) = \mathsf{lsb}_7(\texttt{NetID})$, and 25 bits are "*arbitrarily*" assigned by the NS. The i value numbers the AES blocks within the payload to encrypt.

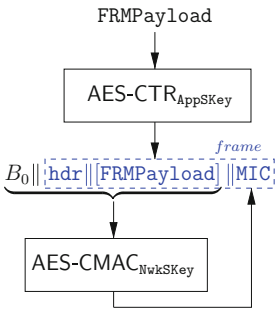


Fig. 3. Generation of an application frame.

A 4-byte authentication tag is computed with CMAC and the network session key NwkSKey on the whole frame (header hdr of size $hlen \in \{8, \ldots, 24\}$ and encrypted payload [FRMPayload] of size $plen$) and a 16-byte prefix block

$$B_0 = \texttt{0x49}\ (1)\|\texttt{0x00}\cdots\texttt{00}\ (4)\|\texttt{dir}\ (1)\|\texttt{DevAddr}\ (4)\|\texttt{cnt}\ (4)\|\texttt{0x00}\ (1)\|(hlen + plen)\ (1)$$

The frame eventually sent is hdr $(hlen)\|[\texttt{FRMPayload}]\ (plen)\|\texttt{MIC}\ (4)$.

The frame header hdr includes, among other fields, DevAddr, the frame counter cnt on 2 bytes, and an (optional) field FOpts which may contain commands exclusively exchanged between the end-device and the NS (see Fig. 3).

[3] The session key K = AppSKey is used when application messages are exchanged between the end-device and the AS, and K = NwkSKey is used when command-only messages are exchanged between the end-device and the NS.

3 Attacks Against LoRaWAN

Hereinafter we present our findings regarding the LoRaWAN protocol version 1.0, the currently deployed version. Table 1 summarises the attacks we have found.

Our attacker, standing between a LoRaWAN end-device and the NS, needs only to act on the air interface: she needs to eavesdrop on data exchanged between the end-device and the server, and to send data to any equipment.

Table 1. Attacks against LoRaWAN 1.0 (n_{ja}: number of Join Accept messages usable by the attacker. n_{jr}: number of Join Request messages usable by the attacker. m: number of new session keys sets stored by the NS. ED: end-device)

Attack	Complexity (# Join message)	Probability of success	Impact
(A1) Replay or decrypt (ED, Sect. 3.1)	$2^{16}/n_{ja}$	$\simeq 1$	Downlink frame replay, uplink frame decryption
(A2) Replay or decrypt (NS, Sect. 3.1)	n_{jr}	$\simeq n_{jr}/2^{24}$	Uplink frame replay, downlink frame decryption
(A3) Desynchronization (ED, Sect. 3.2)	1	1	End-device desynchronization
(A4) Desynchronization (NS, Sect. 3.2)	m	1	End-device desynchronization

3.1 Replay or Decrypt

Targeting an End-Device (Attack A1)

Goal. The purpose of this attack is to compel the end-device to reuse previous session keys and other security parameters. When this happens, frames picked from a previous session become cryptographically valid anew, hence can be replayed. Moreover the same secret keystream is then used to protect the frames exchanged during the new session. This allows an adversary to decrypt frames.[4]

[4] Note that since the end-device ends up reusing previous session keys (which are no longer shared with the NS), this attack is also a kind of "desynchronization" attack. However, contrary to the desynchronization attacks described in Sect. 3.2, this "replay or decrypt" attack has more devastating consequences (and a higher complexity) than "only" desynchronizing the end-device and the NS.

Core. The encryption keystream $S_i = \mathsf{AES}(K, A_i)$ used to protect a frame payload is produced from a session key $K \in \{\texttt{AppSKey}, \texttt{NwkSKey}\}$ and A_i block counters. Within a given session the blocks

$$A_i = \texttt{0x01}\ (1) \| \texttt{0x00} \cdots \texttt{00}\ (4) \| \texttt{dir}\ (1) \| \texttt{DevAddr}\ (4) \| \texttt{cnt}\ (4) \| \texttt{0x00}\ (1) \| i\ (1)$$

(as well as the prefix block B_0) depend mostly on the frame counter `cnt` (set to 0 when the session starts and monotonically increased frame after frame), and on the `DevAddr` parameter (static during the whole session). The other parameters are the direction `dir` unchanged for a given direction, and the i block index which evolves the same way for each frame. Hence the way the keystream S_i changes depends only on the `DevAddr` parameter and the session key (usually `AppSKey`). For a given end-device, which connects to the same NS (hence uses the same static `NetID` parameter), the session keys depend mainly on a secret and static value (`AppKey`) and two pseudo-random values (`DevNonce`, `AppNonce`). Therefore, if one succeeds in compelling the end-device to reuse the same `DevAddr`, `DevNonce` and `AppNonce` parameters, this leads not only to the reuse of previous session keys `AppSKey`, and `NwkSKey`, but also to the reuse of previous keystream S_i and prefix block B_0.

Attack. The purpose is to make the end-device use twice the same `DevNonce`, `AppNonce`, and `DevAddr` values. The 2-byte `DevNonce` and 3-byte `AppNonce` parameters are pseudo-random. Let us assume that an attacker is able to impose the `AppNonce` value that the end-device uses to compute the session keys. The probability that the session keys repeat depends then only on the `DevNonce` parameter. Firstly note that a collision due to the birthday paradox happens with high probability ($p = \frac{1}{2}$) after roughly $\sqrt{2\ln(2) \times 2^{16}} \simeq 301$ activations only. However the attacker can speed up the whole process: the attacker eavesdrops on a given session, and compels the end-device to generate multiple `DevNonce` values until the expected value is produced once again. In such a case only one value among 2^{16} is useful to the attacker. Hence, the end-device must generate on average 2^{16} `DevNonce` values. If the attacker eavesdrops on n_{ja} different Join Accept messages (each one corresponding to a different `DevNonce` value) previously received by the targeted end-device, then the probability for the attacker to succeed is $p = n_{ja}/2^{16}$. If the attacker repeats this experiment k times, the probability to be successful at least once among the k experiments is $1-(1-p)^k \simeq kp$ if p is close to 0. In order for this probability to be close to 1, the number of experiments the attacker has to perform is $k \simeq 2^{16}/n_{ja}$. This means that the end-device has to send $2^{16}/n_{ja}$ Join Request messages before one carries a `DevNonce` value that matches with one of the Join Accept messages.

The shortest receiving window of a Join Accept message is 5 s [14]. If the attacker uses $n_{ja} = 16$ Join Accept messages, the attack is achieved after roughly $2^{16}/16 \times 5\,\text{s} = 5.7\,\text{h}$ (assuming that the time needed to process the Join messages is negligible compared to the communication duration).[5]

[5] See Appendix A.

Once this first phase of the attack is achieved, the attacker ends with two different sessions protected with the same security parameters, denoted respectively s_{old} and s_{new}.

Technique 1 Used to Achieve the Attack: Replay of a Join Accept Message. In order to compel the end-device to use a given `AppNonce` value, the attacker can replay a previous Join Accept message sent to the targeted end-device. Then the end-device will reuse (once again) the parameters included in the message. Indeed the data carried in a Join Accept message correspond to

$$\texttt{AppNonce}\ (3)\|\texttt{NetID}\ (3)\|\texttt{DevAddr}\ (4)\|\texttt{radio parameters}\ (2\ldots18)\|\texttt{MIC}\ (4)$$

where `MIC` is an authentication tag computed on the preceding fields with the (static) root key `AppKey`. These parameters are protected with AES and `AppKey`. The cornerstone of this attack is that all the parameters are chosen by the NS, in particular `AppNonce` and `DevAddr`. `NetID` is the NS' (static) identifier, and the `radio parameters` are also defined by the NS. The only secret parameter involved in the message calculation is static (`AppKey`). Hence, the end-device is not able to verify if the received Join Accept message corresponds to the Join Request it sent. Replaying a Join Accept message allows the attacker to compel the end-device to (re)use both `AppNonce` and `DevAddr` parameters.

Technique 2 Used to Achieve the Attack: Harvest of Join Messages. The ability of the attacker to make the end-device generate multiple `DevNonce` values is related to the behaviour of the end-device when it sends a Join Request message but does not receive a Join Accept response or receives an invalid message. The specification states that the NS shall ignore Join Request messages containing previously used `DevNonce` values in order to thwart a replay attack ([26], Sect. 6.2.4). Hence, the end-device has to generate a new pseudo-random `DevNonce` value each time it computes a Join Request message, even when a previous Join Request message did not receive a response. Otherwise the end-device may fear the subsequent Join Request messages to be dropped by the NS. This allows the attacker to collect multiple new and valid Join Request messages. It is enough for the attacker to send "false" (i.e., invalid) Join Accept messages in response to the end-device's messages. Moreover, if the attacker forbids the NS from receiving the Join Request messages sent by the end-device, he gets "fresh" messages (i.e., unknown to the NS) for free. In order to make the end-device start producing the Join Request messages, the attacker may wait or force (once only) the end-device to start a new session (e.g., the attacker may turn the end-device off and on: once the power supply is re-established, the end-device likely starts a new activation).[6]

[6] Being able to influence on the power supply does not necessarily mean to physically intrude on the end-device. The attacker could turn off or interrupt a remote electric generator the end-device is connected to, or the link between the generator and the end-device (if the end-device is powered by an external source), or use other means (e.g., electromagnetic impulse targeting the end-device and leading to a power outage).

Note that every time the NS receives a Join Request message, it sends a new Join Accept message. Therefore, this procedure is also a way to collect multiples Join Accept messages.

Impact: Frame Replay. Frames drawn from the previous session (s_{old}) can be replayed to the end-device throughout the new session (s_{new}).[7] These frames are valid since they are protected with a cryptographically correct keystream and authentication tag.

Impact: Frame Decryption. The frame payload is encrypted in CTR mode. Once the attack is achieved, the end-device uses twice the same keystream in order to protect different frames. The frame of counter t sent during session s_{old} contains an encrypted payload $c_t^{s_{old}} = m \oplus k_t^{s_{old}}$, where m is the clear data and $k_t^{s_{old}}$ the keystream. The frame of same counter t sent during session s_{new} contains an encrypted payload $c_t^{s_{new}} = m' \oplus k_t^{s_{new}}$. Since $k_t^{s_{old}} = k_t^{s_{new}}$, we have that $c_t^{s_{old}} \oplus c_t^{s_{new}} = (m \oplus k_t^{s_{old}}) \oplus (m' \oplus k_t^{s_{new}}) = m \oplus m'$. Therefore m and m' may (partially or completely) be retrieved, in an obvious manner if either message is known, or through analysis of $m \oplus m'$ [17].

Targeting the NS (Attack A2)

Goal. The same kind of attack can be performed against the NS, aiming at compelling the server to use the same security parameters throughout two different sessions. The goal is then to compel the NS to use twice the same `DevNonce`, `AppNonce`, and `DevAddr` values.

Attack. An attacker who replays a Join Request message sets the `DevNonce` value before knowing the `DevAddr` and `AppNonce` values generated by the NS. These values must correspond to the `DevNonce` value chosen by the attacker. Hence only one such pair among all possible values is of interest to the attacker.

According to the specification, the NS must keep track of "*a certain number*" of received `DevNonce` values in order to prevent replay attacks, without clarifying if this means all values or a few of them. We may reasonably assume that the NS keeps track of a few values (say n). Thus the attacker cannot choose any Join Request she wants to replay. The corresponding `DevNonce` value must not belong to the list of n stored values. If the value the attacker wants to replay still belongs to the server's list (let i be its index, with 0 and $n-1$ the index of the oldest and of the latest received values), she has to wait for $i+1$ additional (legitimate) key exchanges before the NS "forgets" that value. The duration of such an "opportunist" attack depends on the frequency of the key exchanges.

[7] We use the term "session" for the sake of simplicity, but it does not depict precisely what are the actual exchanges since the end-device, at this point, has no "partner": neither the NS nor the AS is able to communicate with the end-device, and the attacker is unable to forge new valid frames.

According to the specification, `AppNonce` is a 3-byte pseudo-random value, and the 32-bit `DevAddr` parameter is made of 7 bits from `NetID`, and 25 bits "*arbitrarily*" chosen by the NS ([26], Sect. 6.1.1). If `DevAddr` is pseudo-random then the probability of success is $2^{-(24+25)} = 2^{-49}$. But "*arbitrarily*" does not mean "pseudo-random" and experiments we have performed show that the `DevAddr` parameter may remain unchanged for a given end-device throughout different sessions.[8] In such a case the probability of success increases to 2^{-24}, and the overall probability of success is 2^{-24} every $n + 1$ sessions. Alternatively, the attacker can eavesdrop on n_{jr} different Join Request messages (that the NS has "forgotten"), and send them to the server.[9] The probability that at least one message triggers the same `AppNonce` value as during a previous session is $1-(1-2^{-24})^{n_{jr}} \simeq n_{jr} \times 2^{-24}$. For instance, if the attacker uses $n_{jr} = 2048$ Join Request messages, her probability to succeed raises to $\frac{1}{8192}$.

This attacker knows if the `AppNonce` value repeats through the direct comparison of the Join Accept messages, even if these messages are encrypted. Indeed, all the parameters of such a message are likely static but the `AppNonce` parameter.

Impact. Once the attacker succeeds in compelling the NS to compute once again the same security parameters, she eventually gets two different sessions (s_{old} and s_{new}) protected with the same security parameters. The attacker is then able to replay uplink frames and attempt decryption of downlink frames. Note that the attacker is then able to send (i.e., to replay) a valid frame that indicates the NS to switch from the current security context to the new one (hence the NS drops the current session keys and uses the new ones).

3.2 Desynchronization

Targeting an End-Device (Attack A3)

Goal. This attack aims at "disconnecting" the end-device from the network. That is the end-device performs a successful key exchange which ends with the end-device not sharing the new session keys with the NS (the end-device has no "partner"). Therefore the frames sent by the end-device are ignored by the NS, and conversely.

Core. The session keys are computed, by a given end-device and the NS, with two static parameters (the NS' unique identifier `NetID`, and the end-device's root key `AppKey`), and two variable parameters (the pseudo-random values `AppNonce` computed by the NS, and `DevNonce` by the end-device). As soon as the end-device receives a (valid) Join Accept message it can derive the session keys and

[8] Thus some NS implementation derives the `DevAddr` parameter from the unique end-device's identifier `DevEUI`. Also the `DevAddr` value may be chosen once and for all at the time of the end-device provisioning.

[9] The messages may come from different end-devices, hence, may have to be sent to one or several NS servers.

start transmitting protected frames. In the key derivation, if the end-device uses values different from those actually sent by the NS (say $(\texttt{DevNonce}, \texttt{AppNonce}) = (x, \tilde{y})$ on the one hand, and $(\texttt{DevNonce}, \texttt{AppNonce}) = (x, y)$, on the other hand, $y \neq \tilde{y}$), it eventually computes different session keys than those computed by the server. This does not forbid the end-device to send protected frames though. However those frames will be dropped by the NS since they are invalid from the server perspective. Conversely, the frames sent by the NS will be discarded by the end-device. Thus the end-device, unable to communicate with the NS, is "disconnected" from the network.

Attack. In order to perform such a desynchronization attack, an attacker can first passively eavesdrop on a Join Accept message sent by the NS in response to the end-device's Join Request message. When the end-device starts a new session and sends another Join Request message, the attacker replies before the NS and replays the eavesdropped Join Accept message. This replayed message likely contains an $\texttt{AppNonce} = \tilde{y}$ value different from the fresh one sent by the NS ($\texttt{AppNonce} = y$). Hence, the end-device and the NS compute different session keys and security parameters.

Means Used to Achieve the Attack. The attacker is able to replay a previous Join Accept message thanks to the peculiarities of the LoRaWAN protocol: indeed the end-device has no means to verify neither if the message is a replay, nor if it is an actual response to the Join Request message it just sent. Moreover the attacker can use the procedure described in Sect. 3.1 to collect several Join Accept messages and use these "desynchronization ammunition" anytime later. The Join Accept message used by the attacker must be intended to the targeted end-device. Indeed such a message is protected with the root key of the end-device it is sent to.

Impact. Such a desynchronization attack may be harmful because it can lastingly disturb the operating of a LoRaWAN network. So then the usual behaviour of a sensor may be to regularly send some measurements without expecting a response unless the server detects an anomaly in the collected data. If the end-device sends its measurement at a low rate, days or even weeks may elapse before something abnormal is noticed, even if the end-device is supposed to react if it does not receive a downlink frame after a fixed number of sent frames.

Targeting the NS (Attack A4)

Goal. The same kind of desynchronization attack can be done against the NS, aiming at disconnecting a given end-device from the network. In that case, the NS completes the key exchange without being "partnered" with the intended end-device (i.e., identified by the `DevEUI` parameter within the Join Request message). Therefore the frames the NS (or the AS) may send are ignored by the end-device, and conversely.

Attack. Upon reception of a (valid) Join Request message, the NS generates a new `AppNonce` value and computes new session keys. If an attacker succeeds in replaying to the NS a valid Join Request message, the corresponding end-device will no longer share the same session keys with the NS. The attacker can do the following. She waits for the end-device to start a new activation. New session keys ($\texttt{seskeys}_{i+1}$) are then computed. The end-device stores $\texttt{seskeys}_{i+1}$ only while the NS stores both $\texttt{seskeys}_i$ and $\texttt{seskeys}_{i+1}$ (respectively the current and the new session keys). Before the end-device sends a frame, the attacker immediately sends to the NS a Join Request message she previously eavesdropped on (and not received, hence new to the server). The server computes new session keys $\texttt{seskeys}_{i+2}$ which replace the unconfirmed keys $\texttt{seskeys}_{i+1}$. Then the NS stores $\texttt{seskeys}_i$ and $\texttt{seskeys}_{i+2}$ while the end-device stores $\texttt{seskeys}_{i+1}$. Hence the end-device and the NS do not share the same session keys. More generally, if the NS keeps the latest valid session keys and m new sets of keys, the attacker must send successively m new Join Request messages in order to desynchronize the NS and the end-device.

Means Used to Achieve the Attack. In order to get a new Join Request message the attacker can use the technique described in Sect. 3.1 aiming at compelling the end-device to generate multiple Join Request messages. The attacker can gather several such messages and use these anytime later as "desynchronization ammunition".

Impact. The consequences of this attack against the NS are the same as the one against the end-device: the targeted end-device is disconnected from the network. Unaware that the NS does not share the same security parameters, it may keep sending uplink frames for quite a long time while the NS is unable to process them. Conversely, the frames the NS may send cannot be understood by the end-device.

4 Recommendations

In this section we aim at providing recommendations that thwart the attacks described in Sect. 3. This may lead to major changes in the protocol specification and break the interoperability between patched and non-modified equipment. Hence, as an additional constraint, we aim at proposing improvements that could solve the issues as best as possible while retaining at the same time the compliance with unchanged version of end-devices or servers, in particular equipment that is already deployed and may not be easily patched.

The methods to be implemented in order to thwart the attacks against LoRaWAN must be chosen with caution. Indeed, the reduced LoRaWAN parameters size limits the efficiency of some countermeasures one may think of by paving the way to new attacks.[10] In order to preclude all the attacks, we recommend to implement all the following changes.

[10] For instance turning the `DevNonce` parameter into a counter is not a suitable solution (see Appendix B).

Generate `AppNonce` *Values with no Repetition.* This countermeasure aims at thwarting attack A2. A counter may be used to produce the `AppNonce` values. The counter must not overlap, and one different counter must be used for each end-device in order not to artificially lower the number of activations per end-device.[11]

Detect a Replay of `AppNonce` *Values.* This countermeasure aims at thwarting attack A1. It may be implemented using computationally and memory efficient techniques such as Bloom filters [2,6]. However the `AppNonce` parameter being turned into a counter, it is enough for the end-device to store the last received `AppNonce` value in order to detect a replay.

Verify that the Received Join Accept Message Corresponds to the Sent Join Request Message. This countermeasure aims at checking that the Join Request and Join Accept messages are bound in order to thwart attack A3. We recommend to compute the `DevAddr` parameter in the following way. Let `NwkAddr` be the least 25 significant bits. `NwkAddr` is computed as $\texttt{NwkAddr} = \mathsf{H}(\texttt{DevNonce}, \texttt{AppNonce}, \texttt{DevEUI})$ where H is a collision-resistant function.

Verify that the Session Keys are Shared. This countermeasure aims at thwarting attack A4. We suggest to implement it the following way. Straight after the key exchange is done, the NS must send a so-called *DevStatusReq* command and verify (authentication tag) the *DevStatusAns* response from the end-device, or verify, if it comes earlier, the first frame sent by the end-device. The lack of response must be read into this as an issue (device or NS under attack).

In addition the NS must keep all sets of session keys from the last valid one up to the latest computed one. When the NS receives an uplink frame (carrying a *DevStatusAns* response, or another uplink frame), it checks the authentication tag with all keys, starting from the latest. If the keys that match with the authentication tag belong to one of the (currently) unapproved sets, then the NS keeps this set of session keys only and drops all the others. This set becomes then the last valid one.

5 Related Work

Few analyses on LoRaWAN have been done and publicly released. Most of the public reviews deal with technical consideration such as the network management (secret keys storage, etc.) and generic attacks (e.g., hardware attacks, web attacks) unrelated to the LoRaWAN protocol. Some attacks, which exploit specific features of the protocol, are mentioned but without excess of details.

Regarding the presentation [12], no paper nor slides were made publicly available after the conference (to the best of our knowledge), however we got a summary of the talk. Yet we cannot claim to be aware of all the specifics provided during the talk.

[11] This would also make easier an attack aiming at exhausting the `AppNonce` counter (see Appendix B).

Desynchronization Against an End-Device. According to Lifchitz [12], L'Héréec and Joulain [11], and Miller [18] a way to attack the end-device is to replay to the NS a previous Join Request message, leading to the end-device "disconnection" from the network. Conversely Zulian indicates that it is possible to replay a previous Join Accept message to an end-device ending with different session keys used by the NS and the end-device [31]. However Zulian does not exploit all the possibilities provided by such a replay. In particular, he does not envisage more devastating attacks such as our "replay or decrypt" attack.

Frame Replay and Frame Decryption. Regarding the pseudo-random `AppNonce` parameter, Lifchitz notes that it may repeat due to the birthday paradox [12]. Hence, under the strong assumption that the `DevNonce` value is "*forced*" (device controlled by an attacker), a keystream reuse is possible with high probability after $\sqrt{2\ln(2) \times 11 \times 2^{24}} \simeq 16{,}000$ activations, or 22 h if a key exchange is done in 5 s. In fact, such a statement is wrong or, at least, hazy: if both `DevNonce` and `AppNonce` values repeat, this leads to a *session keys* reuse. In order to get a *keystream* reuse, it is necessary for the `DevAddr` parameter to repeat as well. Moreover this means a continuous series of key exchanges without any intermediary application frame. Hence the sake of such an attack may be questioned.

Finally this attack is unlikely successful against a NS implementing version 1.0.2 (the current 1.0 version). Indeed, according to the specification, the NS must receive a valid uplink frame protected by the new security parameters before dropping the current ones and using the new ones. The attack leads to the computation of the same session keys two different times. Yet, with high probability, these keys are fresh (i.e., never used previously by the NS with a legitimate end-device) because the attacker has no control on the `AppNonce` parameter. This means that the attacker has to forge a valid uplink frame if she wants to compel the NS to use these keys. That is the attacker must forge a valid 32-bit authentication tag (without the corresponding key). That being said, we are not aware of the LoRaWAN version analysed in that talk (1.0.1 or 1.0.2). Moreover Lifchitz does not consider the attacks doable against an end-device (without any physical intrusion on it).

Yang notes that it is possible to replay previous frames and to decrypt frames if some security parameters are reused (namely frame counter, keytsream) [28]. According to the author, this can be done if the frame counter is reset or wraps around. However the way to achieve the latter is not explained (in particular regarding an OTAA end-device). Moreover Yang's attacker targets the NS. It is unclear why the server would accept frame replays (it seems that Yang confuses the *gateway* and the NS). Similarly, the reuse of the keystream (allowing to decrypt frames) is due to a reuse of the same frame counter (with unchanged session keys). Yet, how to get the latter is not explained.

Data Integrity. Yang indicates that the lack of data integrity between the NS and the AS allows an attacker to modify the plaintext by changing the encrypted payload (due to the encryption in counter mode) [28].

6 Responsible Disclosure

We have informed the LoRa Alliance of the vulnerabilities and the subsequent attacks against LoRaWAN 1.0. Prior to our communication, the LoRa Alliance's technical committee decided to start the development of a new version (namely 1.1). As a result of our disclosure, some countermeasures we propose have been included in the version 1.1 (turning the `AppNonce` parameter into a counter), while some features similar to other countermeasures were already included in the specification (binding the Join Request and the Join Accept messages, doing a key confirmation between the end-device and the NS).

7 Conclusion

The extensive analysis we perform of the security protocol LoRaWAN 1.0 shows that it suffers from several weaknesses. We describe precisely how these flaws can be exploited to carry out attacks, including practical ones. These attacks lead to a breach in data integrity, data confidentiality, and in the network availability.

The first type of attacks ends up with the end-device desynchronization from the network (that is the end-device is "disconnected"). The second kind allows an attacker to replay and to decrypt frames, therefore deceiving the NS (and the AS) or the end-device (which may be an actuator). The aforementioned attacks, due to the protocol flaws, do not lean on potential implementation or hardware bugs, and are likely to be successful against any equipment implementing LoRaWAN 1.0.

We present new attacks and, contrary to previous works (to the best of our knowledge), the attacks we describe target both types of equipment (end-device or NS). Moreover our attacker needs only to act on the air interface (to eavesdrop and send data), but she does not need to get a physical access to any equipment (in particular the end-device). In addition, the success of the attacks is independent from the means used to protect the secret values (e.g., using a tamper resistant module such as a Secure Element).

In addition we provide practical recommendations allowing to thwart the attacks we have found, while at the same time being compliant with the specification, and keeping the interoperability between patched and unmodified equipment. According to us, the recommended countermeasures can be implemented in a straightforward manner.

Acknowledgment. The authors thank Sébastien Canard for valuable comments and suggestions, and the anonymous reviewers for helpful comments. This article is based upon work from COST Action IC1403 CRYPTACUS, supported by COST (European Cooperation in Science and Technology).

A Duty Cycle

The ducty cyle is a mechanism used to regulate the occupation rate of the radio channel by the end-device. Enforcing the duty cycle implies that an end-device cannot repeatedly send a lot of messages. Hence one could claim that the

duration of the attack is greater than the figure we provide. However, the duty cycle is a regulation mechanism, not a security one (even if it could cleverly be used as such). And not all countries compel to use such a mechanism. Also, an end-device may well be certified (by the LoRa Alliance [13]) and yet not apply the duty cycle. Indeed a LoRa Alliance certification document explicitly states that "*the LoRa Certification testing will not do any duty cycle testing*" [8].

B Exhaustion Attack

Generating a parameter (`DevNonce`, `AppNonce`) with no repetition, and detecting replays are some countermeasures one may think of. Yet, in LoRaWAN, size does matter. Applying one of these methods *while keeping at the same time* the original parameter size (for compliance reasons) may lead to an attack aiming at exhausting all possible `DevNonce` or `AppNonce` values, hence forbidding the NS or the end-device to start a new activation. Therefore this exhaustion attack, targeting the end-device or the NS it connects to, may lead to an irrevocable disconnection of the end-device.

B.1 Against the `DevNonce` Parameter

Core. If the end-device generates `DevNonce` values with no repetition or if the NS keeps track of all `DevNonce` values it receives, it is possible to disconnect the end-device once and for all.

Attack. Every time the end-device sends a Join Request message, the attacker replies with a "false" Join Accept message. Hence the end-device generates a new message once again. If unique `DevNonce` values are generated, all values will be eventually used. If the NS keeps track of all `DevNonce` values, the NS will refuse further Join Request messages once all possible `DevNonce` values have been received, be these values pseudo-random or not.

Numerical Example. Let us assume that a key exchange is done in 5 s. If the `DevNonce` values never repeat, the attack targeting the end-device is achieved in $2^{16} \times 5\,\text{s} = 91\,\text{h}$.

Let us consider the case when the NS keeps track of all `DevNonce` values. If the values are pseudo-random, the proportion effectively generated by the end-device, hence received by the NS after ℓ key exchanges, is $p = 1 - \exp(-\frac{\ell}{2^{16}})$. In order this proportion to be $p = 99\%$, the number of key exchanges must be at least $\ell = -2^{16} \times \ln(1-p)$. This corresponds to $\ell \simeq 301{,}804$ activations and more than 17 days to exhaust almost all `DevNonce` values. Remind that such an end-device is supposed to have an autonomous lifespan of up to ten years.

B.2 Against the `AppNonce` Parameter

Core. If the NS generates the `AppNonce` parameter so that it never repeats, or if the end-device keeps track of all `AppNonce` values it receives, then it is possible to disconnect the end-device once and for all.

Attack. Let us consider the first case. The purpose is to compel the NS to use all possible `AppNonce` values. The NS generates a Join Accept message (hence a new `AppNonce` value) only if it receives a valid Join Request message. Therefore the NS must accept as many Join Request messages as possible `AppNonce` values. Since $|\texttt{DevNonce}| < |\texttt{AppNonce}|$, this is possible only if the NS does not keep track of all `DevNonce` values it receives (which is likely its behaviour). Then the attacker can use a circular list of Join Request messages. Such messages can be collected using the technique described in Sect. 3.1, and then used in a similar way as the one described in Sect. 3.1.

Note that if the NS uses the same pool of `AppNonce` values for all the end-devices, this leads to the definitive disconnection of all these end-devices. In such a case the attack may be distributed among several "false" end-devices (controlled by the attacker; no duty cycle enforced).

Let us consider the second case. The purpose of this attack is to make the end-device keep track, hence receive, all possible `AppNonce` values. This means that the NS has to accept as many Join Request messages as possible `AppNonce` values. Therefore the NS must not keep track of all the `DevNonce` values it receives (since $|\texttt{DevNonce}| < |\texttt{AppNonce}|$). Yet this is not sufficient. Indeed the end-device accepts as many Join Accept messages (hence `AppNonce` values) as Join Request messages it sends. Therefore if the end-device generates `DevNonce` values with no repetition, it limits the number of received `AppNonce` values. Therefore this attack is possible if the NS does not keep track of all `DevNonce` values, and if the end-device does not generate unique `DevNonce` values (which is likely their basic behaviour).

Moreover the implementation of this second case implies to be able to compel the end-device to send multiples Join Request messages *while receiving* the corresponding Join Accept responses. We have not identified such means but to be able to influence on the end-device power supply. Yet, if the end-device is switched off, it may lose memory of the stored `AppNonce` values, which is orthogonal to the goal of this attack.

Numerical Example. If the `AppNonce` values do not repeat, they are all produced after $2^{24} \times 5\,\text{s} = 2.66$ years (using one end-device).

If the same pool of `AppNonce` values is used by the NS for all end-devices, the attack may be distributed among several end-devices controlled by the attacker. If 300 such end-devices are used in parallel, the attack is achieved in 3 days approximately.

If the end-device keeps track of all `AppNonce` values, and if the values are pseudo-random, $p = 99\%$ values are received by the end-device after $\ell = -2^{24} \times \ln(1-p) \simeq 77.26 \times 10^6$ activations. This means more than 12 years to exhaust almost all `AppNonce` values.

References

1. BiztechAfrica: FastNet announces Africa's first dedicated M2M network and IoT developer academy, October 2015. http://www.biztechafrica.com/article/fastnet-announces-africas-first-dedicated-m2m-netw/10718/
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). http://doi.acm.org/10.1145/362686.362692
3. Blutetooth SIG: Bluetooth specification. https://www.bluetooth.com/specifications/adopted-specifications
4. Briodagh, K.: Japan Opens New LoRaWAN Network for IoT Testing, July 2016. http://www.iotevolutionworld.com/iot/articles/423324-japan-opens-new-lorawan-network-iot-testing.htm
5. Diffie, W., Hellman, M.E.: Privacy and authentication: an introduction to cryptography. Proc. IEEE **67**(3), 397–427 (1979)
6. Dillinger, P.C., Manolios, P.: Bloom filters in probabilistic verification. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 367–381. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30494-4_26
7. Fearn, N.: Orange deploys LoRa network in thousands of French towns, September 2016. https://internetofbusiness.com/orange-lora-network-france/
8. Hunt, D., Jouko, N., Ridder, M.: LoRa Alliance - End Device Certification Requirements for EU 868MHz ISM Band Devices, v1.2 (2016)
9. Kim, G.: Tata to deploy LoRa network for IoT, June 2016. http://spectrumfutures.org/tata-to-deploy-lora-network-for-iot/
10. Kinney, S.: 100 US cities covered by Senet LoRa network for IoT, June 2016. http://www.rcrwireless.com/20160615/internet-of-things/100-u-s-cities-covered-senet-lora-network-iot-tag17
11. L'Héréec, F., Joulain, N.: Sécurité LoRaWAN. In: Computer & Electronics Security Applications Rendez-vous - C&ESAR (2016)
12. Lifchitz, R.: Security review of LoRaWAN networks. In: Hardwear.io (2016)
13. LoRa Alliance: LoRaWAN Certified Products. https://www.lora-alliance.org/certified-products. Accessed 8 Dec 2017
14. LoRa Alliance Technical committee: LoRaWAN Regional Parameters, LoRa Alliance, version 1.0, July 2016
15. LORIOT.io: https://www.loriot.io
16. Marek, S.: SK Telecom & KPN Deploy Nationwide LoRa IoT Networks, July 2016. https://www.sdxcentral.com/articles/news/sk-telecom-kpn-deploy-nationwide-lorawan-iot-networks/2016/07/
17. Mason, J., Watkins, K., Eisner, J., Stubblefield, A.: A natural language approach to automated cryptanalysis of two-time pads. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, pp. 235–244. ACM (2006). http://doi.acm.org/10.1145/1180405.1180435
18. Miller, R.: LoRa the explorer - attacking and defending LoRa systems. In: Information Security Conference - SyScan360 (2016)
19. National Institute Of Standards and Technology: NIST FIPS 197 Specification for the Advanced Encryption Standard (AES), November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
20. National Institute Of Standards and Technology: NIST Special Publication 800–38A Recommendation for Block Cipher Modes of Operation - Methods and Techniques, December 2001. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

21. National Institute Of Standards and Technology: NIST Special Publication 800–38B Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
22. SmartCitiesWorld: IoT connectivity for 100 million homes in China, November 2016. https://smartcitiesworld.net/news/news/iot-connectivity-for-100-million-homes-in-china-1139
23. SmartCitiesWorld: LoRaWAN IoT network deployed in Japan, September 2016. https://smartcitiesworld.net/connectivity/connectivity/lorawan-iot-network-deployed-in-japan
24. SmartCitiesWorld: Semtech LoRa chosen for new IoT network in New Zealand, September 2016. https://smartcitiesworld.net/news/news/semtech-lora-chosen-for-new-iot-network-in-new-zealand-949
25. Song, J.H., Poovendran, R., Lee, J., Iwata, T.: The AES-CMAC Algorithm. RFC 4493, June 2006
26. Sornin, N., Luis, M., Eirich, T., Kramp, T., Hersent, O.: LoRaWAN Specification, LoRa Alliance, version 1.0.2, July 2016
27. The Things Network. https://www.thethingsnetwork.org
28. Yang, X.: LoRaWAN: Vulnerability Analysis and Practical Exploitation (2017). https://repository.tudelft.nl/islandora/object/uuid:87730790–6166-4424-9d82-8fe815733f1e/datastream/OBJ/download
29. Z-Wave Alliance: Z-Wave specification. http://z-wave.sigmadesigns.com/design-z-wave/z-wave-public-specification/
30. ZigBee Alliance: ZigBee specification. http://www.zigbee.org/download/standards-zigbee-specification/
31. Zulian, S.: Security threat analysis and countermeasures for LoRaWAN join procedure (2016). http://tesi.cab.unipd.it/53210/

Not So Predictable Mining Pools: Attacking Solo Mining Pools by Bagging Blocks and Conning Competitors

Jordan Holland, R. Joseph Connor, J. Parker Diamond, Jared M. Smith, and Max Schuchard(✉)

Department of Electrical Engineering and Computer Science,
UT Computer Security Laboratory, University of Tennessee, Knoxville, TN, USA
{jholla19,rconnor6,jdiamon3,jms,mschucha}@utk.edu
https://volsec.eecs.utk.edu

Abstract. In this paper we present three attacks against the predictable solo mining (PSM) scheme. In PSM, miners receive shares for submitting partially valid solutions to the current Proof of Work, adding those shares to their account. When the pool successfully mines a block, the block is awarded to the miner with the most shares, and the rewarded miner "pays" an amount of shares equal to the next highest miner's to claim the block. Our attacks take advantage of the fact that the amount of shares expended winning two different blocks, which have the *same monetary value*, can vary by up to a factor of *four*. We show that by strategically spreading its shares across multiple accounts, a malicious miner can generate more revenue than a naive miner of the same computational power by only claiming blocks with a low share cost. By doing so, a miner can reduce computational power it must expend to win a block by more than 30%. Our other two attacks reduce the profitability of victim miners in the pool by minimizing the gap between first and second place when the victim wins a block. This drives up the average amount of computational power the victim must contribute to receive a reward. An adversary not concerned with cost can reduce the number of shares a victim retains after winning a block by up to 26%. We also find that an adversary with more computational power than their victim can reduce the number of shares the victim retains after winning a block by more than 8% with only limited impact on the adversary's profitability.

1 Introduction

Cryptocurrency mining represents a several hundred million dollar a year industry. As a point of reference, Bitcoin miners alone generated 563 million US dollars worth of revenue in 2016 [10]. The increasing popularity and profitability of cryptocurrency mining has resulted in an exponential increase in the total computational power dedicated to the task. For example, the cryptocurrency Ethereum has seen a more than 18 fold increase in overall network hashrate

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 272–287, 2018.
https://doi.org/10.1007/978-3-662-58387-6_15

between September 2016 and September 2017 [5]. This increase in overall network compute power means that individuals possessing small amounts of computational resources find themselves unable to consistently generate mining revenue. As a result, miners often opt to join mining pools where they aggregate their computational resources in an effort to receive a consistent, reliable income proportional to their contribution to the pool.

The *payout scheme* of a mining pool decides how to distribute the pool's revenue between individual miners. Ideally, a payout scheme should demonstrate *proportional fairness*, where miners receive rewards proportional to their contribution to the pool. Additionally, pool operators want a payout scheme that is *incentive compatible* between themselves and their miners. Specifically, the miner's best strategy should be that they dedicate all of their computational resources towards the pool, resulting in maximized revenue by both miner and pool operator. With the number of different mining pools increasing, so too has the number of different payout schemes that are implemented by these pools. Many of these schemes are not vetted for incentive compatibility or fairness, resulting in attacks against deployed payout schemes which break either proportional fairness or incentive compatibility [7,15,17].

In this paper we present three attacks against a relatively new payout scheme, the predictable solo mining (PSM) scheme. Unlike other payout schemes that split each won block between all mining participants, PSM awards the entire block to the miner that currently has saved the most "computational credit" or *shares*. The winning miner's shares are then adjusted to be the difference between their current shares and the next highest miner's shares. This payment scheme results in a variance in the number of shares "needed" to win a block, even when the pool's computational power is held constant. Our attacks take advantage of the fact that the amount of shares expended winning two different blocks, which have the *same monetary value*, can vary by up to a factor of *four*. By strategically adjusting our malicious miner's behavior, we can exclusively take advantage of highly efficient blocks, while at the same time forcing victims to more frequently accept less efficient blocks. The attacks we develop present evidence that this payout scheme is neither incentive compatible nor fair.

In our first attack, we show that by launching a Sybil attack against a pool using PSM, a malicious miner can generate more revenue than an honest miner of equivalent computational power. Additionally, we show that in an ecosystem of PSM pools, the PSM payout scheme is not incentive compatible, and encourages miners to spread computational resources between several pools to maximize returns of their computational power investment. We find that an adversary can trivially increase their profits by 10%, even if they just interact with a single PSM pool, and can increase their profits by more than 30% when they apply our strategy across multiple PSM pools. This means that miners are incentivized to spread their computational resources across several PSM pools, making the PSM mining scheme not incentive compatible between miner and pool operator. Our second attack shows that miners can *degrade* the profitability of victim miners by driving up the average amount of computational power a victim must

invest to receive a reward at a financial cost to themselves. We demonstrate than an adversary can reduce the number of shares retained by a victim after winning a block by up to 26%, reducing the victim's mining efficiency. Lastly, our third attack utilizes an altered variant of our original Sybil attack to degrade competitor profitability while at the same time maintaining the adversary's.

The rest of this paper is laid out as follows: Sect. 2 discusses the process of mining cryptocurrencies, mining pools, and payout schemes across mining pools; Sect. 3 examines the attacks we have constructed and lays out how they work; Sect. 4 discusses the simulator we built and the data we collected to test our various attacks; Sect. 5 examines our evaluation in carrying out our attacks; Sect. 6 dives into existing attacks against mining pools and where our contributions lie and the current state of integrity in the cryptocurrency ecosystem; finally, Sect. 7 summarizes what we have discovered and presents potential future work.

2 Background

2.1 Mining Cryptocurrencies

Cryptocurrency mining is the process of adding transaction records to to the currency's public ledger of past transactions. This ledger of past transactions is called the *blockchain*, and serves as a means to tracking valid transactions in the network. In order to add transaction records to the blockchain, miners must compute a resource-intensive *proof-of-work* (PoW) that serves to provide security guarantees against double-spending and ensure that transactions are tamper-resistant. In exchange for computing the PoW, miners are rewarded with a number of units of currency to compensate the miner [1].

In order to control the rate at which records are added to the blockchain, cryptocurrency networks can adjust the what is considered a valid solution to the PoW in an effort to require miners to spend more computational cycles to find one. The *network difficulty* is a relative measure of how difficult it is to find a new block, where the difficulty is adjusted periodically as a function of how much hashing power has been deployed by the network of all miners for a cryptocurrency.

2.2 Mining Pools

As a result of the large number of miners working on the most profitable cryptocurrencies, new miners often face a high barrier to entry due to the variability associated with successfully computing solutions to PoW. Miners with small hashrates relative to the largest miners have a lesser probability of mining a block, which causes the variance in its overall profitability to increase. Additionally, as the mining difficulty of the network increases so too does the variability, meaning that individual miners can expect to mine for long periods without any reward.

To mitigate this variability in rewards, cryptocurrency miners can join a *mining pool* which aggregates computing power (i.e. collective hashing power),

thus lessening the variability in finding blocks. Pools distribute the rewards of successfully mined blocks to their member miners regardless of whether the miner was the actual node to solve the PoW. This scheme allows smaller-scale miners who would otherwise receive only sporadic rewards to instead receive a steady, reliable income. In exchange, pool owners will collect a small fee from each block found by the pool before distributing the remaining rewards to pool participants.

Pools attempt to distribute rewards proportionally to each miner based on their contribution to the pool's overall hashrate. Therefore, pools need a secure manner to estimate the computational power of each miner. To accomplish this, miners occasionally submit *shares*, solutions to the PoW for the current block that are at a lower difficulty than the network difficulty, to their pool. Miners find shares in the process of hunting for a full solution to the PoW. The frequency and difficulty of the shares submitted to the pool serves as a proxy for the miner's computational power. When a block is found, the pool uses a pre-determined *payout scheme* that determines how to allocate the reward of the found block to the miners in the pool based on submitted shares.

2.3 Mining Pool Payout Schemes

Various payout schemes have been used in mining pools, as explored by Rosenfeld et al. [17]. The pay-per-last-N shares (PPLNS) scheme and predictable solo mining (PSM) are two examples of current mining pool payout schemes. PPLNS is used in the largest Ethereum mining pool [4], Ethermine.org, and PSM is used in another major Ethereum mining pool [6], Ethpool.org, as well as Bitcoin, Litecoin, ZCash, DogecCoin, ZClassic, Komodo, and Hush solo mining pools [2,9,16,19]. The two schemes work as follows:

- **PPLNS Scheme:** This reward system is round based, where one round is an arbitrary number of minutes. When a block is found by the pool, the block reward is distributed according to the number and difficulty of the shares submitted by each during the last hour. Payout takes place immediately after the minimum payout amount of 1 coin has been reached.
- **PSM Scheme:** In this scheme, each submitted share will increase the credits of the miner who submitted the share by the share difficulty. The miner who accumulates the most credits receives the reward of the next mined block and their credits will be reset to their current credits minus the credits of the runner-up miner. Re-setting the credits of the miner who did receive the block reward to 0 was abandoned as it did penalize miners having an above average hashrate. Typically, a miner will receive a full block reward as soon as their accumulated credits equals the current block difficulty (+/− pool luck) [6].

3 Attacking PSM Pools

In a PSM pool, the entire block reward is awarded to the miner with the most shares any time the pool finds a block, and the winner's shares are then decreased

by the number of shares held by the miner with the second-most shares. Therefore, the "cost" of a block reward can be characterized by the number of shares held by the second-place miner at the time the block is mined by the pool. Likewise, the efficiency of a miner can be characterized by the number of blocks won per share contributed.

The average cost of a block in a PSM pool is equal to the network difficulty [6]. However, because the mining process is random, the time between mined blocks in the pool varies. As a result, the number of shares that miners have when a block is found (and by extension, the cost for the block reward) also varies.

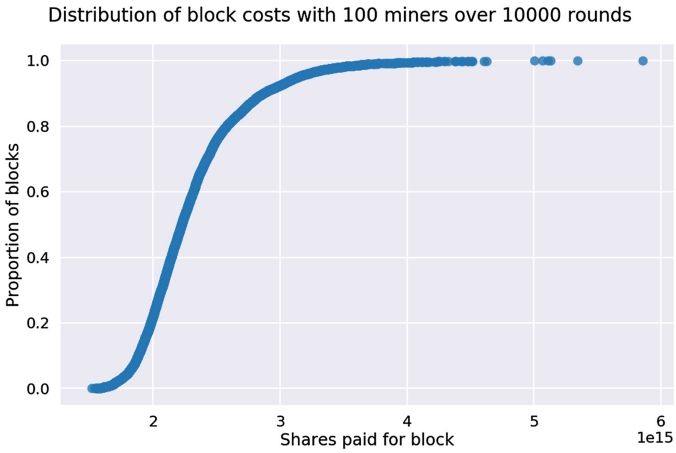


Fig. 1. The average cost per block is equal to the network difficulty (approximately 2.3×10^{15} in Ethereum at the time of writing), but block costs are distributed randomly.

3.1 Minimizing the Cost of Winning Blocks

A key observation to make about a PSM pool is that a miner can never pay more shares for a block than the miner current has in their account. Miners who naively submit all found shares to a single account in a single PSM pool can expect to see block costs drawn from the population seen in Fig. 1, resulting in an average cost that is approximately equal to the network difficulty. However, by refusing to place more shares into their account, a malicious miner will only ever win the cheapest cheapest blocks produced by the pool. A miner who uses this strategy can expect to pay less per block, on average, compared to a naive miner with a comparable hash rate.

A miner who employs this strategy may have leftover computing power after filling its account to the target number of shares. When the miner succeeds at filling an account to the target number of shares, the miner in turn begins crediting shares into a new account, filling it the target number. Our adversaries

goal is to always have an account at the target number, meaning that if a block would be claimed for that value or lower, they will always win it, paying exactly their targeted cost.

A natural risk of the cost minimization strategy is that blocks at or bellow the adversary's particular cost threshold will be insufficiently frequent. If our adversary can fill accounts to the target number of shares faster than blocks at that cost appear then some of those accounts will be wasted. In other words, although a miner may decrease their average cost per block, they may win fewer total blocks if any of their hashing ability goes unused. Since these shares do not directly contribute to the adversaries revenue, the adversary has no motivation to generate them; we term such shares *withheld shares*.

In order to counteract this loss of throughput the adversary can adopt one of two strategies. First, the adversary can increase the cost it is willing to pay for blocks, reducing its efficiency, but increase its throughput. Alternatively, the adversary can spread computational power across multiple pools, attempting to claim inexpensive blocks from several different pools. As long as the miner can submit shares to alternate pools, it may use the cost minimization strategy without a loss in throughput. In order to show that the cost-minimization strategy is advantageous for miners in practice, we also demonstrate that realistic miners can achieve a significant reduction in per-block cost without withholding too many shares. In particular, for a miner with access to N mining pools, the maximum acceptable proportion of wasted shares is $1 - \frac{1}{N}$.

The total profit of a miner for a given time period can be calculated as:

$$\text{Profit} = \frac{\text{(Shares earned)}}{\text{(Share cost per block)}} \times \text{(Block reward)}$$

Assuming that a miner is never idle (that is, that there are sufficient alternate pools where leftover computing power can be used to earn shares), then an individual miner's shares are roughly proportional to the miner's hashing power (barring the effects of luck), which is assumed to be constant. The block reward is likewise assumed to be constant, as is the case in Ethereum. Therefore, a miner who lowers their average cost per block for a fixed time period can expect a greater total profit during that time.

Mining pools profit from the pool fees that are deducted for blocks mined by the pool, so the pool is solely interested in miners contributing as much hashing power as possible to the pool. If miners can obtain a greater payout by contributing certain shares to alternate pools, then the PSM payout model cannot be incentive-compatible. Furthermore, since the pool cannot expect to find blocks at an average cost that is less than the network difficulty, a single miner who wins blocks at an average cost that is less than the network difficulty necessarily increases the average cost for the other miners in the pool.

3.2 Increasing a Miner's Block Costs by Donating Shares

Malicious miners in PSM pools can abuse the lack of integrity on share submissions in order to inflate a target miner's average block cost. Consider the "cost"

of each block as the number of shares that the second place miner has when a block is found. An adversary can submit shares under another miner's public key to the corresponding pool, inflating that miner's share count. Our adversary can increase a victim's costs by artificially closing the gap between the victim and the miner in second place at the time that the victim wins a block. By consistently minimizing the difference between the target miner and this runner up we can effectively define the cost of the block for the target, and increase the target miner's average block cost. Since the malicious miner is donating shares to other miners, this attack will cost the adversary money, while also reducing the victim's profitability.

3.3 Multiple Account Idling to Drive Up Target Miner Block Cost

The previous attack is intuitive and effective, but a malicious miner can do almost as well in driving up the cost of a target miner without donating all of their work to other miners. Furthermore, we can do this even the pool has implemented integrity-checking measures. This attack involves having at least two accounts in the same pool and spreading one's submitted shares among those accounts. Consider the "cost" of a block being the number of shares the second place miner has when a block is found. Increasing the average cost of a target miner then involves minimizing this difference. By driving an attack account up near the top of leaderboard of the pool and then "idling" at a specific rank, the attack account can wait until the target is in range to attempt to position himself in second by a minimum number of shares in the case of a target miner victory.

While idling the attack account, the adversary can offloads extra shares to an offload account, slowly driving that account up the leaderboard. Once the target passes the attacker, the malicious miner then uses all shares it finds to constantly minimize the gap between itself and the target. By riding the target miner up the few spots left in the leaderboard, the attacker is able to minimize the gap between it and the target at the time of the target miner's victory, and thus define the cost of the block for the target.

Once the target miner has won, on a successful attack the attacker should be at the top of the leaderboard at the start of the new block. The attacking account now uses all of its hashing power to win the first block it can. After the attacker wins a block he switches to the offloading account, making it the new attacker account. Due to the offloading of shares while "idling" the new attacking account is in a much better position to get back to the "idling" state where the attack on the target can be launched. By spreading out across multiple accounts (ultimately as many as needed) the attacker is able to almost constantly be in an "attacking" position given a higher hash rate than the target.

4 Experimental Setup

4.1 Simulation Methodology

To evaluate our attacks against the integrity of PSM pools, we have built a generic discrete mining pool simulator, which is publicly available at [11]. We use random probabilistic distributions to model both the rounds in which blocks are found as well as the shares distributed to each miner based on their total hash rate, the average difficulty, and the number of shares we wish each miner to receive per second. Most documented configurations of real-world mining pools can be configured in our simulator by manipulating the following parameters:

- **Number of Honest Miners:** the number of miners which model generic, non-malicious miners that contribute all shares available per round to their own total.
- **Number of Malicious Miners:** the number of miners which behave maliciously in one of several considerations, depending on the chosen attack. These miners are global adversaries and can see all members of the mining pool, including each miner's current shares, and have the ability to make predictions about the next round's accumulated shares for a given miner. Malicious miners can freely distribute their valid shares to other miners or completely separate pools in any manner they see fit. Finally, malicious miners can make accurate guesses on when the next block will occur, and use this information to make informed decisions about what to do with their currently valid shares.
- **Miner hash rates:** the distribution of hash rates both the honest and malicious miners should pull from, which come from both real-world PPNS and PSM pools. The collection of this data will be discussed in the next section, Sect. 4.2.
- **Number of block rounds to simulate:** the number of blocks our simulator should simulate being found by the combined power of the miners in the pool. Note, there is not necessarily a 1-to-1 mapping of simulated rounds and rounds where blocks are found; in general, there can be thousands of rounds for every block round.
- **Round length:** the round length in simulated seconds that each round should last. This value informs the calculation of the block rounds and the share distribution to each miner per round by influencing the difficulty to find blocks.
- **Shares per second:** the number of shares per second on average that a miner should find. We use this value when calculating share distribution per round for each miner.
- **Network difficulty:** the total difficulty of finding a block for the given pool, which is pulled from the real-world pool being simulated. If we are simulating a PSM pool, we pull this from the Ethpool API; otherwise, if it is a PPLNS pool, we pull this from the Ethermine API.

To represent realistic pools, we use known statistical distributions for modeling both the rounds where blocks will be found and the distribution of shares to miners per round. These distributions are calculated as follows:

- **Finding Blocks:** We model finding blocks using a geometric distribution based on the configured round length, the total pool hash rate as given by summing the hash rates of all honest and malicious miners in the pool, and the average network difficulty of the Ethereum network measured as the number of hashes required on average to find a block. We calculate the probability of finding a block in a given round as:

$$P(finding\ block) = \frac{round\ length \times total\ pool\ hash\ rate}{network\ difficulty} \tag{1}$$

  We then sample n samples from a geometric distribution of Bernoulli trials, where n is the number of blocks we wish to find from the parameters above, and the probability is calculated from Eq. 1.
- **Distributing Shares to Miners:** We model the distribution of shares per round to each miner with a binomial distribution based on the miner's hash rate, the desired shares per second, and the configured round length. We first calculate the difficulty of getting shares by Eq. 2.

$$D(shares) = \left\lfloor \log_2 \frac{miner\ hash\ rate}{shares\ per\ second} \right\rfloor \tag{2}$$

  Then, we get the actual probability of the miner finding a valid hash in one second by Eq. 3 using the difficulty we just computed.

$$P(finding\ hashes) = \frac{miner\ hash\ rate}{2^{difficulty}} \tag{3}$$

  Next, we derive the shares the miner should find each round by sampling from a binomial distribution based on n trials, where n is the round length in seconds, and the probability computed in Eq. 3 and multiplying it by $2^{difficulty}$, where the difficulty comes from Eq. 2.

This generic simulator gives us a way to simulate the interactions of honest and malicious miners over any arbitrary number of blocks as well as various pool types. In the next section, we discuss our method to collect data from existing, real-world pools to use as our miner hash rates. In the sections following, we discuss how we have used our simulator to find novel attacks on PSM pools that are not incentive-compatible and can be or may already be actively exploited on the largest Ethereum mining pools.

4.2 Collecting Real-World Miner Data

Our simulator uses hash rates and difficulties collected from the Ethpool and Ethermine APIs, where Ethpool is a PSM mining pool and Ethermine is a

PPLNS mining pool. Using the miner addresses that Ethpool and Ethermine use to pay miners, we were able to find miners that were paid by the mining pools on the Ethereum block-chain via geth [8], the official Ethereum command-line interface. The paid miners' addresses were then queried against the Ethpool and Ethermine APIs to locate active miners, their hash rates, and the difficulty of the Ethereum network. At the time of writing, Ethpool has roughly 1,000 active miners, and Ethermine has approximately 40,000 active miners.

5 Evaluation

5.1 Cost Minimization Attack

32 simulations were run for pools of 100 miners for with hash rates drawn at random from the real distribution of miners in Ethpool. In each case, the simulation was allowed to run until the pool found 10000 blocks. In each run, a single miner used the "cost minimization" strategy by mining until reaching a fixed target (75% of the expected block cost) and then stopping. The remainder of the miners naively contributed every share found immediately. Miners whose hash rates were so small as to never win a block for the duration of the simulation are omitted. Results are shown in Fig. 2.

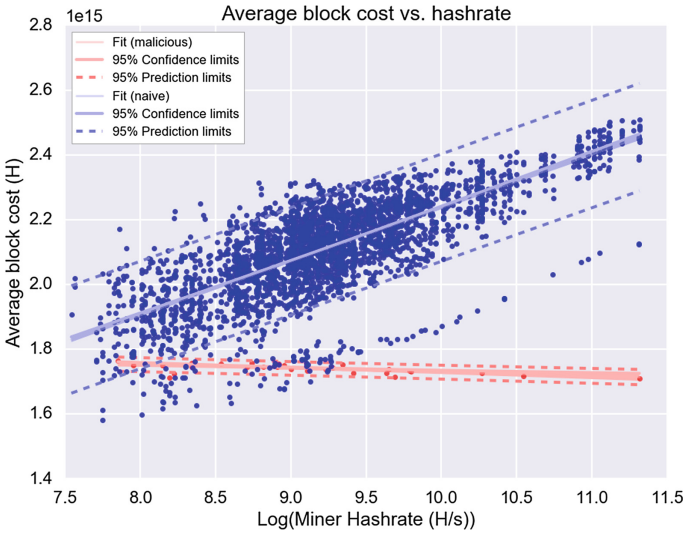


Fig. 2. Average cost per block as a function of miner hash capacity, comparing naive (blue) and malicious (red) miners. (Color figure online)

Figure 2 shows that the PSM payout scheme already gives an advantage to miners with lower hash rates. The average block cost for an honest miner increases proportionally with the logarithm of the miner's hash rate. But even

many lower-than-average miners can expect to decrease their average cost per block with the cost minimization attack, compared to honest miners with the same hash rate. Unlike normal PSM schemes where miners with higher hash rates are less efficient, our malicious miner becomes more efficient as hash rate grows. This is a result of the miner being able to more rapidly fill accounts to the target share value, allowing the miner to take advantage of several inexpensive blocks in short succession.

A second set of simulations was run to demonstrate that this attack is practical at a significant advantage for a typical miner without reducing throughput, assuming access to only a small number of alternate pools. Each simulation from this set used a fixed set of 100 miners with 1 malicious and the remainder honest. The malicious miner's hash rate for these simulations was fixed at the median hash rate from Ethpool, to represent a typical miner. The malicious miner's target block cost was varied across simulations to observe the relationship between the average block cost and the proportion of shares not submitted to the pool.

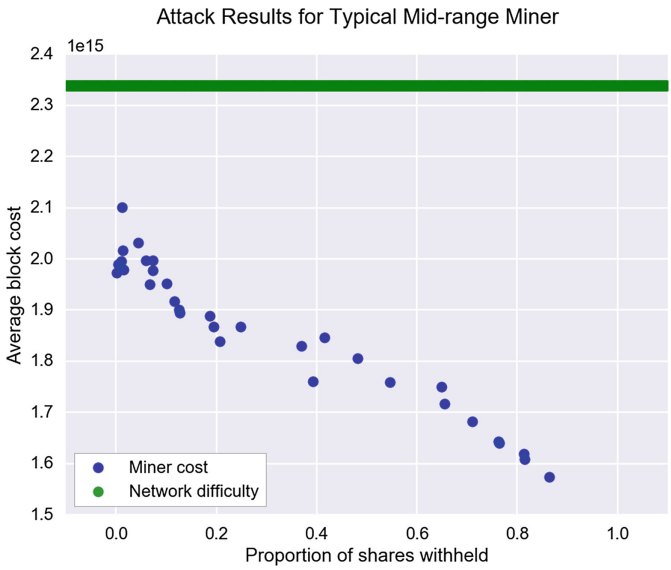


Fig. 3. Average block cost as a function of shares not contributed to the pool for a median Ethpool miner (1.3 GH/s) with network difficulty 2.3×10^{15}. A naive miner would have an average block cost equal to the network difficulty in PSM pools.

Figure 3 shows that a typical miner can expect to reduce their average cost per block from approximately 2.1×10^{15} hashes to 1.8×10^{15} hashes by withholding only 50% of its shares. Assuming the existence of only a single alternative mining pool of comparable size, this malicious miner can achieve this reduction without any wasted hashing power, by contributing the leftover hashing power to the alternate pool.

5.2 Malicious Donation Attack

An antagonistic miner can inflate the cost of a victim miner by allocating shares to miners immediately below the victim in shares when the victim is about to win a block in the PSM pool. As the victim miner increases its shares, the malicious miner can give its own shares to the runner-up miner in order to decrease the difference in shares between the victim and the runner-up to 1. This ensures that the victim almost always has no shares to put towards the next block. Under this scenario, the attacker saves no shares for itself and gives shares to any miner so long as it closes the gap between the victim – when the victim is about to win a block – and the next miner.

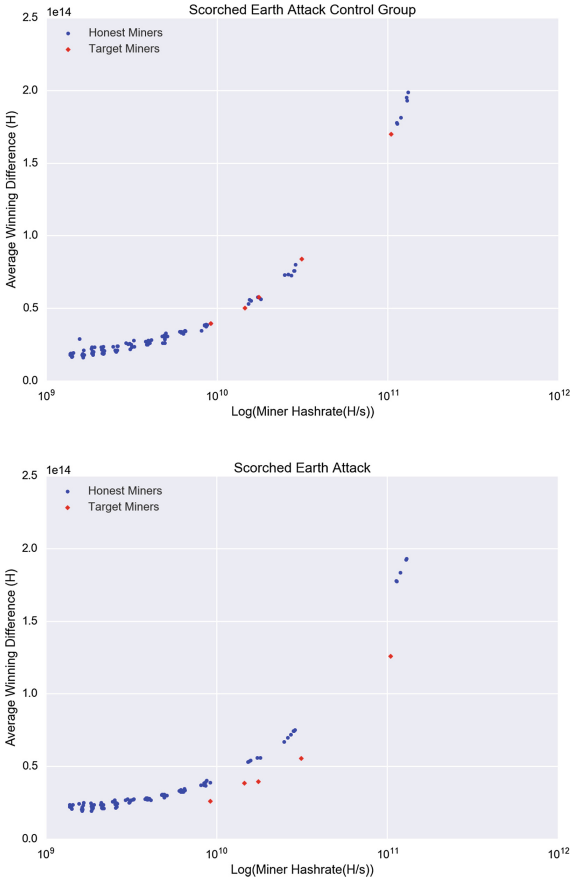


Fig. 4. Average shares remaining after winning a block without (left) and with (right) a scorched-earth attacker

Figure 4 shows that by using the above strategy, an attacker can consistently force a victim to spend all shares available to win a block, or to put it differently

the gap between the victim and the second place miner is always a difference of 1 share. The amount of shares left over after winning a block generally follows an exponential increase with respect to hash rate. The aforementioned figure shows that the victim miner's left-over shares decrease by roughly 26% in the presence of this attack, given this distribution of 100 hash rates. Over time this decrease in the amount of shares available for the next block will ultimately increase the average block cost required for the victim to obtain a block. This assumes that the malicious miner values punishing the victim more than making money, so this study disregards the performance of the attacker under this scenario.

5.3 Idling Attack

A malicious miner can increase the average block cost of a target miner in PSM mining pools even if the pool has integrity measures implemented, provided the attacking miner has a sufficiently large hash rate in comparison with the target. By situating multiple accounts strategically in the pool, the malicious miner can be in second place almost every time the target miner is close to winning a block. This in turn means that the adversary can ensure that the victim miner *always* has their shares reset to zero after winning a block.

Table 1. The average decrease in a victim's gap when it wins a block by a computationally stronger adversary launching our "idling" attack.

Attacker/target ratio	% Decrease in average winning difference
1.2	.03
4.2	5.02
7.5	6.31
9.0	5.6
14.2	8.36

Table 1 shows that by using this attack a miner can cause the target to have a smaller average number of leftover shares after winning a block. The data for this table was gathered by running a control group of 100 miner's for 10,000 blocks, 10 times. Afterwards, we ran the same group of 100 miners, taking control of the miner with the highest hashrate in the pool, making it our attacker. Each attack scenario was then ran 10 times for 10,000 blocks, and then averaged out over the different runs. This results in raising the target miner's average block cost, and therefore reduces the miner's profits. The only cost incurred by the malicious miner is when the malicious miner accidentally exceeds the victim's shares essentially attack themselves. Our reported results are with a hand optimized algorithm. We believe that with some machine optimization the attacker could further drive up the target's average block cost at a very small overall cost to himself.

6 Related Work

Other attacks have been proposed against different mining pool payout schemes which break incentive compatibility and fairness. Rosenfeld et al.'s work [17] examines several payout schemes including Pay-Per-Share (PPS) and Pay-Per-Last-N-Shares (PPLNS), exploring their vulnerability to *pool-hopping* attacks. In a pool hopping attack an adversarial miner jumps between several pools hoping to capitalize on instances where any one of those pools successfully mines a block earlier than expected. This attack has a similar end goal to our cost minimizing attack in Sect. 3.1, however our attack can achieve greater efficiency if multiple pools exist to spread attacker resources across, it can still provide the adversary gains *even if only one pool exists.*

Concurrent work recently published by Zamyatin et al. [20] also examined PSM pools[1]. Several key differences exist between our work and Zamyatin et al.'s. First, our attacks utilize different properties than Zamyatin's work. Our cost-based attack described in Sect. 3.1 successfully wins blocks at a minimum cost, rather than building up a large gap between the malicious miner and the second-place miner. This allows our attacks to be more efficient than Zamyatin's in terms of average work performed per winning block by between 10% to 30%. Increasing other miner's block costs via multiple idling accounts, presented in Sect. 3.3, is a new attack not proposed by Zamyatin and does not require donating shares to other miners not controlled by the adversary. As a result, our idling attack, unlike their tactical donation attack, can not be trivially defeated by adding authentication to the mining pool. Lastly, in Zamyatin. et al.'s work, the actual attack simulations were conducted with a mining pool consisting of only *two* miners, a high hashrate and low hashrate miner. As described in Sect. 4, our work evaluates attacks with realistic simulations based on representative population of miners and hashrates taken the Ethpool API. Our more accurate model better captures the dynamics of what an adversary needs to accomplish to realize the attacks presented in our work.

Besides these related attacks, a wide body of literature on payout schemes exists. Many related papers, for example work by Lewenberg et al. [14] and Schrijvers et al. [18], examine cryptocurrency mining pools from a game-theoretic perspective. These works often focus on theoretical constructions of mining pools involving limited players. To our knowledge, no such game theoretic analysis has been conducted specifically on PSM pools. Other studied attacks include denial-of-service attacks [12,13] and withholding attacks [3,7,15], where malicious pools attack rival pools and damage their reward by withholding valid blocks.

7 Conclusions

In this paper we have presented three separate attacks against the predictable solo mining scheme. The variability in the cost of the blocks in this scheme leave it vulnerable to a malicious miner strategically gaining an advantage. First, we

[1] In Zamyatin's work, these pools were called *queue-based* pools.

showed that a miner that strategically spreads out its shares across multiple accounts can generate more revenue than a naive miner with the same computational power by only claiming blocks below the average share cost. The second attack presented shows that an adversary, not concerned with cost, can reduce the shares a victim retains after winning a block, reducing their profitability. Our third attack on the PSM scheme shows that even on pools with integrity implemented, an adversary can reduce the number of shares a victim retains limited impact to his overall profitability. In conclusion, we recommend the PSM scheme not be implemented in any new pools and any pools currently using the scheme change to another, more incentive compatible and fair, payout scheme.

References

1. Bitcoin.it: Bitcoin Mining (2017)
2. Kolivas, C.: Anonymous Solo Bitcoin Mining for Everyone (2017)
3. Courtois, N.T., Bahack, L.: On subversive miner strategies and block withholding attack in bitcoin digital currency. arXiv.org (2014)
4. Ethermine.org: Ethermine (2017)
5. Etherscan.io: Ethereum Network HashRate Growth Rate (2017)
6. Ethpool.org: Credits on Ethpool (2017)
7. Eyal, I.: The miner's dilemma. In: 2015 IEEE Symposium on Security and Privacy (2015)
8. Go-ethereum: Geth (2017)
9. Hellcatz: Solo Mining Pool for ZCash, Hush, Komodo, Zclassic, and Zen (2017)
10. Hileman, G., Rauchs, M.: Global cryptocurrency benchmarking study. Cambridge Centre Altern. Finan. (2017)
11. Holland, J., Connor, J., Diamond, P., Smith, J., Schuchard, M.: Aminingpoolsimulator - mining pool simulator. https://github.com/VolSec/aminingpoolsimulator (2017)
12. Johnson, B., Laszka, A., Grossklags, J., Vasek, M., Moore, T.: Game-theoretic analysis of DDoS attacks against bitcoin mining pools. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014. LNCS, vol. 8438, pp. 72–86. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44774-1_6
13. Laszka, A., Johnson, B., Grossklags, J.: When bitcoin mining pools run dry. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) FC 2015. LNCS, vol. 8976, pp. 63–77. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_5
14. Lewenberg, Y., Bachrach, Y., Sompolinsky, Y., Zohar, A., Rosenschein, J.S.: Bitcoin mining pools: a cooperative game theoretic analysis. In: International Foundation for Autonomous Agents and Multiagent Systems, May 2015
15. Luu, L., Saha, R., Parameshwaran, I.: On power splitting games in distributed computation: the case of bitcoin pooled mining. In: 2015 IEEE 28th Computer Security Foundations Symposium (2015)
16. NiceHash: NiceHash Solo Mining Pool for Multiple Currencies (2017)
17. Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems, December 2011
18. Schrijvers, O., Bonneau, J., Boneh, D., Roughgarden, T.: Incentive compatibility of bitcoin mining pool reward functions. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 477–498. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_28

19. TBDice: Anonymous solo Litecoin and Dogecoin mining pool based on ckpool (2017)
20. Zamyatin, A., Wolter, K., Werner, S., Mulligan, C.: Swimming with fishes and sharks: beneath the surface of queue-based ethereum mining pools, 20 September 2017. sba-research.org

Applied Cryptography

Practically Efficient Secure Distributed Exponentiation Without Bit-Decomposition

Abdelrahaman Aly(✉), Aysajan Abidin, and Svetla Nikova

imec-COSIC KU Leuven, Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
{abdelrahaman.aly,aysajan.abidin,svetla.nikova}@esat.kuleuven.be

Abstract. Bit-decomposition is a powerful tool which can be used to design constant round protocols for bit-oriented multiparty computation (MPC) problems, such as comparison and Hamming weight computation. However, protocols that involve bit-decomposition are expensive in terms of performance. In this paper, we introduce a set of protocols for distributed exponentiation without bit-decomposition. We improve upon the current state-of-the-art by Ning and Xu [1,2], in terms of round and multiplicative complexity. We consider different cases where the inputs are either private or public and present privacy-preserving protocols for each case. Our protocols offer perfect security against passive and active adversaries and have constant multiplicative and round complexity, for any fixed number of parties. Furthermore, we showcase how these primitives can be used, for instance, to perform secure distributed decryption for some public key schemes, that are based on modular exponentiation.

1 Introduction

The use of Internet connected devices has become essential in people's daily lives. However, serious privacy concerns have been raised as more and more sensitive private information is transmitted (over the Internet), processed and stored in third party databases or the cloud. This movement of information has been fostered by decades of research in cryptography, helping to develop tools and techniques to perform computational tasks in a privacy-friendly manner. Secure multiparty computation (MPC) is one such tool that allows the computation of functions on private inputs by mutually non-trusted parties. The initial techniques and problems proposed in the seventies and eighties have indeed evolved into a growing field with not only theoretical results but into the implementation of practical applications (e.g., [3–5]). More recently, thanks to the advent of frameworks such as SPDZ [6] or MASCOT [7], it is possible to implement any functionality in a relatively *efficient* fashion.

One of the basic and commonplace problems in applied MPC is distributed modular exponentiation. Classical approaches for distributed modular exponentiation over arithmetic circuits rely on bit-decomposition, which was first proposed by Damgård et al. in [8]. However, bit-decomposition is an expensive

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 291–309, 2018.
https://doi.org/10.1007/978-3-662-58387-6_16

procedure in terms of performance. Recently, Ning and Xu proposed protocols for modular reduction and exponentiation without bit-decomposition in [1,2]. In this paper, we build upon [1,2], and present mechanisms to compute modular exponentiation in a simpler and practically efficient fashion. Our approach does not require exhaustive parallelization and has a constant round complexity. Finally, we show how our approach can be used in, for example, distributed decryption of public key protocols.

Bit-Decomposition. The process of decomposing secretly held finite field elements into their bit-representations for exponentiation was originally introduced by Damgård et al. [8]. The same decomposition mechanism was then used not only to construct secure exponentiation mechanisms, but also as a basic building block for (in)equality tests and modular operations [9]. Although highly practical, the cost of operating bitwise on arithmetic circuits cannot be dismissed. An extended polynomial representation, such as a bit extension, implies a large bound on the amount of work i.e. multiplication performed.

It can be argued that in many cases involving bit-decomposition, the operations are not co-dependent and hence can be parallelized (low round complexity). Even if we put aside the increase in transmission costs due to increased number of shares to be transmitted, the fact that the total amount of work (amount of multiplications) depends on the input size, cannot be addressed only by parallelization. For example, 4096 concurrent multiplications would require a batching structure composed of several threads to compute the multiplications not only in a single round, but in an equivalent amount of CPU time. This would necessitate the need for developing practically efficient mechanisms for modular operations, such as exponentiation, without relying on bit-decomposition.

Related Work. As previously mentioned, initial alternatives focus on bit-decomposition. Namely, the work introduced by Damgård et al. [8], that offered security against both active and passive adversaries for all three cases that we consider in this paper. However, for the case concerning the publicly available exponent, certain adaptations are needed to achieve security against malicious adversaries. It also requires the production of additional randomness r and r^a, where a is the public exponent, which would require additional communication rounds. Our protocol gets rid of such requirements and offers security against malicious adversaries.

The other two exponentiation protocols introduced in [8] make use of bit-decomposition, which has the aforementioned performance drawbacks. None of our protocols require bitwise operations, and optimizes the process in a simple straight-forward fashion. The main advantage of our approach is the reduction of non-linear operations which are, in general terms, expensive [10], as well as the decoupling the protocol complexity from the input size.

Recently, Ning and Xu [1,2], introduced several protocols aimed to remove the necessity of decomposing secret inputs into their bit representations. They achieve this by using a series of constructions based on bitwise operations over

some randomness. Their work achieves constant round, with a linear asymptotic bound on the amount of work (on the size of input). We not only simplify their process, but also reduce round complexity making use of constant amount of multiplications.

On a similar line, Grassi et al. introduced a mechanism to perform secure exponentiation over a publicly available base to a secret shared exponent in [11]. Their protocol was used in the context of the implementation of symmetric key primitives. Their protocol, however, was designed such that the output of the protocol is disclosed to the computational parties every time. Our protocols, on the other hand, allow the parties to keep the output secret and produce the output only when needed.

Note that, although the input size is not a factor in any of our protocols' complexity, this is not the case for the number of parties performing the computation. However, it is safe to assume this number remains unaltered throughout common practical applications giving room to specialized frameworks for a constant number of parties; see, e.g., [3,5,12,13]. This is not the case for the input size, given their variability in practical applications.

Our Contribution. We propose distributed modular exponentiation protocols without bit-decomposition, for the following configurations:

- **Public Base:** the base is public and the exponent is secret;
- **Public Exponent:** the base is private and the exponent is public; and
- **Private Exponentiation:** both the base and the exponent are privately held.

Note that, since the output of these functionalities would remain secret, our protocols can be used as sub-routines for more complex functionalities. This is indeed congruent with the literature on this topic e.g., [1,2], without prejudice to its security under composition [14]. Our protocols outperform their most recent counterparts in terms of round complexity and amount of work, which in our case are both constant for a fixed number of parties. Table 1 contrasts our results with the results by Ning and Xu [2]. Note that in this context l stands for the bit-size of the input and n for the number of parties.

As an application, we explore the case where decryption of common public key encryption schemes (such as RSA [15] and El-Gammal [16]) performed over MPC. The scenario that we consider is as follows: a user encrypts a message using a public key pk, corresponding to a private key sk, such that pk is publicly available and sk is secretly shared among several parties. An MPC decryption would not only facilitate what is commonly called threshold decryption, a powerful technique used for example in voting systems [17], but also transforms any ciphertext into secretly held shares of the message.

Outline. This work is organized as follows: We introduce the necessary background material in Sect. 2. We then give a detailed description of our results on exponentiation in Sect. 3. We provide an overview on the usage of our protocols for public key decryption in Sect. 4. Section 5 concludes the paper.

Table 1. Round/multiplication complexity for modular exponentiation without bit-decomposition.

Protocol	Semi-honest [2]		Semi-honest (this work)	
	Rounds	Multiplications	Rounds	Multiplications
$\texttt{exp}([b], a)$	5	33 $(10 \cdot n + 3)$	8	8 $(4 + 2 \cdot \lfloor log(n) \rfloor)$
$\texttt{exp}(b, [a])$	N/A	N/A	3	3 $(1 + \lfloor log(n) \rfloor)$
$\texttt{exp}([b], [a])$	20	$157 \cdot l + 7 \cdot l + 7 + 5$	13	13 $(7 + 3 \cdot \lfloor log(n) \rfloor)$

2 Preliminaries

2.1 Notation

We follow the square notation introduced in [18]. Let $[x]$ denote a secretly shared input x. Let P be the set of n parties, P_i be the i-th party for $i = 1, \cdots, n$ and q be a large prime number. Let $\mathbb{Z}_q^*$ be the multiplicative group $\mathbb{Z}_q - \{0\}$. To distinguish secretly shared elements of $\mathbb{Z}_q$ from that of $\mathbb{Z}_q^*$, we use $[x]_q$ for $x \in \mathbb{Z}_q$ and $[x]_{q^*}$ for $x \in \mathbb{Z}_q^*$. Additionally, consider $p = q - 1$, such that we can define $\mathbb{Z}_p$ and $\mathbb{Z}_p^*$ accordingly. Furthermore, our protocols make use of the infix notation, to make calls to the secure functionality e.g. $[z] \leftarrow [x] + [y]$. Note that in practice, basic operations are carried out by the underlying MPC protocol e.g., [6,7,13,19,20]. Our exponentiation protocols do not make direct use of any specific representation for signed values. However, other protocols such as the *modular* operation used in later sections may need such support definitions.

To denote a shared value x we use $[x] \leftarrow \texttt{Share}(x)$. To denote open or reconstruct the shared value we use $x \leftarrow \texttt{Open}([x])$. Note that for simplicity, we assume, in this work, that the inputs have been secretly shared using Shamir's Secret Sharing Scheme [21] which uses public constants α_i (coefficients of a Lagrange polynomial) in the reconstruction of the secret. To be specific, $x \leftarrow \texttt{Open}([x])$ is computed as $x = \sum_{i=1}^{|P|} \alpha_i \cdot x_i$, where x_i is the i-th party's share. Fan-in multiplication of shared secrets resulting in the sharing of their product $[c] \leftarrow \texttt{Product}([a], [b])$. Similarly inversion (in the considered finite field) of a shared value resulting in the sharing of the inverse value $[x^{-1}] \leftarrow \texttt{Inverse}([x])$.

2.2 Complexity Metrics

Typically, complexity is measured by the number of non-concurrent operations executed by the parties in charge of executing the functionality. We follow the relevant literature in the field, and differentiate between the operations that require any level of communication exchange in between the participants and those operations that can be executed locally. Indeed, following [10], we consider that the linear operations such as additions or scalar multiplications to be of negligible cost, given they do not require any message exchange and hence, are *free*. On the other hand, non-linear operations, such as multiplications, require

such exchanges and are substantially more expensive. Furthermore, we define round complexity as the number of sequential (non-parallelizable) invocations to functionality that requires at least one communication round i.e. the multiplicative depth of an arithmetic circuit. However, this is not the only metric that we use, the number of multiplications and equivalent operations, also referred to as the amount of work holds great importance as well, given that it would determine the volume of the information exchanged.

2.3 Secure Multiparty Computation

Secure multi-party computation lets us compute any functionality that can be composed into either an arithmetic or boolean circuit, among any number of mutually distrustful parties. Moreover, notable results known as BGW [19] and CCD [22] prove that any functionality can be calculated with perfect security, as long as a majority of players remained honest for the semi-honest case, and two thirds for the malicious case, thanks to the use of Verifiable Secret Sharing (VSS).

More recent results have focus on offering security against malicious adversaries in the presence of dishonest majorities e.g., [6,7,20], at the cost of an offline phase that provides cryptographic security. Furthermore, sub-protocols implementing functions can be securely composed thanks to their Universal Composability (UC) security [14]. Indeed, we make use of well known mechanisms designed to work on MPC settings, namely the following:

Secure Comparison. We specifically refer to equality and inequality tests for integer ring elements. Several protocols for secure comparison have been introduced by the literature, designed under the same model used by this paper achieving different security guarantees i.e. perfect security [8] or statistical security [23,24]. Such functionality can be further defined as follows:

$$[z] \leftarrow \texttt{Equal}([x],[y]) \quad ([x] \stackrel{?}{=} [y]) \quad \text{where} \quad [z] \in \{0,1\}, \tag{1}$$

$$[z] \leftarrow \texttt{Compare}([y],[y]) \quad ([x] \stackrel{?}{<} [y]) \quad \text{where} \quad [z] \in \{0,1\}. \tag{2}$$

Randomization. Protocols for efficient random number generation have been introduced by [25]. This work, commonly referred to as PRNG (Pseudo-Random Number Generation) introduced techniques to generate secret shared randomness at a negligible cost i.e., no communication cost associated to it. Furthermore all randomness can be easily pre-computed before the execution of any of our protocols. This includes the generation of the multiplication triples (see [6,7]) and any other process randomness needed across the underlying MPC protocols and our results.

Security. Security of MPC is typically defined following the Universal Composability (UC) framework [14,26], which is a general framework allowing arbitrary MPC protocols to be represented and analyzed. Informally speaking, in the UC

framework, for every real protocol an ideal functionality is first defined that may include a trusted third party which securely interacts with all involved players, performs computations on the players' private inputs and distributes the output to the parties. That is, for every real world protocol an ideal functionality $\mathcal{F}$ must be defined, which takes all the inputs from the players and performs the desired computation in an ideal way. Then the real protocol is said to be secure if whatever can be done in the real world by an adversary can also be simulated in the ideal world by an (ideal-world) simulator. Hence, UC security of a protocol ensures that the security provided by the ideal functionality is not stronger than that provided by the real protocol. This is done by introducing an environment $\mathcal{Z}$. The environment $\mathcal{Z}$ chooses all the inputs to every involved party, receives all outputs, and communicates freely with the adversary $\mathcal{A}$ throughout the entire protocol run. Basically, in the ideal case, adversary $\mathcal{A}$ is replaced by a simulator $\mathcal{S}$, which internally simulates $\mathcal{A}$ and acts as a buffer between $\mathcal{Z}$ and $\mathcal{F}$.

The ideal functionality for MPC is modeled by *arithmetic blackbox (ABB)* [18]. ABB allows the players (or users) to provide input/output values that are to be secret shared and that performs arithmetic operations on the values of the secret shares over a finite field, say $\mathbb{Z}_q$. It can be thought of as a generic procedure for secure computation. Any party (or parties) can send its (or their) private input to ABB and ask it to compute any computable function. The computation results are stored in the internal state of ABB so that they can be used in the subsequent computations. Stored values can only be made public if majority of the players agree on it. ABB provides us with abstraction of the details of MPC operations and of secret sharing. The ABB functionality is defined as follows.

Definition 1 (ABB Functionality $\mathcal{F}_{ABB}$). *The ideal functionality $\mathcal{F}_{ABB}$ for MPC, where $\nu \in \{q, q-1\}$, is defined as follows:*

- `Input`*: Receive a value $x \in \mathbb{Z}_\nu$ or x from some party and store x.*
- `Share`(x)*: Create a share $[x]$ of x.*
- `Product`$([x],[y])$*: Compute $z = x \cdot y$ and store $[z]$.*
- `Compare`$([x],[y])$*: Compare x and y, and return 0 if $x < y$ and 1 otherwise.*
- `Equal`$([x],[y])$*: Check if $x = y$; return 1 if $x = y$, 0 otherwise.*
- `sRand`$(\mathbb{Z}_\nu)$*: Sample $r \xleftarrow{R} \mathbb{Z}_\nu$ and store $[r]$.*
- `Open`$([x])$*: Send the value x to all players.*

Addition and scalar multiplication are denoted by their corresponding conventional symbols $+$ and $\cdot$.

Definition 2 (UC-security [14]). *A real protocol π is UC-secure if, for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ for which no environment $\mathcal{Z}$ can distinguish with a non-negligible probability if it is interacting with $\mathcal{A}$ and players running π or $\mathcal{S}$ and players using the ideal functionality $\mathcal{F}$.*

As we shall see later, our protocols for exponentiation offer perfect security against active and passive adversaries.

2.4 Exponentiation Based on Bit-Decomposition

Currently, the literature offers mechanisms to solve exponentiation, using bit-wise decomposition techniques. Initially, work introduced by Damgård et al. [8], showed how to achieve this with perfect security. However, as mentioned before, the cost related to the bit-decomposition of the inputs is relatively high, and has been seen as restrictive by other works in the field e.g. [2,24]. For instance, bit-decomposition protocols, such as the one described by [8], require bit-wise addition. A naive implementation of a carry for the addition would require at least as many sequential multiplications as the bit-wise input size. The costs related to the random bit generation have to be considered as well. In this section, we give a more detailed view and explanation on the Damgård et al. methods for exponentiation. The protocols follow the same notation as the rest of the paper. Furthermore, we assume the existence of the function $[x]_{bits} \leftarrow \texttt{bd}([x])$, which receives a secret shared input $[x]$ and return its shared bit-decomposition $[x]_{bits}$.

Exponentiation Protocol to a Public Value: This is achieved by executing a fan-in multiplication in $\mathbb{Z}_q$ as expressed by the following equation

$$[b^a] \leftarrow \texttt{Product}_{i=1}^{a}([b]). \tag{3}$$

This naive approach provides perfect security for both passive and malicious cases. However, the process has its own drawback in complexity terms, given that it directly depends on a i.e. $\mathcal{O}(a)$, in case no optimized implementation of the fan-in operation is used. Damgård et al. [8] introduced a more efficient procedure that can be executed in constant time. Protocol 1 shows its implementation.

Protocol 1. Secure Damgård et al. `exp([b],a)` operation to a public exponent

Input: secretly shared base $[b]$ in $\mathbb{Z}_{q^*}$, publicly available exponent a in $\mathbb{Z}_p$
Output: secret shared $[b^a]$

1 $[r], [r^a] \leftarrow \texttt{sRand}(\mathbb{Z}_q^*, a)$;
2 $[c] \leftarrow \texttt{Product}([b], [r])$;
3 $c \leftarrow \texttt{Open}([c])$;
4 $[b^a] \leftarrow (c^a) \cdot \texttt{Inverse}([r^a])$;

As its naive counterpart, it provides security against passive adversaries, but requires extra processing for the active case. Basically, during the generation of $[r]$ and $[r^a]$ an adversary could maliciously select the share corresponding to $[r^a]$. We invite the reader to revise [8] to explore the malicious case.

Exponentiation Protocol with a Public Base: To achieve this, the authors propose securely bit-decomposing the inputs (`bd`). In this case, the protocol

decomposes the secretly shared exponent into bits, and uses fan-in multiplications of a specially crafted term to achieve the desired behavior. On complexity, this process delivers the results in a constant round complexity for fixed input sizes, but it linearly grows with the input size. Protocol 2 shows its implementation.

Protocol 2. Secure `exp(b,[a])` operation on a public base

Input: publicly available base b in $\mathbb{Z}_q$,secretly shared exponent $[a]$ in $\mathbb{Z}_p$, size of the inputs in bits l

Output: secret shared $[b^a]$

1 $[a]_{bits} \leftarrow \texttt{bd}([a])$; // $([a_0],[a_1],...,[a_{l-1}])$ s.t. $a_i \in \{0,1\}$

2 $[b^a] \leftarrow \texttt{Product}_{i=0}^{l-1}([a_i] \cdot b^{2^i} + [1] - [a_i])$;

Privacy Preserving Exponentiation: This case considers that both, base and exponent are secretly held, can be achieved in a similar fashion. In this case, a secure `exp([b],a)` should be used instead of the plain text base exponentiation step, as shown by Protocol 3.

On Extending Exponentiation for $\mathbb{Z}_q$: Damgård et al. [8] presented an easy to implement technique to avoid revealing the secret $[b]$ when it is equal to 0, and hence extending this functionality to $\mathbb{Z}_q$ instead of $\mathbb{Z}_q^*$. This is achieved by simply adding the result of the following equality test $[b] \stackrel{?}{=} [0]$ to $[b]$ such that $b + ([b] \stackrel{?}{=} [0])$, and then subtracting the equality test at the end of the computation. This can be achieved in $log(l)$ rounds. This way the secret $[b]$ is not disclosed. The same can be applied to Protocols 1 and 3, to preserve the privacy of the inputs after being multiplicatively masked, hence we do not revisit this issue.

3 Secure Distributed Exponentiation

In this section, we explore different protocols for exponentiation based on publicly available and privately held inputs of any set of parties. Note that the protocols are designed to work for the case when the base b is secret shared in $\mathbb{Z}_q$ and its exponent a in $\mathbb{Z}^*{}_p$.

3.1 Public Base Exponentiation

Intuitively, this is the case where a publicly available base b is raised to a secretly shared exponent $[a]$. Its ideal functionality is defined as follows:

Definition 3. *Let base b and exponent a be elements of $\mathbb{Z}_q$ and $\mathbb{Z}*_p$, respectively, and let b be publicly available and a be privately preserved that is hosted by any subset of honest parties. The ideal functionality $\mathcal{F}_{\texttt{exp}(b,[a])}$ takes b and the secret $[a]_{p^*}$ as input and returns $[b^a]$ as output.*

Protocol 3. Secure `exp([b],[a])` operation

Input: secretly shared base $[b]$ in $\mathbb{Z}_{q^*}$,secretly shared exponent $[a]$ in $\mathbb{Z}_p$, size of the inputs in bits l
Output: secret shared $[b^a]$
1 $[a]_{bits} \leftarrow \texttt{bd}([a])$; // $([a_0],[a_1],...,[a_{l-1}])$ s.t. $a_i \in \{0,1\}$
2 $[b^a] \leftarrow \texttt{Product}_{i=0}^{l-1}(\texttt{Product}([a_i],\texttt{exp}([\texttt{b}],2^{\texttt{i}})) + [1] - [a_i])$;

Protocol 4 shows how this functionality can be achieved, for the semi-honest case. We further upgrade the construction in Protocol 4 to provide perfect security against active adversaries later in this section. Note that, without loss of generality, we assume the inputs have been secretly shared, using Shamir's scheme [21], where α_i are the publicly available interpolation coefficients as it was previously described.

Protocol 4. Secure `exp(b,[a])` operation with public base

Input: publicly available base b in $\mathbb{Z}_q$, secret shared exponent $[a]$ in $\mathbb{Z}_{p^*}$
Output: secret shared $[b^a]$
1 each party P_i locally computes $c_i \leftarrow b^{\alpha_i \cdot a_i}$;
2 $[c_i]_q \leftarrow \texttt{Share}(c_i)$;
3 $[b^a]_q \leftarrow \texttt{Product}_{i=1}^{|P|}([c_i]_q)$;

Protocol 4 returns $b^{[a]_{p^*}}$ by directly reconstructing the shares of $[a]_{p^*}$ on the exponent. This requires every party P_i to multiply locally its share a_i by its corresponding α_i constant. Let us analyze Shamir's scheme, for instance: parties should make use of the Lagrange interpolation multipliers (which are publicly held constants). Note that for the case of additive secret sharing, it suffices for the parties to apply the share of the exponent. Parties then proceed to calculated c_i locally, secret share it and multiply the resulting shares, finally obtaining $b_{p^*}^{[a]}$ as a result. This is a standard and common technique used for threshold decryption, for instance in voting systems e.g., [27], or more recently by some symmetric key techniques over MPC [11].

However, one issue becomes immediately obvious - the protocol, as described, cannot offer security for the malicious case. Protocol 4 description is secure against passive adversaries, that is because each party can choose its share as c_i. The protocol, nonetheless can be extended to provide malicious security at the cost of adding communication rounds.

Malicious Case. Let us first explore the basic naive approach to achieve malicious security: First, we calculate the result provided by the $\texttt{exp}(b,[a]_{p^*})$ functionality, and then we compute $\texttt{exp}(b,[a']_{p^*})$ for $[a']_{p^*}$ which is the product of

Protocol 5. Secure `exp(b,[a])` operation with public base against malicious adversaries

Input: publicly available base b in $\mathbb{Z}_q$, secret shared exponent $[a]$ in $\mathbb{Z}_{p^*}$
Output: secret shared $[b^a]$

1 $[b^a]_q \leftarrow \texttt{exp}(b, [a]_{p^*})$;
2 $[r]_{p^*} \leftarrow \texttt{sRand}(\mathbb{Z}_p^*)$;
3 $[a']_{p^*} \leftarrow \texttt{Product}([r]_{p^*}, [a]_{p^*})$;
4 $[b^{a'}]_{p^*} \leftarrow \texttt{exp}(b, [a']_{p^*})$;
5 $r \leftarrow \texttt{Open}([r]_{p^*})$;
6 $[r']_q \leftarrow \texttt{sRand}(\mathbb{Z}_q^*)$;
7 $[v]_q \leftarrow \texttt{Product}([r'], \texttt{exp}([b^a]_q, r) - [b^{a'}]_q) + [1]$; // `correct if` $[v] == 1$
8 $v_q \leftarrow \texttt{Open}([v]_q)$;

$[a]_{p^*}$ with some randomness $[r]_{p^*}$. We then just verify both calculated inputs are equal. We show how to achieve this in Protocol 5.

A similar process can be implemented by means of performing additional calls to any semi-honest $\texttt{exp}(b, [a]_{p^*})$ functionality, including ours. To achieve this we take a somewhat different approach as shown in Protocol 6, in the sense that we *sacrifice* randomness, and use b as the base of every call to $\texttt{exp}(b, [a]_{p^*})$, so that we can arithmetically operate over the exponents for verification.

Protocol 6. Secure $\texttt{exp}^+\texttt{(b,[a])}$ operation with public base against malicious adversaries

Input: publicly available base b in $\mathbb{Z}_q$, secret shared exponent $[a]$ in $\mathbb{Z}_{p^*}$
Output: secret shared $[b^a]$

1 $[b^a]_q \leftarrow \texttt{exp}(b, [a])$;
2 $[r]_{p^*} \leftarrow \texttt{sRand}(\mathbb{Z}_p^*)$;
3 $[a']_{p^*} \leftarrow \texttt{Product}([a]_{p^*}, [r]_{p^*} - [1])$;
4 $[b^{a'}]_q \leftarrow \texttt{exp}(b, [a']_{p^*})$;
5 $[w]_{p^*} \leftarrow \texttt{Product}([a]_{p^*}, [r]_{p^*})$;
6 $w \leftarrow \texttt{Open}([w]_{p^*})$;
7 $[r']_q \leftarrow \texttt{sRand}(\mathbb{Z}_q^*)$;
8 $[v]_q \leftarrow \texttt{Product}([r']_q, \texttt{Product}(\texttt{Product}([b^a]_q, [b^{a'}]_q), b^{-w}) - [1]) + [1]$; // `correct if` $[v] == 1$
9 $v \leftarrow \texttt{Open}([v]_q)$;

Given that all factors being multiplied are powers of the same base b, Protocol 6 then performs the following operation: $[a]_{p^*} + \texttt{Product}([a]_{p^*}, [r]_{p^*} - [1]) - [w]$. This in turn, translates to $[a]_{p^*} + \texttt{Product}([a]_{p^*}, [r]_{p^*} - [1]) - \texttt{Product}([a]_{p^*}, [r]_{p^*})$. Therefore we can correctly validate if the intermediate calls to the $\texttt{exp}(b, [a]_{p^*})$ functionalities are correct. Succinctly speaking $[v]$ would be 1 if, and only if integrity was maintained. Note that no other information is leaked by means

of applying $[r']_q$ to the output, this is also true for Protocol 5. Note that the protocol keeps the basic structure of its naive counterpart.

Security. Let $\pi_{\texttt{exp(b,[a])}}$ be the Protocol 6. Then the ideal functionality $\mathcal{F}_{\texttt{exp(b,[a])}}$ for the procedure `exp(b,[a])` exactly $\mathcal{F}_{ABB}$ extended with $\texttt{exp}(r, [e])$. This means that $\pi_{\texttt{exp(b,[a])}}$ uses only the MPC operations provided by $\mathcal{F}_{ABB}$. Therefore, it is straightforward to see that $\pi_{\texttt{exp(b,[a])}}$ is secure. Formally, we have the following.

Theorem 1. *The protocol $\pi_{exp(b,[a])}$ securely implements $\mathcal{F}_{exp(b,[a])}$ in the $\mathcal{F}_{ABB}$ framework.*

Proof. Since $\mathcal{F}_{\texttt{exp(b,[a])}}$ is the same as $\mathcal{F}_{ABB}$, the security of $\pi_{\texttt{exp(b,[a])}}$ inherits the security of the MPC operations in $\mathcal{F}_{\texttt{ABB}}$. □

3.2 Public Exponent Case

We now present our constant time secure exponentiation protocol for the case when the base b is privately held and the exponent a is public. The results on this section make use of the $\texttt{exp}(b, [a]_{p^*})$ functionality introduced by this work, however, any other mechanism that implement such functionality could be used instead. Furthermore, we assume $\texttt{exp}(b, [a]_{p^*})$ can be realized providing perfect security against semi-honest and malicious adversaries. This case ideal functionality can be defined as follows:

Definition 4. *Let base b and exponent a be elements of $\mathbb{Z}_q$ where $[b]$ is privately preserved, a is publicly available and hosted by any subset of honest parties. The ideal functionality $\mathcal{F}_{\texttt{exp}([b],a)}$ retrieves the secret b and $[a]$ and returns to the adversary $[b^a]$.*

We show how to implement our $\texttt{exp}([b], a)$ functionality in Protocol 7, note that it directly provides security against passive and active adversaries.

Protocol 7. Secure `exp([b],a)` operation with public exponent

Input: secret shared base $[b]$ in $\mathbb{Z}_q$, public available exponent a in $\mathbb{Z}_{p^*}$
Output: secret shared $[b^a]$

1 $g \leftarrow \texttt{getGenerator}(\mathbb{Z}_q)$; $[r']_{p^*} \leftarrow \texttt{sRand}(\mathbb{Z}_{p^*})$;
2 $[\bar{r}]_q \leftarrow \texttt{exp}(g, [r']_{p^*})$; // $g^{r'}$
3 $[c]_q \leftarrow \texttt{Product}([\bar{r}]_q, [b]_q)$;
4 $c \leftarrow \texttt{Open}([c]_q)$;
5 $c' \leftarrow c^a$; // $c' = g^{[r']\cdot a} \cdot [b]^a$
6 $[e]_{p^*} \leftarrow -a \cdot [r']_{p^*}$;
7 $[b^a]_q \leftarrow c' \cdot \texttt{exp}(g, [e]_{p^*})$; // $c' \cdot r^e$

The protocol itself works as follows: Parties agree on a unique generator g and some secret shared randomness $[\bar{r}]_q$. The protocol, then makes use of $[\bar{r}]_q$ to

mask the base $[b]_q$ by calculating $[c]_q = \texttt{Product}([\bar{r}]_q, [b]_q)$. Note that $[c]_q$ is then immediately made public. The protocol proceeds to compute $[c^a]_q$ by means of calling `exp(b,[a]_p*)` functionality. Finally, the protocol is able to construct the expression $[r'] \cdot a - [r'] \cdot a$ as the exponent of an $[r]$ whilst multiplying $[b^a]$, which is equivalent to $\texttt{Product}([r^0], [b^a]) = [b^a]$.

Security. Let $\pi_{\texttt{exp([b],a)}}$ be the protocol described in Protocol 7. Then the ideal functionality $\mathcal{F}_{\texttt{exp([b],a)}}$ for the procedure `exp([b],a)` is $\mathcal{F}_{ABB}$ extended with $\texttt{exp}(b, [a])$, which is described by Protocol 6. But $\mathcal{F}_{\texttt{exp}}(b, [a])$ is the same as $\mathcal{F}_{ABB}$, hence $\mathcal{F}_{\texttt{exp([b],a)}}$ is also the same as $\mathcal{F}_{\texttt{ABB}}$. Therefore, the security of this protocol is also straightforward.

Theorem 2. *The protocol $\pi_{exp([b],a)}$ securely implements $\mathcal{F}_{exp([b],a)}$ in the $\mathcal{F}_{ABB}$ framework.*

Proof. The same as the proof of Theorem 1. □

3.3 Privacy Preserving Exponentiation

We now introduce our constant time protocol for secure exponentiation given a privately held base b and exponent a. Its ideal functionality can be expressed as follows:

Definition 5. *Let base $b \in \mathbb{Z}_q$ and exponent $a \in \mathbb{Z}_{p^*}$ be privately preserved and hosted by any subset of honest parties. The ideal functionality $\mathcal{F}_{\texttt{exp}([b],[a])}$ takes $[b]_q$ and $[a]_{p^*}$ as input and returns $[b^a]$ as output.*

Our construction assumes that `exp(b,[a])` functionality is available. Our construction, showcased by Protocol 8, offers perfect security against both, passive and active adversaries.

Protocol 8. Secure `exp([b],[a])` operation with shared exponent and base

Input: secret shared base $[b]$ in $\mathbb{Z}_q$, and exponent $[a]$ in $\mathbb{Z}_{p^*}$
Output: secret shared $[b^a]$

1 $g \leftarrow \texttt{getGenerator}(\mathbb{Z}_q)$; $[r']_{p^*} \leftarrow \texttt{sRand}(\mathbb{Z}_{p^*})$;
2 $[\bar{r}]_q \leftarrow \texttt{exp}(g, [r']_{p^*})$ //$g^{[r']}$
3 $[c]_q \leftarrow \texttt{Product}([\bar{r}]_q, [b]_q)$;
4 $c \leftarrow \texttt{Open}([c]_q)$;
5 $[c']_q \leftarrow \texttt{exp}(c, [a]_{p^*})$; // $g^{[r'] \cdot [a]} \cdot [b]^{[a]}$
6 $[e]_{p^*} \leftarrow -1 \cdot \texttt{Product}([r']_{p^*}, [a]_{p^*})$;
7 $[b^a]_q \leftarrow \texttt{Product}([c']_q, \texttt{exp}(g, [e]_{p^*}))$;

In this case, we make use of many of the mechanisms introduced by Protocol 7 with some basic dissimilarities. Basically, to obtain $[c^a]_{p^*}$, the protocol makes

use of the secure $\texttt{exp}(b, [a])$ functionality. This way the protocol can obtain $[c']_q$, instead of c'. Despite this, protocol behaves just in the same way as described for Protocol 7. Note that also plain and scalar arithmetic operations are replaced by their equivalent privately preserving counterparts.

Security. Let $\pi_{\texttt{exp([b],[a])}}$ be the protocol described in Protocol 8. Then the ideal functionality $\mathcal{F}_{\texttt{exp([b],[a])}}$ for the procedure `exp([b],[a])` is again $\mathcal{F}_{ABB}$ extended with $\texttt{exp}(b, [a])$, the ideal functionality of which is nothing but $\mathcal{F}_{ABB}$. Hence, the security of $\mathcal{F}_{\texttt{exp([b],[a])}}$ is also straightforward.

Theorem 3. *The protocol $\pi_{\texttt{exp([b],[a])}}$ securely implements $\mathcal{F}_{\texttt{exp([b],[a])}}$ in the $\mathcal{F}_{ABB}$ framework.*

Proof. The same as the proof of Theorem 1. □

Remark on Security. As we can see, the security of our protocols follow directly from the actual security of the MPC operations in $\mathcal{F}_{ABB}$ achieved in practice. As we have mentioned in the previous section, seminal results such as BGW [19] and CCD [22] showed that any functionality can be achieved with perfect security, as long as a majority of the players are honest for the semi-honest case, and two thirds for the malicious case. Security against malicious adversaries in the presence of dishonest majorities can also be achieved at the cost of an offline computation phase [6,7,20]. Furthermore, sub-protocols (i.e., the MPC operations in $\mathcal{F}_{ABB}$) are UC secure, hence they can be securely composed.

3.4 Complexity

All of our protocols have constant round and multiplicative complexity with respect of the size of the input. Note that in all our protocols the amount of work and their multiplicative depth grows linearly with respect to the number of parties n. Given that the focus of industry and academia have been centered on developing and optimizing protocols for the 2-Party and 3-Party case e.g. [12,13,28,29], we have introduced in our complexity analysis both scenarios. It is worth notice that other protocols designed for n-parties such as [6,30] have been mainly used on either the 2 or 3 party scenario e.g. [31]. Table 2 showcases how such protocol complexities vary on these different scenarios. Note that round complexity (r.) or multiplicative depth is the same as the multiplicative complexity (amount of work) for all our protocols.

Although the multiplicative depth of the protocol is altered by the number of parties, its effect is constrained to the linear behavior we mentioned above. For the malicious case the asymptotic complexity of all protocols is the same i.e., $\mathcal{O}(log(n))$, albeit the function constants would be larger (protocols would require to use the $\texttt{exp}(b, [a])$ variant, that is secure against active adversaries, instead of its passive counterpart, which requires a proportionally larger amount of work).

Table 2. Protocol complexity for exponentiation protocols

Protocol	Semi-honest		
	2-P	3-P	n-P
$\texttt{exp}([b], a)$	$\mathcal{O}(1) \rightarrow 6$ r.	$\mathcal{O}(1) \rightarrow 8$ r.	$\mathcal{O}(log(n)) \rightarrow (4 + 2 \cdot \lceil log(n) \rceil)$
$\texttt{exp}(b, [a])$	$\mathcal{O}(1) \rightarrow 2$ r.	$\mathcal{O}(1) \rightarrow 3$ r.	$\mathcal{O}(log(n)) \rightarrow (1 + \lceil log(n) \rceil)$
$\texttt{exp}^{+}(b, [a])$	$\mathcal{O}(1) \rightarrow 2$ r.	$\mathcal{O}(1) \rightarrow 3$ r.	$\mathcal{O}(log(n)) \rightarrow (1 + \lceil log(n) \rceil)$
$\texttt{exp}([b], [a])$	$\mathcal{O}(1) \rightarrow 10$ r.	$\mathcal{O}(1) \rightarrow 13$ r.	$\mathcal{O}(log(n)) \rightarrow (7 + 3 \cdot \lceil log(n) \rceil)$

3.5 Performance

To estimate our protocols performance, we consider that the cost of an atomic non-linear (or equivalent) operation is negligible (additions and scalar multiplications), given they do not depend on communication rounds. Given that our multiplicative complexity is equivalent to our round complexity for all our protocols, it follows that: we can easily project the computational time that our protocols need by measuring one multiplication. For instance, we have used the Ben-Or, Goldwasser and Wigderson [19] (BGW) protocol implementation described by [32] to test its multiplicative performance. The implementation is a `C++` self-contained library based on Number Theory Library (NTL) [33] designed for 3 parties in a semi-honest setting. For our estimations, we have averaged two million of multiplications (using Gennaro's protocol [34]) on a 64-bit server with $2 * 2 *$ 10-cores Intel Xeon E5-2687 at 3.1 GHz where only the minimum 2 core pero process needed where used. The results yield a $2.08 \cdot 10^{-5}$ s time per each communication round consisting of a single multiplication. We can then extrapolate the computational time of our protocols as follows: $\texttt{exp}(a, [b])$ is $6.24 \cdot 10^{-5}$ s; $\texttt{exp}([a], b)$ is $1.665 \cdot 10^{-4}$ s and; $\texttt{exp}([a], [b])$ is $2.7 * 10^{-4}$ s. Similar estimations could be performed for other settings using any other suitable MPC protocol e.g., SPDZ [6,7].

4 Public Key Decryption

We now show how to use our primitives to build a particular secure distributed decryption scheme that can be utilized for threshold decryption. Under this scenario a private key sk is shared among n parties, and a publicly available ciphertext c (of a message m) has to be decrypted, such that the output of the decryption is a share under any linear secret sharing scheme used for MPC. In other words from c and $[\mathsf{sk}]$, we have to obtain $[m]$. This way we can use public key decryption as a subprotocol for any other MPC functionality. We explore such scenario for two basic and well known public key schemes, namely RSA [15], ElGamal [16]. We start by giving an overview about these protocols. Note that all schemes described in this section are IND-CPA-secure. In all the protocols below, λ denotes the security parameter.

We stress that such distributed decryption can be of interest in applications, such as privacy-preserving biometric authentication. For instance, one can encrypt users' biometric templates and share the decryption key among different parties. Then, when a user wants to authenticate, the user provides an encrypted fresh biometric template, which is then compared with the stored template in a distributed fashion. At the end, the result indicating whether there is a match is jointly decrypted by the parties following the procedures that we present in this section. We plan to demonstrate this in the future.

4.1 RSA

The RSA encryption is based on the hardness of prime factorization of integers.

- KeyGen: The key generation algorithm takes a security parameter λ as input and outputs a public key $\mathsf{pk} = (n, e)$ and a private key $\mathsf{sk} = (p, q, d)$, i.e., $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$. Here and throughout forward, for the purposes of this section p and q are large distinct primes, $n = pq$, e is such that $\gcd(e, \phi(n)) = 1$, and d satisfies $de \equiv 1 \mod \phi(n)$, where $\phi()$ is the Euler's totient function.
- Enc: The encryption algorithm takes the public key $\mathsf{pk} = (n, e)$ and a message which is converted into a number $m < n$ as input and outputs a ciphertext $c \equiv m^e \mod n$; i.e., $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$.
- Dec: The decryption algorithm takes $\mathsf{sk} = (p, q, d)$ and a ciphertext c as input and outputs a message $m \equiv c^d \mod n$; i.e., $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$.

4.2 ElGamal

The ElGamal encryption is based on the hardness of discrete logs.

- KeyGen: The key generation algorithm takes a security parameter λ as input and outputs a public key $\mathsf{pk} = (G, g, q, h)$ and a private key $\mathsf{sk} = x$, i.e., $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$. This is done first by choosing a large prime p and a generator for the multiplicative group $G = \mathbf{Z}_p^*$, whose order is $q = p - 1$, and then by choosing a random $x \in G$ and computing $h \equiv g^x \mod p$.
- Enc: The encryption algorithm takes the public key $\mathsf{pk} = (G, g, p, h)$ and a message which is converted into an element $m \in G$ as input and outputs a ciphertext $c = (c_1, c_2)$, i.e., $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, by computing $c_1 \equiv g^y \mod p$ and $c_2 \equiv h^y m$, for a randomly chosen $y \in G$.
- Dec: The decryption algorithm takes $\mathsf{sk} = x$ and a ciphertext $c = (c_1, c_2)$ as input and outputs a message $m \equiv c_1^{-x} c_2 \mod p$; i.e., $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$.

4.3 Privacy Preserving Decryption

The data oblivious implementation of the decryption protocols of this public key schemes can be easily achieved by reusing both the exponentiation functionality described in Sect. 3 and the modulo operation showcased above. In our case, We assume that the ciphertext is publicly available, whereas the private key is secret

shared. The objective for us is to have a secret shared version of the message without leaking its content to any party involved in the computation.

Note that protocols to perform secure modulo operations for MPC, have indeed been extensively studied by the literature. We can name for instance the construction from the seminal paper by Damgård et al. [8]. In recent years, more efficient results although sometimes with more limited security capabilities e.g. statistical instead of perfect security, have been introduced as well e.g. [9].

RSA. For this case, given that the decryption protocol $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$, where $c \equiv m^e$, the computation of $[m]$ can be summarized by the computation of: $m = c^d = (m^e)^d \mod N$. Thus, with the functionality provided by this work it suffices for the parties computing the decryption to do $[m] \leftarrow \mathtt{exp}(c, [d]) \mod N$.

ElGamal. The decryption protocol in this case consists on some basic operations on what we assume to be publicly available c_1 and c_2 values, as follows: $m = \frac{c_2}{c_1^x} \mod p$. To implement such protocol on MPC, where the sk $[x]$ is secretly shared we make use of the following: a scalar multiplication, our exponentiation method, the multiplicative inverse of such result, which can be computed in one round, and a secure mod operation: $[m] = c_2 \cdot \mathtt{Inverse}(\mathtt{exp}(c_1, [x])) \mod p$.

5 Conclusions

In this paper, we introduce secure mechanisms to perform exponentiation over MPC for arithmetic circuits without bit decomposition. Our protocols are simple and easy to follow mechanisms that have constant round/multiplicative complexity and offer security against semi-honest and malicious adversaries. Our protocols, besides being lean and simplified, improves the current state of the art for such kind of mechanisms. Additionally, we included a possible application for our techniques, in the form of public key decryption. Further work should explore the viability of these results on other related MPC fundamental applications such as comparisons. Another direction for future work would be to validate our protocols in practical applications such as in biometric settings, where templates need to be securely transmitted to some MPC based processing server, using PKI infrastructure.

Acknowledgements. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through H2020-ICT-2014-644371 WITDOM and the Flemish Government through the imec Distributed Trust program and through ICON Diskman. The authors would like to thank Prof. Nigel Smart and the anonymous reviewers for their comments and inputs.

References

1. Ning, C., Xu, Q.: Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 483–500. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_28
2. Ning, C., Xu, Q.: Constant-rounds, linear multi-party computation for exponentiation and modulo reduction with perfect security. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 572–589. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_31
3. Bogetoft, P., et al.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03549-4_20
4. Bogdanov, D., Kamm, L., Laur, S., Sokk, V.: Rmind: a tool for cryptographically secure statistical analysis. IEEE Trans. Dependable Secure Comput. **15**(3), 481–495 (2018)
5. Aly, A., Van Vyve, M.: Practically efficient secure single-commodity multi-market auctions. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 110–129. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_7
6. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
7. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Proceedings of ACM SIGSAC, CCS 2016, pp. 830–842. ACM (2016)
8. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_15
9. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_13
10. Cramer, R., Damgård, I., Nielsen, J.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press, Cambridge (2015)
11. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 430–443. ACM, New York (2016)
12. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: a framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88313-5_13
13. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the ACM SIGSAC, pp. 805–817 (2016)
14. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)

15. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
16. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_2
17. Szepieniec, A., Preneel, B.: New techniques for electronic voting, p. 30, Report 2015/809 (2015)
18. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_15
19. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, pp. 1–10. ACM (1988)
20. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_11
21. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
22. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC, pp. 11–19. ACM (1988)
23. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 134–150. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_9
24. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 645–656. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_56
25. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_19
26. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS 2001, pp. 136–145 (2001)
27. Peeters, R., Nikova, S., Preneel, B.: Practical RSA threshold decryption for things that think. In: 3rd Benelux Workshop on Information and System Security (2008)
28. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM (1987)
29. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security Symposium (2011)
30. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1
31. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. Cryptology ePrint Archive, Report 2015/1006 (2015). http://eprint.iacr.org/2015/1006
32. Aly, A.: Network flow problems with secure multiparty computation. Ph.D. thesis, Universté catholique de Louvain, IMMAQ (2015)

33. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_23
34. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC. ACM (1998)

A Fourier Analysis Based Attack Against Physically Unclonable Functions

Fatemeh Ganji(✉), Shahin Tajik, and Jean-Pierre Seifert

Security in Telecommunications, Technische Universität Berlin, Berlin, Germany
{fganji,stajik,jpseifert}@sec.t-labs.tu-berlin.de

Abstract. Electronic payment systems have leveraged the advantages offered by the RFID technology, whose security is promised to be improved by applying the notion of Physically Unclonable Functions (PUFs). Along with the evolution of PUFs, numerous successful attacks against PUFs have been proposed in the literature. Among these are machine learning (ML) attacks, ranging from heuristic approaches to provable algorithms, that have attracted great attention. Our paper pursues this line of research by introducing a Fourier analysis based attack against PUFs. More specifically, this paper focuses on two main aspects of ML attacks, namely being provable and noise tolerant. In this regard, we prove that our attack is naturally integrated into a provable Probably Approximately Correct (PAC) model. Moreover, we show that our attacks against known PUF families are effective and applicable even in the presence of noise. Our proof relies heavily on the intrinsic properties of these PUF families, namely arbiter, Ring Oscillator (RO), and Bistable Ring (BR) PUF families. We believe that our new style of ML algorithms, which take advantage of the Fourier analysis principle, can offer better measures of PUF security.

Keywords: Physically Unclonable Functions · Boolean analysis
Noise sensitivity · Low degree algorithm · Machine learning
PAC Learning

1 Introduction

Payment systems, including electronic payment and ticketing systems, provide one of the prominent examples of the diversified applications of RFID-tags. As an effective and cost-efficient security mechanism for these tags, PUFs have been introduced in the literature cf. [10,48,49]. The design of PUFs relies on inherent manufacturing process variations, being uncontrollable, but exploitable by a circuitry to generate either a source of randomness or an instance-specific fingerprint [18]. The growing need for using PUFs in several applications stems from two main issues. On the one hand, the ineffectiveness of traditional security measures, e.g., secure key generation/storage, has been widely accepted. On the other hand, the inevitable fact that overbuilt and counterfeit hardware primitives

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 310–328, 2018.
https://doi.org/10.1007/978-3-662-58387-6_17

can be used in various important applications further contribute to this need for robust security measures [26]. Since the notion of PUFs has been introduced to address the aforementioned issues, several studies have focused on the advantages and disadvantages of this concept. Designing such circuits and their respective security assessments, more particularly, cryptanalysis of PUFs are within two ends of the wide spectrum of these studies.

In addition to invasive and semi-invasive attacks, e.g., [20,37,45–47], a broad range of cryptanalysis of PUFs is covered by non-invasive attacks, for instance [42]. A great variety of these frameworks and numerous models have been developed around the principles of linear algebra [11], stochastic optimization [5], and machine learning (ML) [12,14–16,23,42]. When launching the latter attacks, the adversary observes only a small subset of challenges and their corresponding responses (i.e., the inputs and the outputs of the PUF) in order to build a model of the challenge-response behavior of the PUF. Therefore, when compared with invasive and semi-invasive attacks, these attacks are cost-effective and nondestructive, and consequently, attractive for adversaries.

Applying empirical ML algorithms (e.g., [42]) in the assessment of the security of PUFs marked the beginning of an era, after which the well-established concepts and existing algorithms in the field of ML were applied to analyze the security of these primitives. Beyond the early heuristic methods, probably approximately correct (PAC) learning frameworks have been developed to *prove* vulnerabilities for the known families of intrinsic PUFs to ML attacks [12,14–16,19]. The results of these studies have been acknowledged, and form now a solid basis for the design of PUFs, c.f. [51]. Albeit being useful for this purpose, the question remains open whether practical aspects of the design of PUFs have been adequately reflected by the PAC learning frameworks. More specifically, except the proof provided for XOR-arbiter PUFs [15], PAC learning in the presence of noise has not been discussed in the literature so far.

This issue is of twofold importance. First, the term "noise" in the PUF-related literature refers to the observation that applying the same challenge may result in obtaining different responses due to the environmental changes, see, e.g. [30]. These noisy responses reveal some information about the challenge-response behavior of the PUF, in a similar way to side channel information, which can be beneficial to model the PUF [5,8,9]. Understanding the mechanisms of generating noisy responses is therefore essential for designing a PUF that is robust against such hybrid attacks. Secondly, the gap between the existing noise models in the ML- and PUF-related literature should be bridged primarily by a thorough understanding of differences and similarities between these models. Accordingly, a refined model of noisy PUFs should be established, which provides a firm basis for analyzing the security of these primitives against ML attacks. This paper aims to address these issues by providing the following contributions.

Establishing a Refined Model of Noisy PUFs that is in Line with Models Widely Accepted in ML Theory. In our model, we take into consideration the impact of noise on the final response of a PUF and as well at the

inter-stage behavior of a PUF. We demonstrate that this model agrees with the noise models in ML theory, namely, attribute and classification noise.

Introducing a New ML Attack Relying on the Principles of Fourier Analysis. Thanks to the representation of PUFs as Boolean functions, we explore the properties of PUFs from the Fourier analysis perspective. We introduce the notion of noise sensitivity of Boolean functions representing PUFs as a powerful analysis tool. Moreover, for known and widely-used PUFs a so-called low degree algorithm approximating the Fourier coefficients of the corresponding Boolean functions is presented in this paper.

Provability of our ML Attack, Even in the Presence of Attribute and Classification Noise. Eventually, we prove that for known families of PUFs our attack can be launched to learn their challenge-response behavior, with prescribed levels of accuracy and confidence, even if the challenge-response pairs are noisy.

2 Notation and Preliminaries

2.1 PUFs

First, we stress that our paper does not cover the topics of formalization and formal definitions of the PUFs. For more details on these topics see, e.g., [3,4]. Note that hereafter the term "PUF" refers to the most popular, and known families of standalone PUFs: arbiter PUFs, Ring Oscillator (RO) PUFs, and Bistable Ring (BR) PUFs. Here, a standalone PUF means a PUF that is not composed of a combination of some PUFs (e.g., XOR arbiter PUFs) or other means. Generally speaking, PUFs are physical mappings from the inputs to the outputs, i.e., from the given *challenges* to the respective *responses*. These mappings are characterized by physical properties of the platform, on which the PUF is implemented. From among several security properties of PUFs, here we consider solely unclonability. Let the mapping $f_{\mathrm{PUF}} : \mathcal{C} \to \mathcal{Y}$, where $f_{\mathrm{PUF}}(c) = y$, describes a PUF. Ideally, for a given PUF f_{PUF} unclonability reflects the fact that creating a clone, i.e., a (physical) mapping $g_{\mathrm{PUF}} \neq f_{\mathrm{PUF}}$, is virtually impossible, where the challenge-response behavior of g_{PUF} is *similar* to f_{PUF} [3].

2.2 Boolean Functions as Representations of PUFs

Similar to the most relevant studies on PUFs, we adopt the general definition of PUFs that is the physical mappings (see Sect. 2.1). This enables us to represent PUFs as Boolean functions over the finite field $\mathbb{F}_2$. To this end, consider $V_n = \{c_1, c_2, \ldots, c_n\}$ that is the set of Boolean attributes or variables, being either ***true*** or ***false*** denoted by "1" and "0", respectively. Moreover, let $C_n = \{0,1\}^n$ be the set of all binary strings with n bits, and an *assignment* be a mapping from V_n to $\{0,1\}$. Therefore, an assignment can be thought of as an n-bits string, where the i^{th} bit associated with the value of c_i (i.e., "0" or "1").

A Boolean formula is a mapping that assigns values from the set $\{0,1\}$ to an assignment. In this regard, each Boolean attribute is also a formula, i.e., c_i is a possible formula. If the Boolean formula assigns "1" to a Boolean assignment, it is a *positive example* of the concept, otherwise a *negative example*. Furthermore, a Boolean function $f : C_n \to \{0,1\}$ defines a Boolean formula accordingly.

In general, Boolean functions can be represented by several different classes of functions, e.g., juntas, Linear Threshold functions (LTFs), and Decision Lists (DLs), cf. [38,41]. A k-junta is a Boolean function, whose output is determined solely by an unknown set of k variables. A list L containing r pairs $(f_1, v_1), \ldots, (f_r, v_r)$ is called a DL, where the Boolean formula f_i is a conjunction of Boolean attributes, and $v_i \in \{0,1\}$ with $1 \leq i \leq r-1$. For $i = r$, we have $v_r = 1$. When representing a Boolean function by a decision list, $L(c) = v_j$, where $c \in C_n$ and j is the smallest index in L so that $f_j(c) = 1$. Let k-DL denote the set of DLs, where each f_i is a conjunction of at most k Boolean attributes.

Before defining linear threshold functions, we define the encoding scheme $\chi(0_{\mathbb{F}_2}) := +1$, and $\chi(1_{\mathbb{F}_2}) := -1$. Hence, the Boolean function f can be defined as $f : \{-1,+1\}^n \to \{-1,+1\}$. Such a function is called a linear threshold function, if there are coefficients $\omega_1, \omega_2, \cdots, \omega_n \in \mathbb{R}$ and $\theta \in \mathbb{R}$ such that $f(c) = \operatorname{sgn}\left(\left(\sum_{i=1}^{n} \omega_i c_i\right) - \theta\right)$. Without loss of generality, we assume that $\sum_{i=1}^{n} \omega_i c_i \neq \theta$ for every $c \in C_n$.

Noise Sensitivity of Boolean Functions. This term should not be mistaken as the notion of noise discussed in the PUF-related literature. The noise sensitivity of the Boolean function $f : \{-1,+1\}^n \to \{-1,+1\}$ can be defined as follows (see Sect. 2.2 for more details on the encoding scheme required to define the noise sensitivity). Let c be a string chosen randomly and uniformly. By flipping each bit of this string independently with probability ε $(0 \leq \varepsilon \leq 1)$ we obtain the string c'. The noise sensitivity of f at ε is

$$NS_\varepsilon(f) := \Pr[f(c) \neq f(c')].$$

When studying the noise sensitivity of Boolean functions, applying methodologies developed for the spectral analysis of Boolean functions can provide a better understanding of this notion. The Fourier expansion of a Boolean function can be written as

$$f(c) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(c),$$

where $[n] := \{1, \ldots, n\}$, $\chi_S(c) := \prod_{i \in S} c_i$, and $\hat{f}(S) := \mathbf{E}_{c \in \mathcal{U}}[f(c)\chi_S(c)]$. Note that $\mathbf{E}_{c \in \mathcal{U}}[\cdot]$ indicates the expectation over examples chosen uniformly.

2.3 PAC Learning Model [24]

Consider a PAC learner that is a learning algorithm, which is given access to a set of *examples* to generate an approximately correct hypothesis, with high probability. More formally, suppose that $F = \cup_{n \geq 1} F_n$ denotes a target concept

class, i.e., a set of Boolean functions over the instance space $C_n = \{0,1\}^n$. In this paper, we are interested in a useful extension of the PAC model, in which each example is drawn from the instance space C_n with regard to the uniform distribution U. The hypothesis $h \in F_n$ that is a Boolean function over C_n is an ε-approximator for $f \in F_n$, if

$$\Pr_{c \in_U C_n}[f(c) = h(c)] \geq 1 - \varepsilon.$$

The complexity of a target concept $f \in F$ is assessed by measuring the size of that under a target representation. In order to define the size of a target concept $f \in F$, $size(f)$, we define the mapping $size : \{0,1\}^n \rightarrow \mathbb{N}$, relating a natural number $size(f)$ with a target concept $f \in F$. A polynomial-time algorithm $\mathtt{A}$, i.e., our learner, is provided with labeled examples $(c, f(c))$, where $f \in F_n$, and c is chosen uniformly at random from C_n. Here we concentrate on the strong uniform PAC learning algorithms, defined as follows.

Definition 1. *A strong uniform PAC learning algorithm* $\mathtt{A}$ *for the target concept class* F *is given a polynomial number of labeled examples to generate an* ε*-approximator for* f *under the uniform distribution* U*, with probability at least* $1 - \delta$*. In this regard, for any* $n \geq 1$*, any* $0 < \varepsilon, \delta < 1$*, and any* $f \in F_n$*, the running time of the algorithm* $\mathtt{A}$ *is* $poly(n, 1/\varepsilon, size(f), 1/\delta)$*, where* $poly(\cdot)$ *denotes a polynomial function.*

3 Noise: Its Origin and Models

In this section we aim to come up with a model for PUFs enabling us to understand, how the noise can affect the functionality of a PUF. As mentioned in Sect. 1, in the PUF-related literature the response to a challenge is noisy, if repeated evaluations of the PUF with the respective challenge results in different responses. This is due to environmental variations and their impact on the functionality of physical components forming the PUF, e.g., the stages in an arbiter PUF. Environmental variations cover a wide range of uncontrollable random noise, e.g., thermal noise, uncertainties in measurement, cross talk, and power supply noise [3,50]. According to the lessons from performance specifications of circuits, these random variations can be conventionally modeled as random variables following Gaussian distributions [2,36,44].

3.1 Impact of the Noise on a Single Stage

To provide a better understanding of the impact of the environmental variations on the internal functionality of a PUF, we focus on a single constitutive physical component of the PUF, hereafter called a stage. As carefully formulated in [29], although the consideration of low-level physical details is a tedious task, (measurable) physical processes can be approximated by "hidden variables", namely the process variable and the noise variable. The latter variable corresponds to

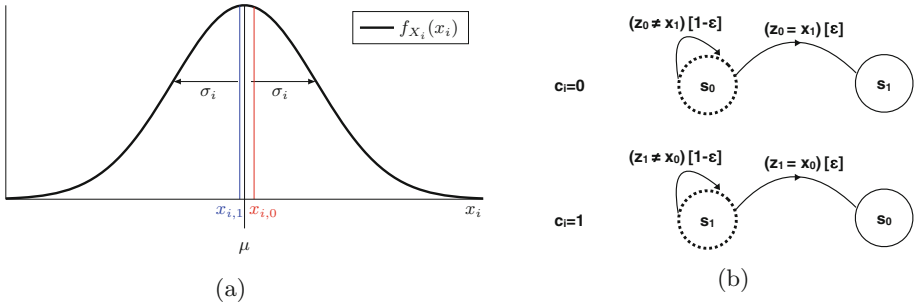


Fig. 1. (a) The Gaussian random variable X_i that corresponds to the i^{th} physical component of the PUF, and its two realizations $x_{i,1}$ and $x_{i,0}$. In a meta-stable state these two realizations will be very close together. (b) Our simple Probabilistic (labeled) Transition Systems (PTS) describing how the noise can affect each stage in a PUF. The expressions included in parentheses denote the *labels*, whereas the information given in brackets refers to the probability of the transition between the states.

the effect of random noise on a single stage of the PUF. This variable follows a Gaussian distribution N_i, with realization n_i for each evaluation of the PUF. The definition of process variable determines the effect of manufacturing process variations on a single stage of the PUF [29]. As discussed before, this variable dented by X_i follows a Gaussian distribution. Similarly and independently, $X_1, \cdots, X_n$ can be defined, where n denotes the number of stages. In this manner, the mean value of the respective distribution (μ) is reported by manufactures as the nominal value, and the standard deviation σ_i is the result of the process variations, cf. [8,16,34]. Two realizations of the random variable X_i, namely $x_{i,1}$ and $x_{i,0}$, are generated during manufacturing. Without loss of generality, suppose that the following holds. When $c_i = 1$ is applied to the i^{th} stage, the realization $x_{i,1}$ is chosen to be involved in generating the final response of the PUF, whereas $x_{i,0}$ corresponds to $c_i = 0$. Moreover, suppose that the order relation between these realizations is $x_{i,1} > x_{i,0}$. Now, the *total impact of hidden variables on a stage* can be formulated as $Z_i = X_i + N_i$ $(1 \leq i \leq n)$, where Z is clearly a Gaussian random variable. In addition, the realizations of this random variable are $z_{i,1} = x_{i,1} + n_{i,1}$ and $z_{i,0} = x_{i,0} + n_{i,0}$, relating to the challenge bit applied to the PUF. Since the realizations $z_{i,1}$ and $z_{i,0}$ are related to two different evaluations of the PUF (with $c_i = 1$ and $c_i = 0$, respectively), the noise realizations vary, as indicated by different indices. As defined in [29], the final response of the PUF is determined by these realizations. Obviously, the difference between $z_{i,1}$ and $z_{i,0}$ is the main factor contributing to the final response of the PUF. Now consider the i^{th} stage that is a meta-stable state, see Fig. 1(a). Here by meta-stable condition we refer to the fact that the realizations of the random variable X_i, i.e., $x_{i,1}$ and $x_{i,0}$ can be very close together so that under the effect of environmental noise one realization can be equal to another: $z_{i,0} = x_{i,1}$ or $z_{i,1} = x_{i,0}$, depending on the value of the challenge bit c_i [7]. To explain this, a simple Probabilistic (labeled) Transition Systems (PTS) can be defined as follows [40].

- There are two processes (i.e., sequences of events) corresponding to the value of the challenge bit applied to the i^{th} stage: $c_i = 1$ and $c_i = 0$, see Fig. 1(b).
- In both processes, the set of states S contains two states denoted by s_0 and s_1. The state s_0 represented the case that the challenge bit $c_i = 0$ is applied and in an ordinary condition (i.e., not meta-stable) we expect that $x_{i,0}$ would be involved in generating the final response of the PUF. Similarly, the state s_1 can be defined.
- $s_{int} \in S$ is the initial state in each process, shown by dashed circles in each process. And the set of action labels is L.
- A transition probability function $T : \quad S \times L \times S \rightarrow [0,1]$ represents, under which circumstances and what degree of probability the system transits from one state to another. Clearly, $\sum_{(l_i,s_j)\in L\times S} T(s_{int}, l_i, s_j) = 1$.

Precisely defining our PTS, the tuple $\mathcal{A}_i$ represents the process related to the case, when the challenge bit c_i is applied: $\mathcal{A}_i = (S, L, T)$.

In each of the processes, as illustrated in Fig. 1(b), the PTS may transit from one state to another with probability ε, otherwise it remains in its initial state. For instance, applying the challenge bit $c_i = 0$, the initial state s_0 indicates that $x_{i,0}$ would contribute to the final response of the PUF. However, if this stage (the i^{th} stage) is in a meta-stable state, i.e., $z_{i,0} = x_{i,1}$, it is not possible to *differentiate* whether $x_{i,1}$ or $z_{i,0}$ would be involved to generate the final response of the PUF[1]. In other words, it can be thought that $c_i = 1$ is applied and the final response is under the influence of $x_{i,1}$, i.e., the PTS is in s_1 state. More precisely, we define a discrete random variable A corresponding to the event of a transition between s_0 and s_1. Formally, let $\Omega := \{\text{transition}, \text{stay}\}$ denote the sample space of the random variable A defined as $A(\omega) = 1$ if $\omega = \text{transition}$, and otherwise, $A(\omega) = 0$. Obviously, this random variable follows a Bernoulli distribution with the success probability ε, i.e., $A \sim \text{Bern}(\varepsilon)$.

Furthermore, as described above, we have translated the impact of noise on a single stage to a transition from one state to another state. Consequently, this change in the states can be seen as a probabilistic change of a challenge bit, e.g., the challenge bit $c_i = 0$ is flipped to challenge bit $c_i = 1$ or vice versa. To precisely summarize, with regard to our model, when applying the challenge c that is a Boolean string, the input to the PUF f_{PUF} can be written as $c \oplus a$, where $\oplus$ denotes the bit-wise XOR operator and a is a random string composed of bits generated independently from the distribution $\text{Bern}(\varepsilon)$.

3.2 Impact of the Noise on the Measuring Element of the PUF

In the second phase, we should take into consideration the impact of uncertainty on the generation of the final response. In the literature this issue has been already explored, when discussing the precision of the measuring element (e.g., the arbiter in the case of arbiter PUFs), which makes a decision whether

[1] Note that such transition does not always lead to a change in the response of the PUF.

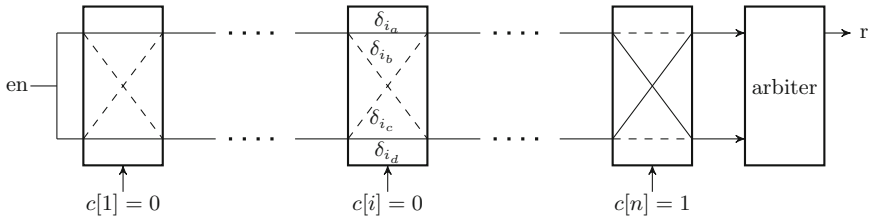


Fig. 2. Schematic of an arbiter PUF composed of n stages and an arbiter terminating the chain. When applying a challenge bit to a respective stage, either the realizations $x_{i,1}$ or the realization $x_{i,0}$ is chosen to be involved in generating the response of the PUF. For instance, we have $x_{i,1} = \delta_{i_a} - \delta_{i_d}$ and $x_{i,0} = \delta_{i_b} - \delta_{i_c}$.

the response of the PUF to the respective challenge is "0" or "1" [8,16,32]. Clearly, being limited in the precision, such component may change the output of the PUF. For the purpose of this paper, we are not interested in the real-world distribution of this noise, but in how the effect of this noise on the responses can be precisely described. A useful interpretation of this effect is given in [15], namely that after generating the response of the PUF an unfair coin (Head with probability $1 - \eta$) is flipped. Depending on the outcome, the final response is determined: when the outcome is Head, the response generated by the PUF remains unchanged, or otherwise, the response of the PUF is flipped. Here we follow the same principle to model the uncertainty with regard to the final response. Let a random variable B represent the impact of a limited precision of the physical component that makes the decision about the final answer. As explained above, this random variable also follows a Bernoulli distribution with the success probability $1 - \eta$, i.e., $B \sim \text{Bern}\,(1 - \eta)$. For the sake of readability hereafter we denote $\text{Bern}\,(1 - \eta)$ by R, and $\text{Bern}\,(\varepsilon)$ by D. We have already defined the random string a containing independent random bits drawn according to the distribution D (see Sect. 3.1), and the random bit b drawn from the distribution R. Hence, the final response of the PUF can be formalized as $y = f_{\text{PUF}}(c \oplus a) \oplus b$.

3.3 Practical Implications of the Noise Model

In this section, we explain, how a relation between the parameters introduced (in Sects. 3.1–3.2) and real-world PUFs can be established.

Arbiter PUFs: First we consider the impact of the noise on a single stage of an arbiter PUF. For the arbiter PUF family, the realizations $x_{i,0}$ and $x_{i,1}$ are associated with the difference between the delays of crossed and straight signal paths, namely, $\delta_{i_a} - \delta_{i_d} = x_{i,1}$ and $\delta_{i_b} - \delta_{i_c} = x_{i,0}$, see Fig. 2. When the difference between these variables is small and the challenge bit c_i is applied to the stage, in the presence of the noise it is not possible to make a decision whether $x_{i,0}$ or $x_{i,1}$ has impacted the final response of the PUF.

Moreover, the impact of the noise on the response of the PUF can be explained by considering the limited precision of the arbiter terminating the chain, see Fig. 2 [16]. In this case, if after the final stage the delay difference between the upper and the lower paths is smaller than the precision of the arbiter, the arbiter could enter a metasable state and thus generate a wrong response.

Ring Oscillator (RO) PUFs: The response of this PUF is generated according to the difference between the frequencies of two ROs selected by the challenge. In other words, the challenge determines a pair of ROs that contributes to the final response of the PUF. When the frequency differences of ROs in two pairs vary insignificantly, under noisy conditions one of those RO pairs can mimic another one. Therefore, it can be thought that some of the bits of the challenge applied to the PUF are flipped so that another RO pair makes impact on the final response of the PUF.

Furthermore, the limited precision of the counters measuring the frequencies of the ROs can affect the response of the PUF. More precisely, if the difference in the oscillation frequencies of a selected RO pair is not significant, the counters cannot measure the frequencies with high precision. Comparison of uncertain frequency measurements can lead to the generation of a wrong response.

Bistable Ring (BR) PUFs: Although a precise analytical model of the BR PUF is missing in the literature [12], we can still describe the impact of the noise on individual stages. For a given challenge in the BR PUF, n inverters are selected, and upon setting the reset signal to low, the created inverter ring starts to oscillate until it settles down to a valid logical state. In this case, the process variables can be intrinsic differences in the propagation delays and electrical gains of each inverter. Therefore, based on the environmental conditions, the noise can be added to the realization of the process variables. However, in contrast to the arbiter and RO PUFs, there is no *explicit* measuring element in this PUF architecture. But the required additional measurement element which introduces noise for this type of PUF is explicitly discussed in [12,21,22].

3.4 Modeling the Noise from the Perspective of Machine Learning

With regard to the discussion from the previous Sects. 3.1–3.3, PUFs can be thought of as Boolean functions, whose input-output behavior is determined by random process variations as well as the inevitable impact of random noise. In line with this view, a model of PUFs as illustrated in Fig. 3 can be established. This model can be seen as an extension of a model introduced and evaluated practically in [29]. Our model is composed of two components: random and deterministic components. The random component represents the random environmental noise, whereas the deterministic component accounts for the deterministic Boolean function realized in the chip. In other words, in the absence of noise, the response of the PUF to a challenge applied repeatedly remains the same. The building blocks as well as the parameters related to the model have

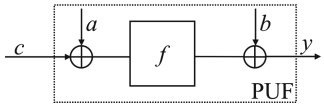


Fig. 3. Our model composed of blocks representing a noisy PUF. The random string a contains independent random bits drawn according to the distribution D (see Sect. 3.1), and the random bit b is drawn from the distribution R (see Sect. 3.2). From the machine learning point of view, they refer to attribute noise and classification noise, respectively.

been introduced previously, although their interpretations in the machine learning context have not yet been considered. As the next step in our framework, this section elaborates on how the functionality of a noisy PUF, as shown in Fig. 3, can be described from the point of view of machine learning. To this end, the following Lemma plays an important role, cf. [6].

Lemma 1. *Consider U, D and R that are a uniform, and two arbitrary distributions*[2] *over the space $\{0,1\}$, respectively. Moreover, let the function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be an arbitrary Boolean function. Let $C \in_U \mathbb{F}_2^n$, $A \in_D \mathbb{F}_2^n$ and $B \in_R \mathbb{F}_2$ be independent random strings and a random variable, respectively. The random variables $(C, f(C \oplus A) \oplus B)$ and $(C \oplus A, f(C) \oplus B)$ follow identical distribution.*

For the proof of this Lemma, we refer to [6]. The conclusions drawn from it are of great importance for us. First, from the machine learning point of view, the noise represented by the random variable A is called the "attribute noise", whereas the random variable B corresponds to the "classification noise". The issue of attribute and classification noise learnability has been well addressed in the relevant literature, see, e.g., [6,52]. Secondly, thanks to the seminal paper published by Bshouty et al. [6], a relationship between machine learning under noisy conditions and the noise sensitivity of Boolean functions has been established. The consequence of this relation is that efficient algorithms developed to estimate the Fourier coefficients of an unknown function can be applied to learn the respective function *even* under noisy conditions.

4 Fourier Analysis Based Attacks Against PUFs

To mount our attack, we apply an algorithm proposed by Linial, Mansour, and Nisan [28] to estimate the Fourier coefficients of an unknown function (i.e., so-called LMN-style algorithm). The rationale behind the LMN-style algorithm, originally called "low degree" algorithm [35], is that some classes of Boolean functions can be approximated by taking into account solely a small number of their Fourier coefficients (called "low" coefficients), corresponding to small subsets of $[n]$ (see Sect. 2.2).

[2] Regarding the physical properties of noisy PUFs we have defined the distributions D and R precisely, but in general these distribution can be arbitrary.

Theorem 1 (Low degree algorithm) [28,35,39]**.** *Assume that an algorithm can determine a set* $\mathcal{S} \subseteq 2^{[n]}$ *containing subsets of* $[n]$ *so that* $\sum_{S \in \mathcal{S}} \hat{f}(S)^2 \geq 1-\varepsilon$*. The algorithm is given a pre-defined confidence level* δ *and access to a polynomial number of input-output pairs of the Boolean function* f *that are chosen uniformly at random. With probability* $1-\delta$ *the algorithm delivers a Boolean function* h *that is an* ε*-approximator of the Boolean function* f *such that*

$$\sum_{S \subseteq [n]} \left(\hat{f}(S) - \hat{h}(S) \right)^2 \leq \varepsilon.$$

The running time of the algorithm is poly $(|\mathcal{S}|, n, 1/\varepsilon, \log_2(1/\delta))$.

For the proof of this theorem, we refer the reader to [28,35], in which the mechanism for determining the set $\mathcal{S}$, and the lower bound on the number of input-output pairs required by the algorithm has been discussed extensively.

4.1 An LMN-Style Algorithm for RO PUFs

Although the security of these PUFs can be easily broken by simply reading out all CRPs, launching a machine learning attack in the specific circumstance of having limited access to the CRPs (e.g., eavesdropping them) has been addressed in the literature, see for instance [14,42]. In the present case, learning of noisy RO PUFs has not been discussed. Our proof of the existence of an LMN-style algorithm for RO PUFs relies on the fact that these PUFs can be represented by k-DLs [14].

Theorem 2 [27,35]**.** *An LMN-style algorithm can be employed that with probability* $1-\delta$ *delivers a Boolean function* h *approximating a decision list* L*, which represents an RO PUF. The running time of this algorithm is poly* $(n, \log_2(1/\varepsilon), \log_2(1/\delta))$.

The proof sketch can be summarized as follows. According to results presented in [14], an RO PUF can be represented by a DL. Furthermore, Mansour proved that a DL could be approximated by a Boolean function h, whose Fourier coefficients concentrate only on a small set of variables, namely, $\log_2(1/\varepsilon)$ variables [35][3]. To find this set of variables, the low degree algorithm can be applied to deliver h and the running time of that is *poly* $(n, \log_2(1/\varepsilon), \log_2(1/\delta))$.

4.2 An LMN-Style Algorithm for Arbiter PUFs

To prove the existence of an LMN-style algorithm for arbiter PUFs we argue as follows. It is known that if a Boolean function exhibits a bounded, small noise sensitivity, its Fourier coefficients are mainly low coefficients. More precisely, the following Corollary can be proved (for the proof see Corollary 2.3.3 in [39]) that forms the basis for proof.

[3] Here we do not discuss the details of the proof. For the proof cf. [35].

Corollary 1 [39]. *Consider* $\alpha : [0,1/2] \rightarrow [0,1]$ *that is a strictly increasing continuous function so that* $NS_\varepsilon(f) \leq \alpha(\varepsilon)$. *We have* $\sum_{|S|\geq m} \hat{f}(S)^2 \leq \varepsilon$, *where* $m = 1/\alpha^{-1}(\varepsilon/2.32))$ *and* $\alpha^{-1}(\cdot)$ *denotes the inverse of the function* $\alpha(\cdot)$.

Now Corollary 2 states how Corollary 1 can be applied to prove the existence of an LMN-style algorithm for arbiter PUFs.

Corollary 2. *Representing an arbiter PUF by an LTF, a Boolean function h approximating this LTF can be delivered by an LMN-style algorithm, whose running time is polynomial in* n, $1/\varepsilon^2$, *and* $\log_2(1/\delta)$.

Proof: Thanks to the results reported in [18,33,42], LTFs are appropriate representations of arbiter PUFs. For any LTF f its noise sensitivity is a bounded, small value depending only on ε, namely we have $NS_\varepsilon(f) \leq 8.54\sqrt{\varepsilon}$ [25]. Now fix $\alpha(\varepsilon) = \sqrt{\varepsilon}$. According to Corollary 1, the running time of the LMN-style algorithm is polynomial in $O(n^m)$, where $m = 1/\alpha^{-1}(\varepsilon/2.32)$. ■

4.3 An LMN-Style Algorithm for BR PUFs

Similar to the proof of the existence of an LMN-style algorithm for arbiter PUFs, we take advantage of the properties of the Boolean functions representing BR PUFs. More specifically, we rely on the fact that a BR PUF can be represented by k-junta, where k is a (relatively) small constant value for practical values of n, as demonstrated in [12]. Moreover, the noise sensitivity of a k-junta function is a bounded, small value: $NS_\varepsilon(f) \leq k\varepsilon/2.$, see, e.g., [17]. Now the following corollary of can be formulated to prove the existence of an LMN-style algorithm for BR PUFs.

Corollary 3. *An LMN-style algorithm can be applied to deliver an* ε*-approximator for a k-junta representing a BR PUF. The running time is polynomial in* n, $1/\varepsilon$, *and* $\log_2(1/\delta)$.

4.4 Provability in the Sense of PAC Model

The low degree algorithm mainly aims to provide an approximator of a Boolean function with a given probability, when it is given a polynomial number of input-output pairs of the Boolean function that are chosen uniformly at random. However, its existence has a serious consequence. More specifically, if the set $\mathcal{S}$ is composed of all the subsets of low degree, Theorem 1 introduces a PAC learning algorithm under the uniform distribution [39]. Before formulating this precisely, we first shift our focus to the issue of dealing with noise.

As explored in Sect. 3.4, we take the attribute and the classification noise into account. The question is how these processes affect the functionality and the efficiency of an LMN-style algorithm. This issue is well addressed by Bshouty et al., [6], and here we briefly summarize their results. They have shown that the attribute and the classification noise attenuate the Fourier coefficients, which

Table 1. Results for learning RO PUFs with N rings.

The number of ROs N	# CRPs in training set	# CRPs in test set	$\lvert\mathcal{S}\rvert$	Accuracy	η	ε
256	5000	65536	1491	99.53	Noiseless	
	5000			97.45	0.1	0
	5500			98.64		
	5000			93.22	0.2	0
	7500			98.56		
	5000			89.06	0	0.1
	12000			98.73		
	5000			85.45	0	0.2
	20000			98.66		
512	7500	262144	1681	99.26	Noiseless	
	7500			93.29	0.1	0
	15000			98.66		
	7500			93.08	0.2	0
	18000			98.72		
	7500			86.03	0	0.1
	22000			99.01		
	7500			82.98	0	0.2
	30000			98.67		

the LMN-style algorithm aims to estimate from the uniformly random examples. To be exact, assume that an LMN-style algorithm attempts to estimate the Fourier coefficient $\hat{f}(S)$. Under the noisy conditions, it delivers $\hat{f}(S)(1-2\eta)\alpha_S$, where $(1-2\eta)$ and α_S are attenuation factors corresponding to the classification and attribute noise, respectively. While the former factor is known, see e.g., [15], the latter requires more attention. The attenuation factor α_S is the defined as $\alpha_S := \mathbf{E}_{a\in D}[\chi_S(a)]$, where $\mathbf{E}_{a\in D}[\cdot]$ denotes the expectation over random examples drawn from the known distribution D. As discussed in Sect. 3.4, here we consider a that is a random string, whose bits are independently generated following a Bernouli distribution Bern (2ε) with $\varepsilon \in (0,1/2]$. Hence, $|\alpha_S| = \prod_{i\in S}(1-2\varepsilon) = (1-2\varepsilon)^{|S|}$. Note that the practical implication of the attenuation factors ($(1-2\eta)$ and α_S) is that after running the LMN-style algorithm each Fourier coefficient delivered by the algorithm should be multiplied by $(1-2\eta)^{-1}$ and α_S^{-1} to eliminate the impact of the noise.

Now we can summarize the above discussion and the results presented in Sects. 4.1–4.3 in a more formal manner, as stated in Corollary 4.

Corollary 4. *Consider a given PUF that is represented by a Boolean function f_{PUF} and can be learned by applying an LMN-style learning algorithm. Then the PUF is PAC learnable under the uniform challenge distribution, even in the presence of attribute and classification noise. The running time of the PAC learner is poly* $(|\mathcal{S}|, n, 1/\varepsilon, \log_2(1/\delta), (1/1-2\eta))$.

Table 2. Results for learning arbiter PUFs with n stages.

The number of stages n	# CRPs in training set	# CRPs in test set	$\lvert\mathcal{S}\rvert$	Accuracy	η	ε
64	2000	100000	1078	99.19	Noiseless	
	2000			97.30	0.1	0
	2250			99.02		
	2000			97.06	0.2	0
	2350			98.86		
	2000			97.09	0	0.1
	2300			99.15		
	2000			97.97	0	0.2
	2300			99.28		
128	2100			99.51	Noiseless	
	2100			98.81	0.1	0
	2300			99.06		
	2100			96.94	0.2	0
	2500			99.29		
	2100			97.23	0	0.1
	2500			99.23		
	2100			98.04	0	0.2
	2500			99.45		

5 Results and Discussion

The effectiveness of our proposed attack is evaluated by conducting simulations on data collected from PUFs that are implemented on FPGAs. The PUF simulators, as well as LMN algorithm, are implemented in Matlab [1]. To simulate the challenge-response behaviors of the PUFs, we have taken the real physical properties of the PUFs into account. For instance, the maximum delay deviation of each inverter and the precision of the arbiter used in our arbiter PUF chain are equal to 9 ps and 2.5 ps, respectively, as reported for a Xilinx Virtex-5 FPGA (65 nm technology) [31,32]. The delays of the stages are generated with respect to a Gaussian distribution with the above-mentioned maximum deviation. By applying a random challenge chosen uniformly, the response of the arbiter PUF is generated and stored in our data set. As for RO PUFs, similar to the approach introduced in [14], the publicly accessible measurement results from a dataset [43] have been taken into account. These results contain 100 samples of the frequency of each and every ring-oscillators, which comprise RO PUFs with 512 rings implemented on 193 Xilinx Spartan-3 FPGAs (90 nm technology). These frequencies are the inputs of our RO PUF simulator that mimics the challenge-response behavior of RO PUFs with 256 and 512 ring-oscillators. The RO PUF simulator randomly selects N ($N = 256, 512$) frequencies corresponding to N different ring-oscillators. Feeding the simulator with random

Table 3. Results for learning BR PUFs with n stages.

The number of stages n	# CRPs in training set	# CRPs in test set	$\lvert\mathcal{S}\rvert$	Accuracy	η	ε
32	500	100000	59	99.72	Noiseless	
	500			95.91	0.1	0
	950			99.45		
	500			95.02	0.2	0
	950			99.63		
	500			94.39	0	0.1
	1000			99.55		
	500			94.72	0	0.2
	1200			99.40		
64	500		165	99.53	Noiseless	
	500			97.3	0.1	0
	900			99.19		
	500			92.65	0.2	0
	950			98.93		
	500			97.56	0	0.1
	950			99.23		
	500			95.63	0	0.2
	950			99.39		

challenges a pair of rings is chosen and their frequency are compared to generate the response.

Moreover, our BR PUF simulator relies on the results presented in [12], where BR PUFs have been implemented on Altera Cyclone IV FPGAs (60 nm technology). The internal functionality of these PUFs has been simulated by taking k-juntas into account. This is valid since in a follow-up work [13], the authors of [12] have demonstrated that a BR PUF (with practical values of n, e.g., 32 and 64) belongs to the class of k-junta functions. Hence, to simulate the challenge-response behavior of BR PUFs, the value k and the conjunctive rule presented in above studies are considered.

For all PUFs, the procedure of adding classification and attribute noise is as discussed in Sect. 3. In our experiments, we have $\delta = 0.01$, and various levels of noises: $\eta = 0, 0.1, 0.2$ and $\varepsilon = 0.1, 0.2$. All simulators and the LMN algorithms are implemented on a MacBook Pro with 2.6 GHz Intel Core i5 processor and 8 GB of RAM. The key difference between our approach and the methodology usually employed in ML attack scenario is that an adversary applying LMN algorithm needs to simply write a script (e.g., in Matlab), which computes a small set of Fourier coefficients. To this end, according to Theorem 1, the total number of relevant coefficients is $\lvert\mathcal{S}\rvert$ and the Fourier coefficients can be computed in a straightforward manner as shown in Sect. 2.2.

Our results are presented in Tables 1, 2 and 3. The accuracy of the final model, i.e., the approximated the function f_{PUF} generated by using the low

degree Fourier coefficients, is reported in these tables. For each experiment, the accuracy reported in the table is the minimum accuracy over 5 repetitions of the experiments that our algorithm achieves. First, as a reference, we conduct experiments on noiseless CRPs collected from PUFs. By adding the noise, the accuracy of the model decreases for the number of CRPs applied in the case of the noiseless PUF. Afterwards, we increase the number of CRPs in the training set to achieve virtually the same accuracy (with the maximum $\pm 1\%$ difference) in both cases. The results demonstrate that as promised by Corollary 4, the increase in the number of CRPs needed in the presence of noise is polynomial in noise levels.

6 Conclusion and Remarks

Our paper presents the first study on the feasibility and the applicability of a new attack, i.e., the LMN algorithm against PUFs. This algorithm has not been applied in the context of PUFs, although being known to the ML community. Thus, similar to other ML attacks against PUFs discussed in the literature, the novelty of our approach is the introduction of a new attack against PUFs, even applicable in the case of noisy CRPs. The proposed attack mainly relies on approximating the low degree Fourier coefficients by applying a so-called low degree algorithm developed in ML theory.

Moreover, our paper is the first to introduce the notion of noise sensitivity to assess the security of PUFs. This notion not only reflects the physical properties of a PUF (discussed in Sect. 3), but also it is closely related to the resilience of a PUF against LMN attacks. In this respect, the implication of Corollaries 3 and 4 (related to the existence of an LMN algorithm for PUFs) is that since the noise sensitivity of the Boolean functions representing the PUFs is a small, bounded value, an attacker can launch the LMN attack. Moreover, in the case of noisy PUFs, the attenuation factors can affect the efficiency of the LMN algorithm. In other words, if the noise sensitivity is well adjusted by the designer, the attacker cannot compute the Fourier coefficients. Hence, when designing a new PUF, it is important to consider the noise sensitivity as an indicator of the robustness of PUF against LMN attacks. We believe that in addition to the proof of PAC learnability in the presence of noise, this paper provides several interesting insights into not only the assessment of the security of PUFs, but also the design of PUFs with better security-related characteristics.

References

1. MATLAB-The Language of Technical Computing. http://www.mathworks.com/products/matlab//
2. Alkabani, Y., Koushanfar, F., Kiyavash, N., Potkonjak, M.: Trusted integrated circuits: a nondestructive hidden characteristics extraction approach. In: Solanki, K., Sullivan, K., Madhow, U. (eds.) IH 2008. LNCS, vol. 5284, pp. 102–117. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88961-8_8

3. Armknecht, F., Maes, R., Sadeghi, A., Standaert, O.X., Wachsmann, C.: A formalization of the security features of physical functions. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 397–412 (2011)
4. Armknecht, F., Moriyama, D., Sadeghi, A.-R., Yung, M.: Towards a unified security model for physically unclonable functions. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 271–287. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_16
5. Becker, G.T.: The gap between promise and reality: on the insecurity of XOR arbiter PUFs. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 535–555. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_27
6. Bshouty, N.H., Jackson, J.C., Tamon, C.: Uniform-distribution attribute noise learnability. Inf. Comput. **187**(2), 277–290 (2003)
7. Danger, J.L.: Metastable latches: a boon for combined PUF/TRNG designs. Hardware Secur. (Dagstuhl Reports on Seminar 16202) **6**(5), 78 (2016). http://drops.dagstuhl.de/opus/volltexte/2016/6721
8. Delvaux, J., Verbauwhede, I.: Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 137–142 (2013)
9. Delvaux, J., Verbauwhede, I.: Fault injection modeling attacks on 65 nm arbiter and RO sum PUFs via environmental changes. IEEE Trans. Circ. Syst. I: Regul. Pap. **61**(6), 1701–1713 (2014)
10. Devadas, S., Suh, E., Paral, S., Sowell, R., Ziola, T., Khandelwal, V.: Design and implementation of PUF-based "Unclonable" RFID ICs for anti-counterfeiting and security applications. In: 2008 IEEE International Conference on RFID, pp. 58–64. IEEE (2008)
11. Ganji, F., Krämer, J., Seifert, J.P., Tajik, S.: Lattice basis reduction attack against physically unclonable functions. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security, pp. 1070–1080. ACM (2015)
12. Ganji, F., Tajik, S., Fäßler, F., Seifert, J.-P.: Strong machine learning attack against PUFs with no mathematical model. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 391–411. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_19
13. Ganji, F., Tajik, S., Fäßler, F., Seifert, J.P.: Having no mathematical model may not secure PUFs. J. Crypt. Eng. **7**(2), 113–128 (2017)
14. Ganji, F., Tajik, S., Seifert, J.-P.: Let me prove it to you: RO PUFs are provably learnable. In: Kwon, S., Yun, A. (eds.) ICISC 2015. LNCS, vol. 9558, pp. 345–358. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30840-1_22
15. Ganji, F., Tajik, S., Seifert, J.-P.: Why attackers win: on the learnability of XOR arbiter PUFs. In: Conti, M., Schunter, M., Askoxylakis, I. (eds.) Trust 2015. LNCS, vol. 9229, pp. 22–39. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22846-4_2
16. Ganji, F., Tajik, S., Seifert, J.P.: PAC learning of arbiter PUFs. J. Cryptogr. Eng. **6**(3), 249–258 (2016)
17. Garban, C., Steif, J.E.: Noise Sensitivity of Boolean Functions and Percolation, vol. 5. Cambridge University Press, Cambridge (2014)
18. Gassend, B., Lim, D., Clarke, D., Van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits. Concurr. Comput.: Pract. Exp. **16**(11), 1077–1098 (2004)
19. Hammouri, G., Öztürk, E., Sunar, B.: A tamper-proof and lightweight authentication scheme. Pervasive Mob. Comput. **4**(6), 807–818 (2008)

20. Helfmeier, C., Boit, C., Nedospasov, D., Seifert, J.P.: Cloning physically unclonable functions. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 1–6 (2013)
21. Hesselbarth, R., Manich, S., Sigl, G.: Modeling and analyzing bistable ring based PUFs (2015). http://futur.upc.edu/16918245. Accessed 15 Mar 2017
22. Hesselbarth, R., Sigl, G.: Fast and reliable PUF response evaluation from unsettled bistable rings. In: 2016 Euromicro Conference on Digital System Design (DSD), pp. 82–90 (2016)
23. Hospodar, G., Maes, R., Verbauwhede, I.: Machine learning attacks on 65nm arbiter PUFs: accurate modeling poses strict bounds on usability. In: WIFS, pp. 37–42 (2012)
24. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
25. Klivans, A.R., O'Donnell, R., Servedio, R.A.: Learning intersections and thresholds of halfspaces. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, pp. 177–186 (2002)
26. Koushanfar, F.: Hardware metering: a survey. In: Tehranipoor, M., Wang, C. (eds.) Introduction to Hardware Security and Trust, pp. 103–122. Springer, New York (2012). https://doi.org/10.1007/978-1-4419-8080-9_5
27. Kushilevitz, E., Mansour, Y.: Learning decision trees using the fourier spectrum. SIAM J. Comput. **22**(6), 1331–1348 (1993)
28. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform, and learnability. J. ACM (JACM) **40**(3), 607–620 (1993)
29. Maes, R.: An accurate probabilistic reliability model for silicon PUFs. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 73–89. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_5
30. Maes, R.: Physically Unclonable Functions: Constructions, Properties and Applications. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41395-7
31. Majzoobi, M., Dyer, E., Elnably, A., Koushanfar, F.: Rapid FPGA delay characterization using clock synthesis and sparse sampling. In: 2010 IEEE International Test Conference (ITC), pp. 1–10 (2010)
32. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF using programmable delay lines. In: 2010 IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6 (2010)
33. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Lightweight secure PUFs. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, pp. 670–673 (2008)
34. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Techniques for design and implementation of secure reconfigurable PUFs. ACM Trans. Reconfigurable Technol. Syst. (TRETS) **2**, 5 (2009)
35. Mansour, Y.: Learning boolean functions via the fourier transform. In: Roychowdhury, V., Siu, K.Y., Orlitsky, A. (eds.) Theoretical Advances in Neural Computation and Learning, pp. 391–424. Springer, Boston (1994). https://doi.org/10.1007/978-1-4615-2696-4_11
36. Mehrotra, V.: Modeling the effects of systematic process variation on circuit performance. Ph.D. thesis, Massachusetts Institute of Technology (2001)
37. Merli, D., Schuster, D., Stumpf, F., Sigl, G.: Semi-invasive EM attack on FPGA RO PUFs and countermeasures. In: Proceedings of the Workshop on Embedded Systems Security, p. 2 (2011)
38. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014)

39. O'Donnell, R.W.: Computational applications of noise sensitivity. Ph.D. thesis, Massachusetts Institute of Technology (2003)
40. Panangaden, P.: Labelled Markov Processes. Imperial College Press, London (2009)
41. Rivest, R.L.: Learning decision lists. Mach. Learn. **2**(3), 229–246 (1987)
42. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 237–249 (2010)
43. Secure Embedded Systems (SES) Lab at Virginia Tech: On-chip Variability Data for PUFs. http://rijndael.ece.vt.edu/puf/artifacts.html
44. Srivastava, A., Sylvester, D., Blaauw, D.: Statistical Analysis and Optimization for VLSI: Timing and Power. Springer, Heidelberg (2006). https://doi.org/10.1007/b137645
45. Tajik, S., et al.: Photonic side-channel analysis of arbiter PUFs. J. Cryptol. **30**(2), 550–571 (2016)
46. Tajik, S., et al.: Physical characterization of arbiter PUFs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 493–509. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_27
47. Tajik, S., Lohrke, H., Ganji, F., Seifert, J.P., Boit, C.: Laser fault attack on physically unclonable functions. In: 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 85–96 (2015)
48. Tuyls, P., Batina, L.: RFID-tags for anti-counterfeiting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 115–131. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_8
49. Van Herrewege, A., et al.: Reverse fuzzy extractors: enabling lightweight mutual authentication for PUF-enabled RFIDs. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 374–389. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_27
50. Wang, X., Tehranipoor, M.: Novel physical unclonable function with process and environmental variations. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1065–1070 (2010)
51. Yu, M.D., Hiller, M., Delvaux, J., Sowell, R., Devadas, S., Verbauwhede, I.: A lockdown technique to prevent machine learning on PUFs for lightweight authentication. IEEE Trans. Multi-Scale Comput. Syst. **PP**(99), 146–159 (2016)
52. Zhu, X., Wu, X.: Class noise vs attribute noise: a quantitative study. Artif. Intell. Rev. **22**(3), 177–210 (2004)

On the Computational Complexity of Minimal Cumulative Cost Graph Pebbling

Jeremiah Blocki and Samson Zhou(✉)

Department of Computer Science, Purdue University, West Lafayette, IN, USA
jblocki@purdue.edu, samsonzhou@gmail.com

Abstract. We consider the computational complexity of finding a legal black pebbling of a DAG $G = (V, E)$ with minimum cumulative cost. A black pebbling is a sequence $P_0, \ldots, P_t \subseteq V$ of sets of nodes which must satisfy the following properties: $P_0 = \emptyset$ (we start off with no pebbles on G), $\mathsf{sinks}(G) \subseteq \bigcup_{j \leq t} P_j$ (every sink node was pebbled at some point) and $\mathsf{parents}\big(P_{i+1} \backslash P_i\big) \subseteq P_i$ (we can only place a new pebble on a node v if all of v's parents had a pebble during the last round). The cumulative cost of a pebbling $P_0, P_1, \ldots, P_t \subseteq V$ is $\mathsf{cc}(P) = |P_1| + \ldots + |P_t|$. The cumulative pebbling cost is an especially important security metric for data-independent memory hard functions, an important primitive for password hashing. Thus, an efficient (approximation) algorithm would be an invaluable tool for the cryptanalysis of password hash functions as it would provide an automated tool to establish tight bounds on the amortized space-time cost of computing the function. We show that such a tool is unlikely to exist in the most general case. In particular, we prove the following results.

- It is NP-Hard to find a pebbling minimizing cumulative cost.
- The natural linear program relaxation for the problem has integrality gap $\tilde{O}(n)$, where n is the number of nodes in G. We conjecture that the problem is hard to approximate.
- We show that a related problem, find the minimum size subset $S \subseteq V$ such that $\mathsf{depth}(G-S) \leq d$, is also NP-Hard. In fact, under the Unique Games Conjecture there is no $(2-\epsilon)$-approximation algorithm.

1 Introduction

Given a directed acyclic graph (DAG) $G = (V, E)$ the goal of the (parallel) black pebbling game is to start with pebbles on some source nodes of G and ultimately place pebbles on all sink nodes (not necessarily simultaneously). The game is played in rounds and we use $P_i \subseteq V$ to denote the set of currently pebbled nodes on round i. Initially all nodes are unpebbled, $P_0 = \emptyset$, and in each round $i \geq 1$ we may only include $v \in P_i$ if all of v's parents were pebbled in the previous configuration ($\mathsf{parents}(v) \subseteq P_{i-1}$) or if v was already pebbled in the last round ($v \in P_{i-1}$). In the sequential pebbling game we can place at most one

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 329–346, 2018.
https://doi.org/10.1007/978-3-662-58387-6_18

new pebble on the graph in any round (i.e., $|P_i \backslash P_{i-1}| \leq 1$), but in the parallel pebbling game no such restriction applies.

Let $\mathcal{P}_G^{\parallel}$ (resp. $\mathcal{P}_G$) denote the set of all valid parallel (resp. sequential) pebblings of G. We define the cumulative cost (respectively space-time cost) of a pebbling $P = (P_1, \ldots, P_t) \in \mathcal{P}_G^{\parallel}$ to be $\mathsf{cc}(P) = |P_1| + \ldots + |P_t|$ (resp. $\mathsf{st}(P) = t \times \max_{1 \leq i \leq t} |P_i|$), that is, the sum of the number of pebbles on the graph during every round. The *parallel* cumulative pebbling cost of G, denoted $\Pi_{cc}^{\parallel}(G)$ (resp. $\Pi_{st}(G) = \min_{P \in \mathcal{P}_G} \mathsf{st}(G)$), is the cumulative cost of the best legal pebbling of G. Formally,

$$\Pi_{cc}^{\parallel}(G) = \min_{P \in \mathcal{P}_G^{\parallel}} \mathsf{cc}(P), \text{ and} \qquad \Pi_{st}(G) = \min_{P \in \mathcal{P}_G} \mathsf{st}(P).$$

In this paper, we consider the computational complexity of $\Pi_{cc}^{\parallel}(G)$, showing that the value is `NP-Hard` to compute. We also demonstrate in the full version of the paper [BZ16] that the natural linear programming relaxation for approximating $\Pi_{cc}^{\parallel}(G)$ has a large integrality gap and therefore any approximation algorithm likely requires more powerful techniques.

1.1 Motivation

The pebbling cost of a DAG G is closely related to the cryptanalysis of data-independent memory hard functions (iMHF) [AS15], a particularly useful primitive for password hashing [PHC, BDK16]. In particular, an efficient algorithm for (approximately) computing $\Pi_{cc}^{\parallel}(G)$ would enable us to automate the cryptanalysis of candidate iMHFs. The question is particularly timely as the Internet Research Task Force considers standardizing Argon2i [BDK16], the winner of the password hashing competition [PHC], despite recent attacks [CGBS16, AB16, ABP17] on the construction. Despite recent progress [AB17, ABP17, BZ17] the precise security of Argon2i and alternative constructions is poorly understood.

An iMHF is defined by a DAG G (modeling data-dependencies) on n nodes $V = \{1, \ldots, n\}$ and a compression function H (usually modeled as a random oracle in theoretical analysis)[1]. The label ℓ_1 of the first node in the graph G is simply the hash $H(x)$ of the input x. A vertex $i > 1$ with parents $i_1 < i_2 < \cdots < i_\delta$ has label $\ell_i(x) = H(i, \ell_{i_1}(x), \ldots, \ell_{i_\delta}(x))$. The output value is the label ℓ_n of the last node in G. It is easy to see that any legal pebbling of G corresponds to an algorithm computing the corresponding iMHF. Placing a new pebble on node i corresponds to computing the label ℓ_i and keeping (resp. discarding) a pebble on node i corresponds to storing the label in memory (resp. freeing memory). Alwen and Serbinenko [AS15] proved that in the parallel random oracle model (pROM)

[1] Because the data-dependencies in an iMHF are specified by a static graph, the induced memory access pattern does not depend on the secret input (e.g., password). This makes iMHFs resistant to side-channel attacks. Data-dependent memory hard functions (MHFs) like `scrypt` [Per09] are potentially easier to construct, but they are potentially vulnerable to cache-timing attacks.

of computation, *any* algorithm evaluating such an iMHF could be reduced to a pebbling strategy with (approximately) the same cumulative memory cost.

It should be noted that *any* graph G on n nodes has a sequential pebbling strategy $P \in \mathcal{P}_G$ that finishes in n rounds and has cost $\mathsf{cc}(P) \leq \mathsf{st}(P) \leq n^2$. Ideally, a good iMHF construction provides the guarantee that the amortized cost of computing the iMHF remains high (i.e., $\tilde{\Omega}\left(n^2\right)$) even if the adversary evaluates many instances (e.g., different password guesses) of the iMHF. Unfortunately, neither large $\Pi_{st}(G)$ nor large $\min_{P \in \mathcal{P}_G^{\parallel}} \mathsf{st}(P)$, are sufficient to guarantee that $\Pi_{cc}^{\parallel}(G)$ is large [AS15]. More recently Alwen and Blocki [AB16] showed that Argon2i [BDK16], the winner of the recently completed password hashing competition [PHC], has much lower than desired amortized space-time complexity. In particular, $\Pi_{cc}^{\parallel}(G) \leq \tilde{O}\left(n^{1.75}\right)$.

In the context of iMHFs, it is important to study $\Pi_{cc}^{\parallel}(G)$, the cumulative pebbling cost of a graph G, in addition to $\Pi_{st}(G)$. Traditionally, pebbling strategies have been analyzed using space-time complexity or simply space complexity. While sequential space-time complexity may be a good model for the cost of computing a single-instance of an iMHF on a standard single-core machine (i.e., the costs incurred by the honest party during password authentication), it does not model the amortized costs of a (parallel) offline adversary who obtains a password hash value and would like to evaluate the hash function on *many* different inputs (e.g., password guesses) to crack the user's password [AS15, AB16]. Unlike $\Pi_{st}(G)$, $\Pi_{cc}^{\parallel}(G)$ models the *amortized* cost of evaluating a data-independent memory hard function on many instances [AS15, AB16].

An efficient algorithm to (approximately) compute $\Pi_{cc}^{\parallel}(G)$ would be an incredible asset when developing and evaluating iMHFs. For example, the Argon2i designers argued that the Alwen-Blocki attack [AB16] was not particularly effective for practical values of n (e.g., $n \leq 2^{20}$) because the constant overhead was too high [BDK16]. However, they could not rule out the possibility that more efficient attacks might exist[2]. As it stands, there is a huge gap between the best known upper/lower bounds on $\Pi_{cc}^{\parallel}(G)$ for Argon2i and for the new DRSample graph [ABH17], since in all practical cases the ratio between the upper bound and the lower bound is at least 10^5. An efficient algorithm to (approximately) compute $\Pi_{cc}^{\parallel}(G)$ would allow us to immediately resolve such debates by automatically generating upper/lower bounds on the cost of computing the iMHF for each running time parameters (n) that one might select in practice. Alwen *et al.* [ABP17] showed how to construct graphs G with $\Pi_{cc}^{\parallel}(G) = \Omega\left(\frac{n^2}{\log n}\right)$. This construction is essentially optimal in theory as results of Alwen and Blocki [AB16] imply that any constant indegree graph has $\Pi_{cc}^{\parallel}(G) = O\left(\frac{n^2 \log\log n}{\log n}\right)$. However, the exact constants one could obtain through

[2] Indeed, Alwen and Blocki [AB17] subsequently introduced heuristics to improve their attack and demonstrated that their attacks were effective even for smaller (practical) values of n by simulating their attack against real Argon2i instances.

a theoretical analysis are most-likely small. A proof that $\Pi^{\|}_{cc}(G) \geq \frac{10^{-6} \times n^2}{\log n}$ would be an underwhelming security guarantee in practice, where we may have $n \approx 10^6$. An efficient algorithm to compute $\Pi^{\|}_{cc}(G)$ would allow us to immediately determine whether these new constructions provide meaningful security guarantees in practice.

1.2 Results

We provide a number of computational complexity results. Our primary contribution is a proof that the decision problem "is $\Pi^{\|}_{cc}(G) \leq k$ for a positive integer $k \leq \frac{n(n+1)}{2}$" is `NP-Complete`.[3] In fact, our result holds even if the DAG G has constant `indeg`.[4] We also provide evidence that $\Pi^{\|}_{cc}(G)$ is hard to approximate. Thus, it is unlikely that it will be possible to automate the cryptanalysis process for iMHF candidates. In particular, we define a natural integer program to compute $\Pi^{\|}_{cc}(G)$ and consider its linear programming relaxation. We then show in the full version of the paper [BZ16] that the integrality gap is at least $\Omega\left(\frac{n}{\log n}\right)$ leading us to conjecture that it is hard to approximate $\Pi^{\|}_{cc}(G)$ within constant factors. We also give an example of a DAG G on n nodes with the property that *any* optimal pebbling (minimizing $\Pi^{\|}_{cc}$) requires more than n pebbling rounds.

The computational complexity of several graph pebbling problems has been explored previously in various settings [GLT80,HP10]. However, minimizing cumulative cost of a pebbling is a very different objective than minimizing the space-time cost or space. For example, consider a pebbling where the maximum number of pebbles used is significantly greater than the average number of pebbles used. Thus, we need fundamentally new ideas to construct appropriate gadgets for our reduction.[5] We first introduce a natural problem that arises from solving systems of linear equations, which we call `Bounded 2-Linear Covering` (`B2LC`) and show that it is `NP-Complete`. We then show that we can encode a `B2LC` instance as a graph pebbling problem thus proving that the decision version of cummulative graph pebbling is `NP-Hard`.

In Sect. 5 we also investigate the computational complexity of determining how "depth-reducible" a DAG G is showing that the problem is `NP-Complete` even if G has constant indegree. A DAG G is (e, d)-reducible if there exists a subset $S \subseteq V$ of size $|S| \leq e$ such that $\mathsf{depth}(G - S) < d$. That is, after removing nodes in the set S from G, any remaining directed path has length less than d. If G is not (e, d)-reducible, we say that it is (e, d)-depth robust. It is known that a graph has high cumulative cost (e.g., $\tilde{\Omega}\left(n^2\right)$) if and only if the graph is highly depth robust (e.g., $e, d = \tilde{\Omega}\left(n\right)$) [AB16,ABP17]. Our reduction from

[3] Note that for any G with n nodes we have $\Pi^{\|}_{cc}(G) \leq 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$ since we can always pebble G in topological order in n steps if we never remove pebbles.

[4] For practical reasons most iMHF candidates are based on a DAG G with constant indegree.

[5] See additional discussion in Sect. 3.2.

Vertex Cover preserves approximation hardness.[6] Thus, assuming that $\mathtt{P} \neq \mathtt{NP}$ it is hard to 1.3-approximate e, the minimum size of a set $S \subseteq V$ such that $\mathsf{depth}(G - S) < d$ [DS05]. Under the Unique Games Conjecture [Kho02], it is hard to $(2 - \epsilon)$-approximate e for any fixed $\epsilon > 0$ [KR08]. In fact, we show in the full version of the paper [BZ16] that the linear programming relaxation to the natural integer program to compute e has an integrality gap of $\Omega(n/\log n)$ leading us to conjecture that it is hard to approximate e.

2 Preliminaries

Given a directed acyclic graph (DAG) $G = (V, E)$ and a node $v \in V$ we use $\mathsf{parents}(v) = \{u : (u, v) \in E\}$ to denote the set of nodes u with directed edges into node v and we use $\mathsf{indeg}(v) = |\mathsf{parents}(v)|$ to denote the number of directed edges into node v. We use $\mathsf{indeg}(G) = \max_{v \in V} \mathsf{indeg}(v)$ to denote the maximum indegree of any node in G. For convenience, we use indeg instead of $\mathsf{indeg}(G)$ when G is clear from context. We say that a node $v \in V$ with $\mathsf{indeg}(v) = 0$ is a source node and a node with no outgoing edges is a sink node. We use $\mathtt{sinks}(G)$ (resp. $\mathtt{sources}(G)$) to denote the set of all sink nodes (resp. source nodes) in G. We will use $n = |V|$ to denote the number of nodes in a graph, and for convenience we will assume that the nodes $V = \{1, 2, 3, \ldots, n\}$ are given in topological order (i.e., $1 \leq j < i \leq n$ implies that $(i, j) \notin E$). We use $\mathsf{depth}(G)$ to denote the length of the longest directed path in G. Given a positive integer $k \geq 1$ we will use $[k] = \{1, 2, \ldots, k\}$ to denote the set of all integers 1 to k (inclusive).

Definition 1. *Given a DAG $G = (V, E)$ on n nodes a legal pebbling of G is a sequence of sets $P = (P_0, \ldots, P_t)$ such that:*

1. $P_0 = \emptyset$
2. $\forall i > 0$, $v \in P_i \backslash P_{i-1}$ *we have* $\mathsf{parents}(v) \subseteq P_{i-1}$
3. $\forall v \in \mathtt{sinks}(G)$ $\exists 0 < j \leq t$ *such that* $v \in P_j$

The cumulative cost of the pebbling P is $\mathsf{cc}(P) = \sum_{i=1}^{t} |P_i|$, and the space-time cost is $\mathsf{st}(P) = t \times \max_{0<j\leq t} |P_i|$.

The first condition states that we start with no pebbles on the graph. The second condition states that we can only add a new pebble on node v during round i if we already had pebbles on all of v's parents during round $i-1$. Finally, the last condition states that every sink node must have been pebbled during *some* round.

We use $\mathcal{P}_G^{\parallel}$ to denote the set of all legal pebblings, and we use $\mathcal{P}_G \subset \mathcal{P}_G^{\parallel}$ to denote the set of all sequential pebblings with the additional requirement that $|P_i \backslash P_{i-1}| \leq 1$ (i.e., we place at most one new pebble on the graph during ever round i). We use $\Pi_{cc}^{\parallel}(G) = \min_{P \in \mathcal{P}_G^{\parallel}} \mathsf{cc}(P)$ to denote the cumulative cost of the best legal pebbling.

[6] Note that when $d = 0$ testing whether a graph G is (e, d) reducible is *equivalent* to asking whether G has a vertex cover of size e. Our reduction establishes hardness for $d \gg 1$.

Definition 2. *We say that a directed acyclic graph (DAG) $G = (V, E)$ is (e, d)-depth robust if $\forall S \subseteq V$ of size $|S| \leq e$ we have $\mathsf{depth}(G - S) \geq d$. If G contains a set $S \subseteq V$ of size $|S| \leq e$ such that $\mathsf{depth}(G - S) \leq d$ then we say that G is (e, d)-reducible.*

Decision Problems

The decision problem `minCC` is defined as follows:
Input: a DAG G on n nodes and an integer $k < n(n+1)/2$. (See footnote 3.)
Output: *Yes*, if $\Pi^{\parallel}_{cc}(G) \leq k$; otherwise *No*.

Given a constant $\delta \geq 1$ we use $\texttt{minCC}_\delta$ to denote the above decision problem with the additional constraint that $\mathsf{indeg}(G) \leq \delta$. It is clear that $\texttt{minCC} \in \texttt{NP}$ and $\texttt{minCC}_\delta \in \texttt{NP}$ since it is easy to verify that a candidate pebbling P is legal and that $\mathsf{cc}(P) \leq k$. One of our primary results is to show that the decision problems `minCC` and $\texttt{minCC}_2$ are `NP-Complete`. In fact, these results hold even if we require that the DAG G has a single sink node.

The decision problem $\texttt{REDUCIBLE}_d$ is defined as follows:
Input: a DAG G on n nodes and positive integers $e, d \leq n$.
Output: *Yes*, if G is (e, d)-reducible; otherwise *No*.

We show that the decision problem $\texttt{REDUCIBLE}_d$ is `NP-Complete` for *all* $d > 0$ by a reduction from Cubic Vertex Cover, defined below. Note that when $d = 0$ $\texttt{REDUCIBLE}_d$ *is* Vertex Cover. We use $\texttt{REDUCIBLE}_{d,\delta}$ to denote the decision problem with the additional constraint that $\mathsf{indeg}(G) \leq \delta$.

The decision problem `VC` (resp. `CubicVC`) is defined as follows:
Input: a graph G on n vertices (`CubicVC`: each with degree 3) and a positive integer $k \leq \frac{n}{2}$.
Output: *Yes*, if G has a vertex cover of size at most k; otherwise *No*.

To show that `minCC` is `NP-Complete` we introduce a new decision problem `B2LC`. We will show that the decision problem `B2LC` is `NP-Complete` and we will give a reduction from `B2LC` to `minCC`.

The decision problem Bounded 2-Linear Covering (`B2LC`) is defined as follows:
Input: an integer n, k positive integers $0 \leq c_1, \ldots, c_k$, an integer $m \leq k$ and k equations of the form $x_{\alpha_i} + c_i = x_{\beta_i}$, where $\alpha_i, \beta_i \in [n]$ and $i \in [k]$. We require that $\sum_{i=1}^{k} c_i \leq p(n)$ for some fixed polynomial n.
Output: *Yes*, if we can find mn integers $x_{y,z} \geq 0$ (for each $1 \leq y \leq m$ and $1 \leq z \leq n$) such that for each $i \in [k]$ there exists $1 \leq y \leq m$ such that $x_{y,\alpha_i} + c_i = x_{y,\beta_i}$ (that is the assignment $x_1, \ldots, x_n = x_{y,1}, \ldots, x_{y,n}$ satisfies the i^{th} equation); otherwise *No*.

3 Related Work

The sequential black pebbling game was introduced by Hewitt and Paterson [HP70], and by Cook [Coo73]. It has been particularly useful in exploring space/time trade-offs for various problems like matrix multiplication [Tom78], fast fourier transformations [SS78, Tom78], integer multiplication [SS79b] and

many others [Cha73,SS79a]. In cryptography it has been used to construct/analyze proofs of space [DFKP15,RD16], proofs of work [DNW05, MMV13] and memory-bound functions [DGN03] (functions that incur many expensive cache-misses [ABW03]). More recently, the black pebbling game has been used to analyze memory hard functions e.g., [AS15,AB16,ABP17,AT17].

3.1 Password Hashing and Memory Hard Functions

Users often select low-entropy passwords which are vulnerable to offline attacks if an adversary obtains the cryptographic hash of the user's password. Thus, it is desirable for a password hashing algorithm to involve a function $f(.)$ which is moderately expensive to compute. The goal is to ensure that, even if an adversary obtains the value $(username, f(pwd, salt), salt)$ (where $salt$ is some randomly chosen value), it is prohibitively expensive to evaluate $f(., salt)$ for millions (billions) of different password guesses. PBKDF2 (Password Based Key Derivation Function 2) [Kal00] is a popular moderately hard function which iterates the underlying cryptographic hash function many times (e.g., 2^{10}). Unfortunately, PBKDF2 is insufficient to protect against an adversary who can build customized hardware to evaluate the underlying hash function. The cost computing a hash function H like SHA256 or MD5 on an Application Specific Integrated Circuit (ASIC) is dramatically smaller than the cost of computing H on traditional hardware [NBF+15].

[ABW03], observing that cache-misses are more egalitarian than computation, proposed the use of "memory-bound" functions for password hashing—a function which maximizes the number of expensive cache-misses. Percival [Per09] observed that memory costs tend to be stable across different architectures and proposed the use of memory-hard functions (MHFs) for password hashing. Presently, there seems to be a consensus that memory hard functions are the 'right tool' for constructing moderately expensive functions. Indeed, all entrants in the password hashing competition claimed some form of memory hardness [PHC]. As the name suggests, the cost of computing a memory hard function is primarily memory related (storing/retrieving data values). Thus, the cost of computing the function cannot be significantly reduced by constructing an ASIC. Percival [Per09] introduced a candidate memory hard function called `scrypt`, but `scrypt` is potentially vulnerable to side-channel attacks as its computation yields a memory access pattern that is data-dependent (i.e., depends on the secret input/password). Due to the recently completed password hashing competition [PHC] we have many candidate data-independent memory hard functions such as Catena [FLW13] and the winning contestant Argon2i-A [BDK15].[7]

[7] The specification of Argon2i has changed several times. We use Argon2i-A to refer to the version of Argon2i from the password hashing competition, and we use Argon2i-B to refer to the version that is currently being considered for standardization by the Cryptography Form Research Group (CFRG) of the IRTF [BDKJ16].

iMHFs and Graph Pebbling. All known candidate iMHFs can be described using a DAG G and a hash function H. Graph pebbling is a particularly useful as a tool to analyze the security of an iMHF [AS15,CGBS16,FLW13]. A pebbling of G naturally corresponds to an algorithm to compute the iMHF. Alwen and Serbinenko [AS15] showed that in the pROM model of computation, *any* algorithm to compute the iMHF corresponds to a pebbling of G.

Measuring Pebbling Costs. In the past, MHF analysis has focused on space-time complexity [Per09,FLW13,BCS16]. For example, the designers of Catena [FLW13] showed that their DAG G had high sequential space-time pebbling cost $\Pi_{st}(G)$ and Boneh *et al.* [BCS16] showed that Argon2i-A and their own iMHF candidate iBH ("balloon hash") have (sequential) space-time cost $\Omega(n^2)$. Alwen and Serbinenko [AS15] observed that these guarantees are insufficient for two reasons: (1) the adversary may be parallel, and (2) the adversary might amortize his costs over multiple iMHF instances (e.g., multiple password guesses). Indeed, there are now multiple known attacks on Catena [BK15,AS15,AB16]. Alwen and Blocki [AB16,AB17] gave attacks on Argon2i-A, Argon2i-B, iBH, and Catena with lower than desired amortized space-time cost—$\Pi^{\|}_{cc}(G) \leq O(n^{1.8})$ for Argon2i-B, $\Pi^{\|}_{cc}(G) \leq \tilde{O}(n^{1.75})$ for Argon2i-A and iBH and $\Pi^{\|}_{cc}(G) \leq O(n^{5/3})$ for Catena. This motivates the need to study cumulative cost $\Pi^{\|}_{cc}$ instead of space-time cost since amortized space-time complexity approaches $\Pi^{\|}_{cc}$ as the number of iMHF instances being computed increases.

Alwen *et al.* [ABP17] recently constructed a constant indegree graph G with $\Pi^{\|}_{cc}(G) = \Omega(\frac{n^2}{\log n})$. From a theoretical standpoint, this is essentially optimal as any constant indeg DAG has $\Pi^{\|}_{cc} = O(\frac{n^2 \log\log n}{\log n})$ [AB16], but from a practical standpoint the critically important constants terms in the lower bound are not well understood.

Ren and Devedas [RD17] recently proposed an alternative metric MHFs called bandwidth hardness. The key distinction between bandwidth hardness and cumulative pebbling cost is that bandwidth hardness attempts to approximate *energy costs*, while cumulative pebbling cost attempts to approximate amortized capital costs (i.e., the cost of all of the DRAM chips divided by the number of MHF instances that can be computed before the DRAM chip fails). In this paper we focus on the cumulative pebbling cost metric as we expect amortized capital costs to dominate for sufficiently large n. In particular, bandwidth costs scale linearly with the running time n (at best), while cumulative pebbling costs can scale quadratically with n.

3.2 Computational Complexity of Pebbling

The computational complexity of various graph pebbling has been explored previously in different settings [GLT80,HP10]. Gilbert *et al.* [GLT80] focused on

space-complexity of the black-pebbling game. Here, the goal is to find a pebbling which minimizes the total number of pebbles on the graph at any point in time (intuitively this corresponds to minimizing the maximum space required during computation of the associated function). Gilbert *et al.* [GLT80] showed that this problem is PSPACE complete by reducing from the truly quantified boolean formula (TQBF) problem.

The optimal (space-minimizing) pebbling of the graphs from the reduction of Gilbert *et al.* [GLT80] often require exponential time. By contrast, observe that $\texttt{minCC} \in NP$ because any DAG G with n nodes this algorithm has a pebbling P with $\texttt{cc}(P) \leq \texttt{st}(P) \leq n^2$. Thus, if we are minimizing `cc` or `st` cost, the optimal pebbling of G will trivially never require more than n^2 steps. Thus, we need different tools to analyze the computational complexity of the problem of finding a pebbling with low cumulative cost.

In the full version [BZ16], we show that the optimal pebbling from [GLT80] does take polynomial time if the TQBF formula only uses existential quantifiers (i.e., if we reduce from 3SAT). Thus, the reduction of Gilbert *et al.* [GLT80] can also be extended to show that it is `NP-Complete` to check whether a DAG G admits a pebbling P with $\texttt{st}(P) \leq k$ for some parameter k. The reduction, which simply appends a long chain to the original graph, exploits the fact that if we increase space-usage even temporarily we will dramatically increase `st` cost. However, this reduction does not extend to cumulative cost because the penalty for temporarily placing large number of pebbles can be quite small as we do not keep these pebbles on the graph for a long time.

4 NP-Hardness of `minCC`

In this section we prove that `minCC` is `NP-Complete` by reduction from `B2LC`. Showing that $\texttt{minCC} \in \texttt{NP}$ is straightforward so we will focus on proving that the decision problem is `NP-Hard`. We first provide some intuition about the reduction.

Recall that a `B2LC` instance consists of n variables $x_1, \ldots, x_n$, and k equations of the form $x_{\alpha_i} + c_i = x_{\beta_i}$, where $\alpha_i, \beta_i \in [n]$, $i \in [k]$, and each $c_i \leq p(n)$ is a positive integer bounded by some polynomial in n. The goal is to determine whether there exist m different variable assignments such that each equation is satisfied by *at least one* of the m assignments. Formally, the goal is to decide if there exists a set of $m < k$ variable assignments: $x_{y,z} \geq 0$ for each $1 \leq y \leq m$ and $1 \leq z \leq n$ so that for each $i \in [k]$ there exists $y \in [m]$ such that $x_{y,\alpha_i} + c_i = x_{y,\beta_i}$—that is the i^{th} equation $x_{\alpha_i} + c_i = x_{\beta_i}$ is satisfied by the y^{th} variable assignment $x_{y,1}, \ldots, x_{y,n}$. For example, if $k = 2$ and the equations are $x_1 + 1 = x_2$ and $x_2 + 2 = x_3$, then $m = 1$ suffices to satisfy all the equations. On the other hand, if $x_1 + 1 = x_2$ and $x_1 + 2 = x_2$, then we require $m \geq 2$ since the equations are no longer independent. Observe that for $m = 1$, `B2LC` seeks a single satisfying assignment, whereas for $m \geq k$, each equation can be satisfied by a separate assignment of the variables (specifically, the i^{th} assignment is all zeroes except $x_{\beta_i} = c_i$).

Suppose we are given an instance of `B2LC`. We shall construct a `minCC` instance $G_{\texttt{B2LC}}$ in such a way that the optimal pebbling of $G_{\texttt{B2LC}}$ has "low" cost if the

instance of B2LC is satisfiable and otherwise, has "high" cost. The graph B2LC will be constructed from three different types of gadgets: τ gadgets $C_i^1, \ldots, C_i^\tau$ for each variable x_i, a gadget E_i for each equation and a "m-assignments" gadget M_i for each variable x_i. Here τ is a parameter we shall set to create a gap between the pebbling costs of satisfiable and unsatisfiable instances of B2LC. Each gadget is described in more detail below.

Variable Gadgets. Our first gadget is a chain of length $c = \sum c_i$ so that each node is connected to the previous node, and can only be pebbled if there exists a pebble on the previous node in the previous step, such as in Fig. 1. For each variable x_i in our B2LC instance we will add τ copies of our chain gadget $C_i^1, \ldots, C_i^\tau$. Formally, for each $j \in [\tau]$ the chain gadget C_i^j consists of c vertices $v_i^{j,1}, \ldots, v_i^{j,c}$ with directed edges $\left(v_i^{j,z}, v_i^{j,z+1}\right)$ for each $z < c$. We will later add a gadget to ensure that we must walk a pebble down each of these chains m different times and that in any optimal pebbling $P \in \mathcal{P}^{\parallel}_{G_{\texttt{B2LC}}}$ (with $\texttt{cc}(P) = \Pi^{\parallel}_{cc}(G_{\texttt{B2LC}})$) the walks on each chain gadget $C_i^1, \ldots, C_i^\tau$ are synchronized e.g., for each pebbling round y and for each $z \leq c$ we have $v_i^{j,z} \in P_y \leftrightarrow \{v_i^{1,z}, \ldots v_i^{\tau,z}\} \subseteq P_y$. Intuitively, each *time* at which we begin walking a pebble down these chains will correspond to an assignment of the B2LC variable x_i. Hence, it suffices to have $c = \sum c_i$ nodes in the chain.

Fig. 1. Example variable gadget C_i^j of length $c = \sum c_i$. $G_{\texttt{B2LC}}$ replicates this gadget τ times: $C_i^1, \ldots, C_i^\tau$. Each of the τ copies behaves the same.

Equation Gadget. For the i^{th} equation $x_{\alpha_i} + c_i = x_{\beta_i}$, the gadget E_i is a chain of length $c - c_i$. For each $j \in [\tau]$ we connect the equation gadget E_i to each of the variable gadgets $C_{\alpha_i}^j$ and $C_{\beta_i}^j$ as follows: the a^{th} node e_j^a in chain E_j has incoming edges from vertices $v_{\alpha_i}^{l,a}$ and $v_{\beta_i}^{l,a+c_i}$ for all $1 \leq l \leq \tau$, as demonstrated in Fig. 2. To pebble the equation gadget, the corresponding variable gadgets must be pebbled synchronously, distance c_i apart.

Intuitively, if the equation $x_\alpha + c_i = x_\beta$ is satisfied by the j^{th} assignment, then on the j^{th} time we walk pebbles down the chain x_α and x_β, the pebbles on each chain will be synchronized (i.e., when we have a pebble on $v_\alpha^{l,a}$, the a^{th} link in the chain representing x_α we will have a pebble on the node $v_\beta^{l,a+c_i}$ during the same round) so that we can pebble all of the nodes in the equation gadget, such as in Fig. 3. On the other hand, if the pebbles on each chain are not synchronized appropriately, we cannot pebble the equation gadget. Finally, we create a single sink node linked from each of the k equation chains, which can only be pebbled if all equation nodes are pebbled.

We will use another gadget, the *assignment gadget*, to ensure that in any legal pebbling, we need to "walk" a pebble down each chain C_i^j on m different times.

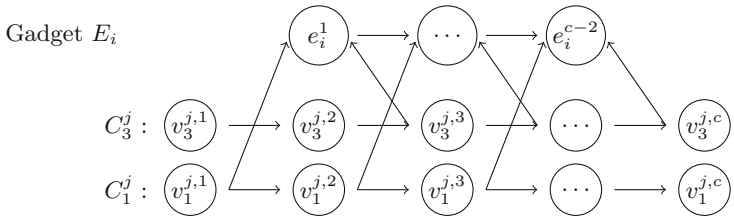


Fig. 2. The gadget E_i for equation $x_3+2 = x_1$. The example shows how E_i is connected to the variable gadgets C_1^j and C_3^j for each $j \in [\tau]$.

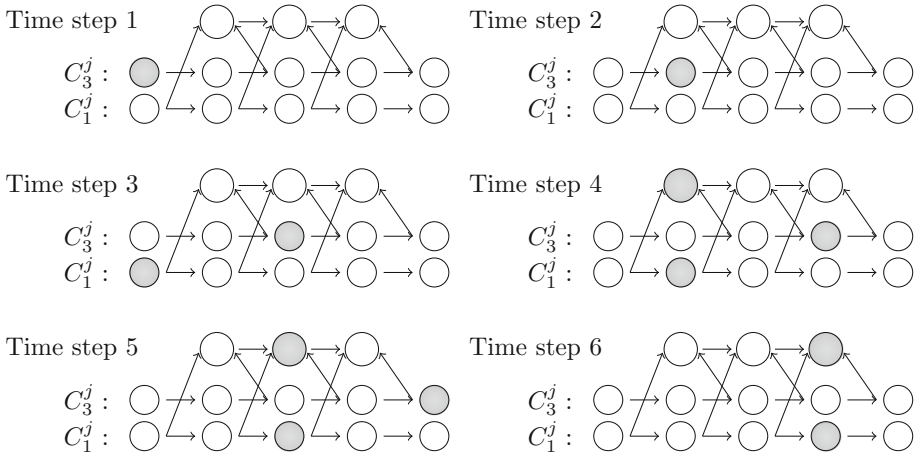


Fig. 3. A pebbling of the equation gadget $x_3 + 2 = x_1$ (at the top) using the satisfying assignment $x_3 = 1$ and $x_1 = 3$.

Each node $v_i^{j,z}$ of a variable gadget in a satisfiable B2LC instance has a pebble on it during exactly m rounds. On the other hand, the assignment gadget ensures that for any unsatisfiable B2LC instance, there exists some $i \leq n$ and $z \leq c$ such that each of the nodes $v_i^{1,z}, \ldots, v_i^{\tau,z}$ are pebbled during at least $m+1$ rounds.

We will tune the parameter τ to ensure that any such pebbling is more expensive, formalized in the full version of the paper [BZ16].

m Assignments Gadget. Our final gadget is a path of length cm so that each node is connected to the previous node. We create a path gadget M_i of length cm for each variable x_i and connect M_i to each the variable gadgets $C_i^1, \ldots C_i^\tau$ as follows: for every node z_i^{p+qc} in the path with position $p + qc > 1$, where $1 \leq p \leq c$ and $0 \leq q < m-1$, we add an edge to z_i^{p+qc} from each of the nodes $v_i^{j,p}$, $1 \leq j \leq \tau$ (that is, the p^{th} node in all τ chains $C_i^1, \ldots C_i^\tau$ representing the variable x_i). We connect the final node in each of the n paths to the final sink node v_{sink} in our graph $G_{\texttt{B2LC}}$.

Intuitively, to pebble v_{sink} we must walk a pebble down each of the gadgets M_i which in turn requires us to walk a pebble along each chain C_i^j, $1 \leq j \leq \tau$, at least m times. For example, see Fig. 4.

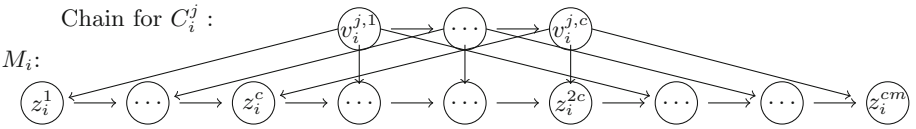


Fig. 4. The gadget M_i for variable x_i is a path of length cm. The example shows how M_i is connected to C_i^j for each $j \in [\tau]$. The example shows $m = 3$ passes and $c = 3$.

Figure 5 shows an example of a reduction in its entirety when $\tau = 1$.

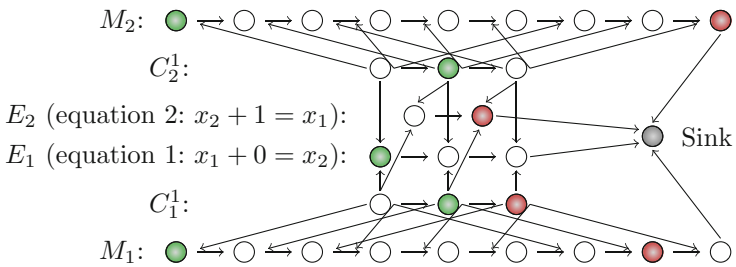


Fig. 5. An example of a complete reduction $G_{\texttt{B2LC}}$, again $m = 3$ and $c = 3$. The green nodes represent the pebbled vertices at time step 2 while the red nodes represent the pebbled vertices at time step 10. (Color figure online)

Lemma 1. *If the* B2LC *instance has a valid solution, then* $\Pi_{cc}^{\parallel}(G_{\texttt{B2LC}}) \leq \tau cmn + 2cmn + 2ckm + 1$.

Lemma 2. *If the* B2LC *instance does not have a valid solution, then* $\Pi_{cc}^{\parallel}(G_{\texttt{B2LC}}) \geq \tau cmn + \tau$.

We outline the key intuition behind Lemma 1 and Lemma 2 and refer to the full version of the paper [BZ16] for the formal proofs. Intuitively, any solution to B2LC corresponds to m walks across the τn chains C_i^j, $1 \leq i \leq n$, $1 \leq j \leq \tau$ of length c. If the B2LC instance is satisfiable then we can synchronize each of these walks so that we can pebble every equation chain E_j and path M_j along the way. Thus, the total cost is τcmn plus the cost to pebble the k equation chains E_j ($\leq 2ckm$), the cost to pebble the n paths M_j ($\leq 2cmn$) plus the cost to pebble the sink node 1.

We then prove a structural property about the optimal pebbling $P = (P_0, \ldots, P_t) \in \mathcal{P}^{\parallel}_{G_{\texttt{B2LC}}}$. In particular, we formally claim in the full version of the paper [BZ16] that if $P = (P_0, \ldots, P_t)$ is optimal (i.e., $\texttt{cc}(P) = \Pi^{\parallel}_{cc}(G_{\texttt{B2LC}})$) then during each pebbling round $y \leq t$ the pebbles on each of the chains $C_i^1, \ldots, C_i^\tau$ are synchronized. Formally, for every $y \leq t$, $i \leq n$ and $z \leq c$ we either have (1) $\left\{v_i^{1,z}, \ldots, v_i^{\tau,z}\right\} \subseteq P_y$, or (2) $\left\{v_i^{1,z}, \ldots, v_i^{\tau,z}\right\} \bigcap P_y = \emptyset$—otherwise we could reduce our pebbling cost by discarding these unnecessary pebbles.

If the B2LC instance is not satisfied then we must adopt a "cheating" pebbling strategy P, which does not correspond to a B2LC solution. We say that P is a "cheating" pebbling if some node $v_i^{j,z} \in C_i^j$ is pebbled during at least $m+1$ rounds. We can use Lemma 2 to show that the cost of *any* "cheating" pebbling is at least $\texttt{cc}(P) \geq \tau\,(mc+1)$. In particular, P must incur cost at least $\tau mc(n-1)$ to walk a pebble down each of the chains $C_{i'}^j$ with $i' \neq i$ and $1 \leq j \leq \tau$. By Lemma 2, any cheating pebbling P incurs costs at least $\tau(mc+1)$ on each of the chains $C_i^1, \ldots, C_i^\tau$. Thus, the cumulative cost is at least $\tau cmn + \tau$.

Theorem 1. minCC *is* NP-Complete.

Proof. It suffices to show that there is a polynomial time reduction from B2LC to minCC since B2LC is NP-Complete (see Theorem 3). Given an instance $\mathcal{P}$ of B2LC, we create the corresponding graph G as described above. This is clearly achieved in polynomial time. By Lemma 1, if $\mathcal{P}$ has a valid solution, then $\Pi^{\parallel}_{cc}(G) \leq \tau cmn + 2cmn + 2ckm + 1$. On the other hand, by Lemma 2, if $\mathcal{P}$ does not have a valid solution, then $\Pi^{\parallel}_{cc}(G) \geq \tau cmn + \tau$. Therefore, setting $\tau > 2cmn + 2ckm + 1$ (such as $\tau = 2cmn + 2ckm + 2$) allows one to solve B2LC given an algorithm which outputs $\Pi^{\parallel}_{cc}(G)$.

Theorem 2. $\texttt{minCC}_\delta$ *is* NP-Complete *for each* $\delta \geq 2$.

Note that the only possible nodes in $G_{\texttt{B2LC}}$ with indegree greater than two are the nodes in the equation gadgets $E_1, \ldots, E_m$, and the final sink node. The equation gadgets can have indegree up to $2\tau + 1$, while the final sink node has indegree $n + m$. To show that $\texttt{minCC}_\delta$ is NP-Complete when $\delta = 2$ we can replace the incoming edges to each of these nodes with a binary tree, so that all vertices have indegree at most two. By changing τ appropriately, we can still distinguish between instances of B2LC using the output of $\texttt{minCC}_\delta$. We refer to the full version of the paper [BZ16] for a sketch of the proof of Theorem 2.

Theorem 3. B2LC *is* NP-Complete.

To show that B2LC is NP-Complete we will reduce from the problem 3-PARTITION, which is known to be NP-Complete. The decision problem 3-PARTITION is defined as follows:

Input: A multi-set S of $m = 3n$ positive integers $x_1, \ldots, x_m \geq 1$ such that (1) we have $\frac{T}{4n} < x_i < \frac{T}{2n}$ for each $1 \leq i \leq m$, where $T = x_1 + \ldots + x_m$, and (2) we require that $T \leq p(n)$ for a fixed polynomial p.[8]

[8] We may assume $\frac{T}{4n} < x_i < \frac{T}{2n}$ by taking any set of positive integers and adding a large fixed constant to all terms, as described in [Dem14].

Output: *Yes*, if there is a partition of $[m]$ into n subsets $S_1, \ldots, S_n$ such that $\sum_{j \in S_i} x_j = \frac{T}{n}$ for each $1 \leq i \leq n$; otherwise *No*.

Fact 4. *[GJ75, Dem14]* `3-PARTITION` *is* `NP-Complete`.[9]

We refer to the full version of the paper [BZ16] for the proof of Theorem 3, where we show that there is a polynomial time reduction from `3-PARTITION` to `B2LC`.

5 NP-Hardness of $\texttt{REDUCIBLE}_d$

The attacks of Alwen and Blocki [AB16, AB17] exploited the fact that the Argon2i-A, Argon2i-B, iBH and Catena DAGs are not depth-robust. In general, Alwen and Blocki [AB16] showed that any (e, d)-reducible DAG G can be pebbled with cumulative cost $O\big(ne + n\sqrt{nd}\big)$. Thus, depth-robustness is a necessary condition for a secure iMHF. Recently, Alwen *et al.* [ABP17] showed that depth-robustness is sufficient for a secure iMHF. In particular, they showed that an (e, d)-depth reducible graph has $\Pi^{\parallel}_{cc}(G) \geq ed$.[10] Thus, to cryptanalyze a candidate iMHF it would be useful to have an algorithm to test for depth-robustness of an input graph G. However, we stress that (constant-factor) hardness of $\texttt{REDUCIBLE}_d$ does not directly imply that `minCC` is `NP-Hard`. To the best of our knowledge no one has explored the computational complexity of testing whether a given DAG G is (e, d)-depth robust.

We have many constructions of depth-robust graphs [EGS75, PR80, Sch82, Sch83, MMV13], but the constant terms in these constructions are typically not well understood. For example, Erdös, Graham and Szemerédi [EGS75] constructed an $\big(\Omega(n), \Omega(n)\big)$-depth robust graph with $\mathsf{indeg} = O\big(\log n\big)$. Alwen *et al.* [ABP17] showed how to transform an n node (e, d)-depth robust graph with maximum indegree indeg to a $(e, d \times \mathsf{indeg})$-depth robust graph with maximum $\mathsf{indeg} = 2$ on $n \times \mathsf{indeg}$ nodes. Applying this transform to the Erdös, Graham and Szemerédi [EGS75] construction yields a constant-indegree graph on n nodes such that G is $\big(\Omega(n/\log(n)), \Omega(n)\big)$-depth robust—implying that $\Pi^{\parallel}_{cc}(G) = \Omega\big(\frac{n^2}{\log n}\big)$. From a theoretical standpoint, this is essentially optimal as any constant indeg DAG has $\Pi^{\parallel}_{cc} = O\big(\frac{n^2 \log\log n}{\log n}\big)$ [AB16]. From a practical standpoint it is important to understand the exact values of e and d for specific parameters n in each construction.

[9] The `3-PARTITION` problem is called P[3, 1] in [GJ75].

[10] Alwen *et al.* [ABP17] also gave tighter upper and lower bounds on $\Pi^{\parallel}_{cc}(G)$ for the Argon2i-A, iBH and Catena iMHFs. For example, $\Pi^{\parallel}_{cc}(G) = \Omega\left(n^{1.66}\right)$ and $\Pi^{\parallel}_{cc}(G) = O\left(n^{1.71}\right)$ for a random Argon2i-A DAG G (with high probability). Blocki and Zhou [BZ17] recently tightened the upper and lower bounds on Argon2i-B showing that $\Pi^{\parallel}_{cc}(G) = O\left(n^{1.767}\right)$ and $\Pi^{\parallel}_{cc}(G) = \tilde{\Omega}\left(n^{1.75}\right)$.

5.1 Results

We first produce a reduction from Vertex Cover which preserves approximation hardness. Let $\texttt{minREDUCIBLE}_d$ denote the problem of finding a minimum size $S \subseteq V$ such that $\texttt{depth}(G - S) \leq d$. Our reduction shows that, for each $0 \leq d \leq n^{1-\epsilon}$, it is NP-Hard to 1.3-approximate $\texttt{minREDUCIBLE}_d$ since it is NP-Hard 1.3-approximate Vertex Cover [DS05]. Similarly, it is hard to $(2-\epsilon)$-approximate $\texttt{minREDUCIBLE}_d$ for any fixed $\epsilon > 0$ [KR08], under the Unique Games Conjecture [Kho02]. We also produce a reduction from Cubic Vertex Cover to show $\texttt{REDUCIBLE}_d$ is NP-Complete even when the input graph has bounded indegree.

The techniques we use are similar to those of Bresar et al. [BKKS11] who considered the problem of finding a minimum size d-path cover in undirected graphs (i.e., find a small set $S \subseteq V$ of nodes such that every undirected path in $G - S$ has size at most d). However, we stress that if G is a DAG, $\hat{G}$ is the corresponding undirected graph and $S \subseteq V$ is given such that $\texttt{depth}(G - S) \leq d$ that this does not ensure that $\hat{G} - S$ contains no *undirected path* of length d. Thus, our reduction specifically address the needs for directed graphs and bounded indegree.

Theorem 5. $\texttt{REDUCIBLE}_d$ *is* NP-Complete *and it is* NP-Hard *to* 1.3*-approximate* $\texttt{minREDUCIBLE}_d$*. Under the Unique Games Conjecture, it is hard to* $(2-\epsilon)$*-approximate* $\texttt{minREDUCIBLE}_d$*.*

Theorem 6. *Even for* $\delta = O(1)$*,* $\texttt{REDUCIBLE}_{d,\delta}$ *is* NP-Complete*.*

We defer the proofs of Theorem 5 and Theorem 6 to the full version of the paper [BZ16]. We leave open the question of efficient approximation algorithms for $\texttt{minREDUCIBLE}_d$. Lee [Lee17] recently proposed a FPT $O(\log d)$-approximation algorithm for the related problem d-path cover problem running in time $2^{O(d^3 \log d)} n^{O(1)}$. However, it is not clear whether the techniques could be adapted to handle directed graphs and in most interesting cryptographic applications we have $d = \Omega(\sqrt{n})$ so the algorithm would not be tractable.

6 Conclusions

We initiate the study of the computational complexity of cumulative cost minimizing pebbling in the parallel black pebbling model. This problem is motivated by the urgent need to develop and analyze secure data-independent memory hard functions for password hashing. We show that it is NP-Hard to find a parallel black pebbling minimizing cumulative cost, and we provide evidence that the problem is hard to approximate. Thus, it seems unlikely that we will be able to develop tools to automate the task of a cryptanalyst to obtain strong upper/lower bounds on the security of a candidate iMHF. However, we cannot absolutely rule out the possibility that an efficient approximation algorithm exists. In fact, our results only establish worst case hardness of graph pebbling. We cannot rule out the existance of efficient algorithms to find optimal pebblings for practical

iMHF proposals such as Argon2i [BDK16] and DRSample [ABH17]. The primary remaining challenge is to either give an efficient α-approximation algorithm to find a pebbling $P \in \mathcal{P}^{\parallel}$ with $\mathsf{cc}(P) \leq \alpha \Pi^{\parallel}_{cc}(G)$ or show that $\Pi^{\parallel}_{cc}(G)$ is hard to approximate. We believe that the problem offers many interesting theoretical challenges and a solution could have very practical consequences for secure password hashing. It is our hope that this work encourages others in the TCS community to explore these questions.

Acknowledgements. We would like to thank Ioana Bercea and anonymous reviewers for helpful comments that improved the presentation of the paper. This work was supported by National Science Foundation Awards #1649515 and #1704587. The opinions expressed in this paper do not necessarily reflect those of the NSF.

References

[AB16] Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 241–271. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_9

[AB17] Alwen, J., Blocki, J.: Towards practical attacks on Argon2i and balloon hashing. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P, pp. 142–157 (2017)

[ABH17] Alwen, J., Blocki, J., Harsha, B.: Practical graphs for optimal side-channel resistant memory-hard functions. In: ACM CCS 17, pp. 1001–1017. ACM Press (2017)

[ABP17] Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 3–32. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_1

[ABW03] Abadi, M., Burrows, M., Wobber, T.: Moderately hard, memory-bound functions. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA (2003)

[AS15] Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, STOC 2015 (2015). http://eprint.iacr.org/2014/238

[AT17] Alwen, J., Tackmann, B.: Moderately hard functions: definition, instantiations, and applications. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 493–526. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_17

[BCS16] Boneh, D., Corrigan-Gibbs, H., Schechter, S.: Balloon hashing: a memory-hard function providing provable protection against sequential attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 220–248. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_8

[BDK15] Biryukov, A., Dinu, D., Khovratovich, D.: Fast and tradeoff-resilient memory-hard functions for cryptocurrencies and password hashing. Cryptology ePrint Archive, Report 2015/430 (2015). http://eprint.iacr.org/2015/430

[BDK16] Biryukov, A., Dinu, D., Khovratovich, D.: Argon2 password hash. Version 1.3 (2016). https://www.cryptolux.org/images/0/0d/Argon2.pdf

[BDKJ16] Biryukov, A., Dinu, D., Khovratovich, D., Josefsson, S.: The memory-hard Argon2 password hash and proof-of-work function. Internet-Draft draft-irtf-cfrg-argon2-00, Internet Engineering Task Force, March 2016

[BK15] Biryukov, A., Khovratovich, D.: Tradeoff cryptanalysis of memory-hard functions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 633–657. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_26

[BKKS11] Bresar, B., Kardos, F., Katrenic, J., Semanisin, G.: Minimum k-path vertex cover. Discrete Appl. Math. **159**(12), 1189–1195 (2011)

[BZ16] Blocki, J., Zhou, S.: On the computational complexity of minimal cumulative cost graph pebbling. CoRR, abs/1609.04449 (2016)

[BZ17] Blocki, J., Zhou, S.: On the depth-robustness and cumulative pebbling cost of Argon2i. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 445–465. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_15

[CGBS16] Corrigan-Gibbs, H., Boneh, D., Schechter, S.: Balloon hashing: provably space-hard hash functions with data-independent access patterns. Cryptology ePrint Archive, Report 2016/027, Version: 20160601:225540 (2016). http://eprint.iacr.org/

[Cha73] Chandra, A.K.: Efficient compilation of linear recursive programs. In: SWAT (FOCS), pp. 16–25. IEEE Computer Society (1973)

[Coo73] Cook, S.A.: An observation on time-storage trade off. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC 1973, pp. 29–33. ACM, New York (1973)

[Dem14] Demaine, E.: 6.890 algorithmic lower bounds: fun with hardness proofs, September 2014. http://ocw.mit.edu

[DFKP15] Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 585–605. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_29

[DGN03] Dwork, C., Goldberg, A., Naor, M.: On memory-bound functions for fighting spam. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 426–444. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_25

[DNW05] Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 37–54. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_3

[DS05] Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Ann. Math. **162**, 439–485 (2005)

[EGS75] Erdös, P., Graham, R.L., Szemeredi, E.: On sparse graphs with dense long paths. Comput. Math. Appl. **1**, 365–369 (1975)

[FLW13] Forler, C., Lucks, S., Wenzel, J.: Catena: a memory-consuming password scrambler. IACR Cryptology ePrint Archive 2013:525 (2013)

[GJ75] Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. **4**(4), 397–411 (1975)

[GLT80] Gilbert, J.R., Lengauer, T., Tarjan, R.E.: The pebbling problem is complete in polynomial space. SIAM J. Comput. **9**(3), 513–524 (1980)

[HP70] Hewitt, C.E., Paterson, M.S.: Record of the project MAC conference on concurrent systems and parallel computation (1970)

[HP10] Hertel, P., Pitassi, T.: The pspace-completeness of black-white pebbling. SIAM J. Comput. **39**(6), 2622–2682 (2010)
[Kal00] Kaliski, B.: Pkcs# 5: password-based cryptography specification version 2.0 (2000)
[Kho02] Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the Thirty-Fourth annual ACM Symposium on Theory of Computing, pp. 767–775. ACM (2002)
[KR08] Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2- ϵ. J. Comput. Syst. Sci. **74**(3), 335–349 (2008)
[Lee17] Lee, E.: Partitioning a graph into small pieces with applications to path transversal. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 1546–1558 (2017)
[MMV13] Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: Kleinberg, R.D. (ed.) ITCS 2013, pp. 373–388. ACM, January 2013
[NBF+15] Narayanan, A., Bonneau, J., Felten, E.W., Miller, A., Goldfeder, S.: Bitcoin and cryptocurrency technology (manuscript) (2015). Accessed 8 June 2015
[Per09] Percival, C.: Stronger key derivation via sequential memory-hard functions. In: BSDCan 2009 (2009)
[PHC] Password hashing competition. https://password-hashing.net/
[PR80] Paul, W.J., Reischuk, R.: On alternation II. A graph theoretic approach to determinism versus nondeterminism. Acta Inf. **14**, 391–403 (1980)
[RD16] Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 262–285. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_11
[RD17] Ren, L., Devadas, S.: Bandwidth hard functions for ASIC resistance. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 466–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_16
[Sch82] Schnitger, G.: A family of graphs with expensive depth reduction. Theor. Comput. Sci. **18**, 89–93 (1982)
[Sch83] Schnitger, G.: On depth-reduction and grates. In: 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7–9 November 1983, pp. 323–328. IEEE Computer Society (1983)
[SS78] Savage, J.E., Swamy, S.: Space-time trade-offs on the fft algorithm. IEEE Trans. Inf. Theory **24**(5), 563–568 (1978)
[SS79a] Savage, J.E., Swamy, S.: Space-time tradeoffs for oblivious integer multiplication. In: Maurer, H.A. (ed.) ICALP 1979. LNCS, vol. 71, pp. 498–504. Springer, Heidelberg (1979). https://doi.org/10.1007/3-540-09510-1_40
[SS79b] Swamy, S., Savage, J.E.: Space-time tradeoffs for linear recursion. In: Aho, A.V., Zilles, S.N., Rosen, B.K. (eds.) POPL, pp. 135–142. ACM Press (1979)
[Tom78] Tompa, M.: Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 196–204. ACM, New York (1978)

Anonymity in Distributed Systems

Pricing Anonymity

Daniel G. Arce[1(✉)] and Rainer Böhme[2,3]

[1] University of Texas at Dallas, Richardson, USA
darce@utdallas.edu
[2] University of Innsbruck, Innsbruck, Austria
rainer.boehme@uibk.ac.at
[3] Westfälische Wilhelms-Universität Münster, Münster, Germany

Abstract. In electronic anonymity markets a taker seeks a specified number of market makers in order to anonymize a transaction or activity. This process requires both coalition formation, in order to create an anonymity set among the taker and makers, and the derivation of the fee that the taker pays each maker. The process has a novel property in that the taker pays for anonymity but anonymity is created for both the taker and the makers. Using the Shapley value for nontransferable utility cooperative games, we characterize the formation of the anonymity set and the fee for any arbitrary number of makers selected by the taker.

Keywords: Anonymity · Markets · Privacy · Game theory
Cryptocurrencies

1 Introduction and Literature Review

Several advances in information technology have brought about that electronic transactions reveal identifying information of transaction partners. Specifically, packet-switched networks transmit addresses in every data packet to ensure delivery, digital signatures include identifying public keys for verification, and public ledger-based cryptocurrencies use references to – unique and therefore potentially identifying – past transactions for verification.

Sometimes identifiability is dysfunctional, and demand for anonymity arises out of private or public interest. However, establishing anonymity in systems that depend on identifying information requires effort. Technical solutions, known as *mixes*, bundle and shuffle the electronic records of similar activities (messages, transactions) from many participants so as to hide the relation between subjects and objects. Practical examples include the Tor network for Internet communication [11], mixes offering transaction anonymization in Bitcoin[1] [25], or the existence of dark pools next to conventional financial markets [40].

[1] The popular belief that Bitcoin payments are anonymous is wrong. This cryptocurrency uses pseudonymous accounts and a public transaction ledger. Agents who want to hide the relation between their accounts, some of which may fully identify them, need anonymizing technology [6].

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 349–368, 2018.
https://doi.org/10.1007/978-3-662-58387-6_19

Surprisingly little research studies the price of anonymity. Acquisti et al. [2] describe the economics of participating in (message) mixing services. They observe that anonymity is co-created by multiple agents sharing activities with the same observable features: one cannot be anonymous alone. More specifically, the authors consider decentralized anonymity infrastructures and suggest non-cooperative game theory to identify viable participation equilibria, but they do not offer a solution to their game. Acquisti and Varian [4] consider optional anonymization with simple technology ("delete cookie"), available without cost to some customers in their model, as a constraint to individual pricing. By contrast, Friedman and Resnick [14] study the effect of optional anonymity on the social level. Their repeated game with random matching predicts negative welfare effects. This is because bad reputation does not stick when agents can choose to be anonymous, resounding with the models of credit information sharing, which also contribute to the formal economic treatment of identity [29,30].

Other works, broadly related to anonymity and prices, focus on the payment system needed to compensate the operators of anonymizing infrastructure without revealing identifying information on the payment channel [5,13,17]; or empirically approximate users' willingness to pay for anonymous Internet access by measuring the tradeoff between the anonymity provided by a mixing service and the experienced performance [19].

The present work is inspired by Böhme and Möser's measurement study of JoinMarket[2] [23], a platform in the Bitcoin ecosystem that matches agents who seek to merge their payments in a single transaction in order to improve anonymity. Such transactions are called CoinJoins in jargon [21] and, to offer some anonymity, they must entail payments of the same amount at the same time.[3] JoinMarket is organized as platform where supply-side agents, called *makers*, offer funds to participate in CoinJoin transactions for an advertised mixing fee.[4] Demand-side agents, called *takers*, initiate an anonymizing transactions by choosing several of these offers. As a result, a typical transaction from this market is funded by exactly one taker and two or more makers. The matching and settlement is supported with software provided by the JoinMarket developers and run in a decentralized manner on many Internet nodes. Möser and Böhme [24] speculate why demand and supply might exist in this market, but acknowledge that "puzzles" remain. They do not formally characterize the relation between key parameters, such as the level of anonymity provided and its price (i. e., the fees paid to form the CoinJoin).

Here, to the best of our knowledge, we provide the first formal solution to price anonymity in systems which require the coordination of multiple participants. While we adopt the terminology of transaction anonymization used in CoinJoins, our results generalize to all anonymization schemes with similar prop-

[2] See http://joinmarket.io. Last visited on June 25th, 2018.

[3] See Meiklejohn and Orlandi [22] on the hardness of untangling CoinJoin transactions.

[4] The fee is composed of fixed and variable parts to account for contributions to the Bitcoin network's miner fees. Our model abstracts from this complexity by assuming a normalized nominal transaction value.

erties. As one cannot be anonymous alone, anonymity requires an agreement among individuals to behave in an indistinguisable way; e.g. via common message length or transaction amount. This results in the formation of an *anonymity set*, being the collection of individuals that nonmembers cannot distinguish between. The degree of anonymity is generally associated with the size of the anonymity set [32]. An anonymity set therefore requires coalition formation, and the traditional game-theoretic approach to coalition formation is cooperative game theory. Consequently, at its most basic level, a CoinJoin transaction requires the formation of an anonymity set/coalition organized around a common transaction amount. In addition, the formation of the anonymity set/coalitions is enforceable because a Bitcon CoinJoin agreement occurs only if all members of the anonymity set sign the transaction [21]. This is again consistent with cooperative game theory.

In a cooperative game representation of a CoinJoin, a characteristic function is specified for all potential coalitions in the transaction. For each coaltion, the characteristic function is a vector of payoffs for each member of that coalition, where payoffs are defined in terms of the size of the anonymity set specified by the taker, each player's valuation of their identity, the fee paid by a taker to get enough makers to join the anonymity set, the fee received by makers for joining the anonymity set, etc. What matters is that this approach captures the key property of anonymity markets, which is that some participants pay for anonymity but all participants benefit from anonymity being created. This is also novel from the perspective of cooperative game theory in that one player (the taker) is *paying* other players (the makers) to form a coalition from which *all* members benefit. Consequently, the characteristic function is defined for the anonymity set (the grand coalition) and all potential subcoalitions of the anonymity set, where the anonymity fee is an unknown to be determined endogenously as a function of the solution concept used to solve the game.

In this paper we use the Shapley value to solve the cooperative game. The Shapley value an economically-motivated solution that gives each player their expected marginal contribution to the anonymity set and all possible subcoalitions of the anonymity set. In particular, we derive expressions for the Shapley value of the market participant demanding anonymity (the taker) and any number of suppliers (the market makers) participating in a CoinJoin. This in turn allows for a characterization of the price of anonymity. The class of anonymization schemes to which our theory applies can be further expanded by adapting the characteristic function to the anonymization scheme and attacker model.

Our work is distinct from a line of formal research on privacy quantification with respect to *attribute* disclosure. For example, differential privacy offers a framework to measure and account the privacy loss when querying private databases interactively [12]. Game theory, also in its cooperative form [9,18], has been applied in this subfield in order to establish the price of attribute values as a function of their precision, or to incentivize disclosure [8,15]. Acquisti et al. [3] survey the economics of privacy more broadly.

This paper is organized as follows. Section 2 specifies anonymity markets as a cooperative game. Section 3 presents the Shapley value as the solution concept

for the game. Section 4 solves the game for the case of three players. Section 5 generalizes to N players, and Sect. 6 concludes.

2 Anonymity Markets as Cooperative Games

In this section we introduce the building blocks for specifying an anonymity market as a cooperative game. We consider anonymity markets with two types of participants: *takers* and (market) *makers* [24]. Makers offer their identities or pseudoidentities (such as Bitcoin addresses) for use in a transaction or activity that the taker seeks to engage in anonymously. Recall from the introduction that existing markets match exactly one taker with two or more makers to form a transaction. All makers are assumed to be honest in that they do not seek to ascertain the taker's identity for their own purposes. Hence, the returns of interest to a maker in an anonymity transaction are the fee that the maker receives from the taker and the anonymity that the maker receives as well in the transaction. This is the peculiarity of anonymity markets: the taker pays for anonymity but the transaction itself anonymizes the identities of the taker and makers alike. In this sense anonymity is akin to a public good that only the taker pays for.[5]

Following Pfitzmann and Köhntopp [32, p. 2], anonymity can be defined as: "the state of being not identifiable within a set of subjects, the anonymity set."[6] The purpose of an anonymity market is to create an anonymity set, S, which is a coalition consisting of a taker and makers. As anonymity is meant to preserve identities, the term D will denote the taker's value of its identity. All makers will be assumed to value their identity identically, with d denoting a maker's valuation of its identity. The fee that a taker pays to each maker in an anonymity set is denoted as f (to be determined endogenously). A cooperative game approach to anonymity is appropriate because the focus is on the distribution of benefits among the taker and makers when they form an anonymity set.

In a CoinJoin, the probability that a player remains anonymous (retains their identity) against a global passive adversary (GPA) in an anonymity set/coalition of size $|S|$ is $(|S|-1)/|S|$, as all $|S|$ members of the anonymity set are indistinguishable to the GPA owing to the common transaction amount and use of different input and output addresses in the transaction. In other words, the probability that the GPA randomly guesses the identity of a member of anonymity set S is $1/|S|$. Hence, participating in an anonymity market involves some risk

[5] Public goods have the property that they are *nonexclusive* and *nonrival* [34]. Nonexclusive means that once created, the associated benefits of the good cannot be withheld from others. Technically, the nonexcludability property of anonymity applies only to the makers and taker engaged in the transaction. Nonrivalry means that use of the good does not prohibit its use by others.

[6] This definition is compatible with common alternatives. For example, the size of the anonymity set corresponds to the parameter k in the k-anonymity model [39]. Entropy-based anonymity metrics generalize to sets with non-uniform priors [10,35].

of loss of identity to each maker, and the fee we derive that is paid by the taker to each maker must compensate makers for this risk.[7]

In addition, it is assumed that anonymity is only created within coalitions involving a taker. Intuitively, potential makers do not costlessly create anonymity amongst themselves because they have no underlying transaction or message that requires anonymization, and forming an anonymity set always carries some risk. Instead, makers rely on a taker (who does have a transaction or message needing anonymization) to compensate them for creating anonymity for both the taker and themselves. This implies that the payoffs for makers in coalitions that do not involve a taker are normalized to zero. Normalizing in this way facilitates an emphasis on taker identity, maker identity and the public nature of anonymity in determining the fee for anonymity, which is the focus of the paper.

The specifics of the transaction to be anonymized (e.g., amount, message length, or time) precludes takers from matching with other takers. As Narayanan et al. [27] observe, a one-taker transaction avoids the necessity for multiple takers to agree on transaction specifics, which is inefficient and costly. Hence, there is no requirement for coincidence of transaction specifics among takers. Only makers customize the transaction to the specific needs of a taker so as to make their activities indistinguishable in the anonymity set. The incentive for makers to form a coalition with the taker is that they are compensated for facilitating anonymity by meeting the taker's transaction needs. This compensation takes the form of a fee, f, paid by the taker *and* the anonymity received by the maker when participating in the CoinJoin.

The taker's alternative is known as a *mix*, and is addressed within the model as the taker's outside option. Specifically, instead of an anonymity market, such as JoinMarket, a taker could go to an outside option (e.g., a mix) and pay a fee, F, for anonymity. Anonymity markets are almost instantaneous transactions for the taker whereas the outside option may involve a significant delay and risk. Mixing services in the Bitcoin ecosystem reportedly stole their clients' funds, a threat that can be mitigated with CoinJoin transactions arranged on anonymity markets [25]. Consequently, the reservation value of anonymity for the taker of going to the outside option for anonymity is δD where $\delta \in (0, 1)$ is a function of both the probability that the funds are transmitted as intended while anonymity is preserved by this outside option, and the taker's time preference (discount factor). As both probabilities and discount factors lie within the (0, 1) interval, $\delta \in (0, 1)$ by definition. By comparison, in a CoinJoin with an anonymity set of size $|S|$ the taker retains its anonymity with probability $(|S| - 1)/|S|$, leading to an expected value of anonymity of $(|S| - 1)/|S| \times D$. Lastly, the mixing fee, F, is posted, whereas the CoionJoin fee, f, is to be determined via the equilibrium process associated with the CoinJoin.

Finally, anonymity is a public good but it need not be valued identically among the anonymity market participants; hence, the value D for the taker's

[7] In addition to the first-order risk of losing one's identity, makers may also face the risk of legal authorities investigating Bitcoin purchases as part of a criminal investigation. This potentiality lies beyond the scope of the present paper.

identity and d for each maker's identity. Moreover, interpersonal comparisons of anonymity are not possible. For example, the taker's value of its identity need not be expressed in terms of the same measure of value as the maker's. Another rationale for nontransferable utility is that it is likely that takers and makers have differing time preferences [24]. Even if the valuation of anonymity was comparable between players, practical anonymity markets are restricted in arranging transfer payments. For example with Bitcoin and JoinMarket, transfer payments are either not enforceable (because alternative payments cannot be part of the same atomic transaction), or compromise the anonymity of the transaction (because the GPA may infer identity information from observing the transfer payments). In this model the transfer payment is the fee for anonymity, which is the same for every maker. Enforceable sidepayments beyond this constant fee would potentially compromise anonymity.

This implies that what is attainable by an anonymity set associated with a coalition of taker and makers cannot be assigned a single real number, as is the case in cooperative games with transferable utility, known as *TU games* [33]. More-to-the point, the assumption of transferable utility would imply that identity can be expressed in the same units of measurement for every member of the anonymity set and that it is possible to distribute the value that each member places on their identity across the membership of the anonymity set in a meaningful way. We do not believe that identity has such properties. Following Shapley [37,38],"Interpersonal comparability of utility is generally regarded as an unsound basis on which to erect theories of multipersonal behavior." For this reason, much of noncooperative game theory steers clear of the transferable utility assumption. For similar reasons we use cooperative games with nontransferable utility, known as *NTU games*. NTU games are a more generalized version of cooperative games, as transferable utility is a restrictive assumption. Indeed, any TU game can be expressed as an NTU game. We therefore turn to the formal definition of an NTU game and the Shapley value solution to NTU games.

3 NTU Games and the Shapley Value

In a NTU game, for any non-empty coalition, S, the associated *NTU characteristic function*, $V(S)$, denotes the set of feasible utility vectors attainable by that coalition. Specifically, $V(S) \subseteq \mathbb{R}^{|S|}$, where $|S|$ is the cardinality of S. For each vector $\boldsymbol{x} \in V(S)$ the entry x_i specifies the maximum payoff to player i should player i be a member of that coalition.[8] Characteristic function $V(S)$ is the vector of utilities that is feasible for the members of S when they cooperate with each other. As described, an anonymity market is a cooperative game with sidepayments but without transferable utility [31].

An NTU game is defined by a pair (N, V), where N is the set of all players, and $V(S)$ is the characteristic function specifying the payoff to each member $i \in S$ for all coalitions $S \subseteq N$. Given the game (N, V) our approach uses the

[8] Technically, any $y_i \leq x_i$ is a potential payoff for player i as well. This property is known as "comprehensiveness" [28].

Shapley value [36] to derive the anonymity fee, f. The Shapley value allocates the total net benefits of an anonymity transaction according to each player's marginal contribution to every subcoalition of the anonymity set that the player can potentially be a member of.

The Shapley value was originally defined for TU cooperative games. Later, Shapley [37,38] established that one can create a TU game associated with an NTU game by creating a *fictitious transfer game* (a λ-transfer game), which converts the NTU game into a TU game that can be solved via Shapley's original method. The steps associated with this procedure can be summarized as follows. First, begin with a nonnegative set of weights for each of the players, $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_{|N|})$. Given these weights, find the maximum sum of the $\boldsymbol{\lambda}$-weighted utility for each vector $\boldsymbol{x} \in V(S)$. Second, these maximal sums essentially define a TU game (the fictitious transfer game) for which a Shapley value can be derived. Solve for the Shapley value. Denote as $\varphi_i(\omega, \boldsymbol{\lambda})$ the allocation received by player i in the Shapley value, where $\omega(S)$ is the characteristic function for the fictitious transfer game. Third, verify that the vector $[\varphi_i(\omega, \boldsymbol{\lambda})/\lambda_i]_{i \in N}$ is feasible for the grand coalition in the NTU game; i. e., $[\varphi_i(\omega, \boldsymbol{\lambda})/\lambda_i]_{i \in N} \in V(N)$. If feasible, then $\varphi_i(\omega, \boldsymbol{\lambda})/\lambda_i$ is the Shapley value allocation for player i in the NTU game.

Given this synopsis we now specify the formal procedure. First, create a fictitious transfer game by specifying a vector $\boldsymbol{\lambda} \in (\mathbb{R}^+)^{|N|}$. For each coalition, S, the function $\omega(S)$ is called the *worth function* (the characteristic function of the fictitious transfer game), where

$$\omega(S) = \max_{\boldsymbol{x} \in V(S)} \sum_{i \in S} \lambda_i x_i. \tag{1}$$

The λ_i/λ_j ratios can be considered as exchange rates between the nontransferable utilities of the players. As a simple example, if a taker measures its identity in terms of US$ and makers in terms of euros, €, then an exchange rate, $\lambda_€/\lambda_\$$, is needed to relate D to d, and $\lambda_\$/\lambda_€$ is needed to relate d to D.[9]

Second, the worth functions, $\omega(S)$, can be regarded as characteristic functions for a TU game derived from the NTU game. The *Shapley value for the associated TU game* is

$$\varphi_i(\omega, \boldsymbol{\lambda}) = \sum_{\substack{i \in S, \\ S \subseteq N}} \frac{(|S|-1)!(|N|-|S|)!}{|N|!} \cdot \left(\omega(S) - \omega(S \setminus \{i\})\right). \tag{2}$$

For any coalition, S, that i is a member of, the term $\omega(S) - \omega(S \setminus \{i\})$ in Eq. (2) measures i's *marginal contribution* to coalition S. That is, the difference $\omega(S) - \omega(S \setminus \{i\})$ is what the coalition can achieve with i as a member less what it

[9] Myerson [26, p. 16] offers an alternative interpretation: "With nontransferable utility, we have no grounds for interpersonal comparison of utility, so we may feel free to rescale either player's utility separately by a positive scaling factor or utility weight λ_i. Now, in the rescaled version of the game, pretend that the weighted-utility payoffs are transferable."

achieves without i. The Shapley value is therefore the expected value of a player's marginal contribution over all potential $|N|!$ orderings of the players in the game. The coefficient on the marginal contribution of i in Eq. (2) is the probability that a particular coalition with i as a member occurs, assuming that all $|N|!$ orderings are equally likely. In this way, the Shapley value allocates each player their marginal contribution averaged over all possible orderings (permutations) of the players. For any TU game the Shapley value exists and has the following properties, among other characteristics [36]: (i) uniqueness, (ii) symmetry: any two players that are treated identically by characteristic function $\omega(\cdot)$ have equal Shapley value allocations, and (iii) (Pareto) efficiency, the gains of the grand coalition must be fully distributed:

$$\sum_{i \in N} \varphi_i = \omega(N). \tag{3}$$

These properties make the Shapley value the predominant solution concept for cooperative games. As a reminder, the Shapley value is also individually rational.

Third, $[\varphi_i(\omega, \boldsymbol{\lambda})/\lambda_i]_{i \in N}$ is the *Shapley value (λ-transfer value) for the NTU game* if it is feasible for the grand coalition in the NTU game. This is the case when $[\varphi_i(\omega, \boldsymbol{\lambda})/\lambda_i]_{i \in N} \in V(N)$. If it is not feasible, then the procedure must be redone for another vector $\boldsymbol{\lambda'} \neq \boldsymbol{\lambda}$ until a solution is found. Shapley [37,38] establishes that such a solution exists.[10] Most importantly, in establishing feasibility we endogenously derive the fee, f, paid by the taker to each maker in the anonymity set. An example is given in the following section.

4 A Three-Player Anonymity Market

In a 3-player anonymity market the taker specifies that it desires two makers in the associated anonymity set. Let player t be the taker and players 1 and 2 be the makers. For single-player coalitions the NTU characteristic functions, which specify the vector of maximum utilities achievable by each member of a coalition when that coalition is formed, are

$$V(\{t\}) = \{x_t \mid x_t \leq \delta D - F\}, \tag{4}$$

this reflects the outside option for the taker (the mix); and

$$V(\{i\}) = \{x_i \mid x_i \leq 0 : i = 1, 2\} \text{ for makers 1 and 2.} \tag{5}$$

Makers require a taker for an anonymity market to form. Without a taker, a maker's utility is normalized to zero.

[10] As the proof is based on a fixed point theorem it does not guarantee uniqueness. We are unaware of any example in the literature where multiple weights are derived that lead to alternative NTU Shapley values. If multiple fixed points exist, selecting among them is a well-defined problem. A natural criterion would be to maximize the taker's payoff.

For 2-player coalitions, in a coalition of $\{t,1\}$ or $\{t,2\}$ each player remains anonymous with probability $1/2$. It is the existence of these intermediate coalitions that separates this analysis from 3-player bargaining. Consequently,

$$V(\{t,1\}) = \{(x_t, x_1) \mid x_t \leq 1/2D - f, x_1 \leq 1/2d + f\}; \tag{6}$$
$$V(\{t,2\}) = \{(x_t, x_2) \mid x_t \leq 1/2D - f, x_2 \leq 1/2d + f\}. \tag{7}$$

Notice how (i) anonymity is akin to a public good that is (probabilistically) produced when an anonymity set/coalition is formed, and (ii) only the taker is paying for anonymity. Consequently, no anonymity is produced in a $\{1,2\}$ maker-only coalition because neither maker pays the other to create anonymity. To wit, makers are direct suppliers of anonymity to the taker. As a byproduct makers supply anonymity to each other. Makers do not contract directly for anonymity amongst themselves:

$$V(\{1,2\}) = \{(x_1, x_2) \mid x_1 \leq 0, x_2 \leq 0\}. \tag{8}$$

Finally, the probability of retaining one's identity is increased to $2/3$ in the grand coalition, $N = \{t,1,2\}$:

$$V(N) = \{(x_t, x_1, x_2) \mid \ x_t \leq 2/3D - 2f, x_1 \leq 2/3d + f, x_2 \leq 2/3d + f\}. \tag{9}$$

Note that in all of these specifications, the anonymity fee in the NTU game is not taken as given, but is an unknown to be solved for.

Following the procedure outlined above, the solution is derived by following these three steps. First, set $\boldsymbol{\lambda} = (\lambda_t, \lambda_1, \lambda_2) = (1,1,1)$.[11] Second, create the worth functions that are consistent with this $\boldsymbol{\lambda}$ and solve for the Shapley value of the TU game. Third, demonstrate feasibility of the NTU solution for this $\boldsymbol{\lambda}$.

The associated worth functions, which can be regarded as characteristic functions for the TU game derived from the NTU game, are constructed via Eq. (1). Therefore the worth functions are

$$\omega(\{t\}) = \delta D - F; \tag{10}$$
$$\omega(\{1\}) = \omega(\{2\}) = 0; \tag{11}$$
$$\omega(\{t,1\}) = 1/2D + 1/2d; \tag{12}$$
$$\omega(\{t,2\}) = 1/2D + 1/2d; \tag{13}$$
$$\omega(\{1,2\}) = 0; \tag{14}$$
$$\omega(N) = 2/3D + 4/3d. \tag{15}$$

Once the worth functions have been derived, the result is a TU game. The Shapley value is calculated according to the formula in Eq. (2). The Shapley values for this TU game are (derivation in Appendix A),

$$\varphi_t(\omega, \boldsymbol{\lambda}) = \frac{14}{36}D + \frac{22}{36}d + \frac{1}{3}(\delta D - F); \text{ and} \tag{16}$$
$$\varphi_1(\omega, \boldsymbol{\lambda}) = \varphi_2(\omega, \boldsymbol{\lambda}) = \frac{5}{36}D + \frac{13}{36}d - \frac{1}{6}(\delta D - F). \tag{17}$$

[11] This is consistent with finding a solution under the condition $\lambda_t = \lambda_1 = \lambda_2$ (where all λ's are finite), which yields an equivalent result.

To establish the NTU Shapley values, feasibility requires:

$$(\varphi_t(\omega, \boldsymbol{\lambda})/\lambda_t, \varphi_1(\omega, \boldsymbol{\lambda})/\lambda_1, \varphi_2(\omega, \boldsymbol{\lambda})/\lambda_2) \in V(N). \tag{18}$$

Recall that $(\lambda_t, \lambda_1, \lambda_2) = (1, 1, 1)$; hence, feasibility for the taker requires that $\varphi_t(\omega, \boldsymbol{\lambda}) \leq {}^2\!/_3 D - 2f$; i. e.,

$$\frac{14}{36}D + \frac{22}{36}d + \frac{1}{3}(\delta D - F) \leq \frac{2}{3}D - 2f. \tag{19}$$

Solving for f yields the following result.

Result 1. *The fee associated with Shapley value of a 3-player (one taker, two makers) anonymity set is*

$$f = \frac{5}{36}D - \frac{11}{36}d - \frac{1}{6}(\delta D - F). \tag{20}$$

Proof. What remains is to show that the solution is feasible for the makers. Given the definition of $V(N)$, feasibility also requires that $\varphi_1(\omega, \boldsymbol{\lambda}) \leq {}^2\!/_3 d + f$ and $\varphi_2(\omega, \boldsymbol{\lambda}) \leq {}^2\!/_3 d + f$. Substituting in the value for f in Eq. (20), ${}^2\!/_3 d + f = \frac{5}{36}D + \frac{13}{36}d - \frac{1}{6}(\delta D - F)$, which according to Eq. (17) is the solution for both $\varphi_1(\omega, \boldsymbol{\lambda})$ and $\varphi_2(\omega, \boldsymbol{\lambda})$.

This result establishes that the Shapley value can be used to characterize the fees for anonymity as a function of the taker's subjective identity valuation (D), the makers' subjective identity valuation (d) and the outside alternative (δ and F). Several novel observations emerge from this result. First, it cannot be the case that anonymity/identity is symmetrically valued across takers and makers ($D = d$) because then $f < 0$.[12] This would require makers to pay the taker. As such an arrangement is never observed, it must be the case that $D \gg d$. Second, the maker fee, f, is increasing in the outside fee, F, but only by a factor of one-sixth. Third, one can re-write the fee in (20) as

$$f = \frac{5 - 6\delta}{36}D - \frac{11}{36}d + \frac{1}{6}F, \tag{21}$$

in which case it is clear that non-fee based characteristics of the outside option, captured by δ, contribute significantly to determining the fee in a one taker, two maker market. Recall that δ is a function of both the taker's time preferences and the size of the anonymity set generated by the outside option (mix). In particular, Möser and Böhme [24] posit that takers' time preference cause takers to pay a premium for immediate anonymity services. Such a low discount factor implies a low value of δ, perhaps approaching zero.

[12] The term $\delta D - F$ must be nonnegative; otherwise, the outside alternative is not viable for the taker.

5 The Price of Anonymity

Now we derive the anonymity fee for an arbitrary number of makers, m. As the makers are assumed to be identical, what matters in expressing the NTU characteristic function for a coalition is the number of makers involved. Let:

- m be the total number of makers that the taker seeks in the anonymity set;
- $n = 1, 2, \ldots, m$ be the number of makers in a coalition;
- $\{t, n\}$ is a coalition with the taker, t, and n makers;
- $\{t\}$ is a coalition where the taker instead uses the outside option (mix);
- $\{n\}$ is a coalition with n makers and no taker.

Then the NTU characteristic functions for the game are

$$V(\{t\}) = \{x_t \mid x_t \leq \delta D - F\}; \tag{22}$$

$$V(\{n\}) = \{(x_i) \mid x_i \leq 0 \ \forall i : 1 \leq i \leq n\}; \tag{23}$$

$$V(\{t,n\}) = \left\{(x_t, (x_i)) \mid x_t \leq \frac{n}{n+1}D - nf; x_i \leq \frac{n}{n+1}d + f \ \ \forall i : 1 \leq i \leq n\right\}. \tag{24}$$

The worth functions for a coaltition, S, in the λ-transfer game are derived by setting $\lambda_i = 1$ for all $i \in S$ and applying Eq. (1):

$$\omega(\{t\}) = \delta D - F; \tag{25}$$

$$\omega(\{n\}) = 0 \ \forall n : 1 \leq n \leq m; \tag{26}$$

$$\omega(\{t,n\}) = \frac{n}{n+1}D + \frac{n^2}{n+1}d. \tag{27}$$

Theorem 2. *The Shapley values for an anonymity set with one taker, t, and m makers are*

$$\frac{\varphi_t(\omega, \boldsymbol{\lambda})}{\lambda_t} = \frac{\delta D - F}{m+1} + \frac{D}{m+1}\sum_{n=1}^{m}\frac{n}{n+1} + \frac{d}{m+1}\sum_{n=1}^{m}\frac{n^2}{n+1}\ ; \textit{and} \tag{28}$$

$$\frac{\varphi_i(\omega, \boldsymbol{\lambda})}{\lambda_i} = \frac{D}{m\cdot(m+1)}\sum_{n=1}^{m}\frac{1}{n+1} + \frac{d}{m\cdot(m+1)}\sum_{n=1}^{m}\frac{n^2+n-1}{n+1} - \frac{\delta D - F}{m\cdot(m+1)}, \tag{29}$$

for all makers $i \in \{1, \ldots, m\}$.

See Appendix B for the proof. An alternative representation using harmonic numbers instead of finite sums is given in Appendix C.

The associated anonymity fee is derived from the requirement that φ_t/λ_t must be feasible for $V(\{t, m\})$; i. e., $\varphi_t/\lambda_t \leq \frac{m}{m+1}D - mf$. From Eq. (28) and given $\lambda_t = 1$, this becomes

$$\frac{\varphi_t}{\lambda_t} = \frac{\delta D - F}{m+1} + \frac{D}{m+1}\sum_{n=1}^{m}\frac{n}{n+1} + \frac{d}{m+1}\sum_{n=1}^{m}\frac{n^2}{n+1} \leq \frac{m}{m+1}D - mf. \tag{30}$$

Setting the two sides of the inequality equal to each other and solving for f yields the following characterization.

Corollary 1. *The fee associated with the Shapley value for an anonymity set with $m \geq 1$ makers is*

$$f = \frac{D}{m+1} - \frac{D}{m \cdot (m+1)} \sum_{n=1}^{m} \frac{n}{n+1} - \frac{d}{m \cdot (m+1)} \sum_{n=1}^{m} \frac{n^2}{n+1} - \frac{\delta D - F}{m \cdot (m+1)}. \tag{31}$$

Once again, the anonymity fee, f, is increasing in the outside fee, F. Yet the increase in f due to F is decreasing in the number of makers, m. Specifically, f is increasing in F by a factor of $1/(m(m+1))$. The anonymity fee is also increasing in the taker identity, D, and decreasing in the makers' identity, d. In particular, the pricing of anonymity has remained a puzzle because the production of anonymity generates a positive externality in that all agents who supply anonymity also receive it as a good [24]. We provide the first characterization of the relation between f and d.

Regarding computational aspects, observe that the calculation of the Shapley value and its associated fee requires linear time in m at fixed precision, and polynomial time at arbitrary precision. Hence, our solution offers an efficient algorithm to determine the price of anonymity.

6 Conclusion

We have specified a cooperative game that captures the features of anonymity markets known as CoinJoins. More generally, our model captures the fact that in anonymity markets it is often the case that one demand-side participant ("taker") pays for anonymity, but all participants of a trade, including $m > 1$ "makers" on the supply side, receive anonymity if the trade happens. This is novel from a game-theoretic perspective as well because one member of a coalition is paying a fee to all other members to form the coalition even though *all* members ultimately benefit from the resulting coalition. Using the Shapley value as solution concept, we have derived the price of anonymity endogenously as a function of the taker's and makers' valuation of their identities, as well as the price and quality of an outside option for the taker. Of particular note is that we are able to characterize the way in which the associated positive externality received by makers (anonymity) affects the fee paid by the taker. The model is general enough to inform the design of all anonymity schemes that create anonymity by coordinating observable activities in order to make them look alike. These include anonymous communication systems and their applications in electronic voting and privacy-enhancing middleware, cryptocurrency transaction systems, and possibly the organization of dark pools in finance.

The model establishes a broad canvas for follow-up work. An immediate example is that alternative cooperative solution concepts exist for NTU games; most notably, the core, and the NTU values introduced by Harsanyi [16] and

Maschler-Owen [20]. The Shapley value is utilitarian in that it maximizes the weighted sum of the individual payoffs in *each* coalition. This is only the case for the grand coalition in the Harsanyi value. Instead, the Harsanyi value is equitable in that weighted net utility gains are equal for each individual in a coalition. By investigating these alternative solutions one may gather whether CoinJoins may be able to compete on different coalitional ethics. Our results can also be experimentally tested via behavioral techniques in which subjects (a taker and makers) are endowed with values for their respective identities. One could then see how fees vary with alternative magnitudes of identity values and also the amount of anonymity provided. In addition, the question of whether the fee varies with the external fee for anonymity mixes can be investigated both experimentally and via comparisons of the prices in CoinJoins and mixes.

In its present form, the game is one-shot. It does not capture the reported practice of repeated anonymization [25], which could increase anonymity as the cardinality of the joint anonymity set grows. Hence, a direction for future work is to consider alternative payoff functions than the global passive adversary (GPA) used here, who guesses exactly once with uniform probability. For example, a straightforward extension is to model varying risk appetite of takers by adjusting the NTU characteristic function. The problem gets substantially more complicated if non-identical and potentially adversarial makers are considered. Möser and Böhme [24] speculate that attackers could try to actively participate in CoinJoin transactions, possibly with multiple identities, in order to extract information about the composition of the anonymity set and eventually de-anonymize the taker. Such attackers could offer their enticing services at subsidized fees, below the Shapley value, in order to increase their odds of being selected. This scenario clearly requires an analysis based on characteristic functions that are derived from an underlying noncooperative game. The same applies to situations where takers choose m out of a large number of competing makers. Let us emphasize again that, although we give solutions for arbitrary m, the present theory does not lend itself to interpretations where m is endogenous.

Other directions of potential interest are to consider (opportunity) costs of engaging in anonymous transactions; to endogenize the quality of the outside option, δ, by modeling the behavior of the mix operator under incentive regimes as suggested by Bonneau et al. [7]; to consider transactions with coins of different quality (such as due to taint or blacklisting) as proposed by Abramova et al. [1]; to relax the strict dichotomy between taker and makers and replace it with heterogenous agents in some preference space. Finally, the mechanism design required to elicit the fair price of anonymity derived here is up to future work. JoinMarket, the platform that inspired this line of research, seems to employ ad-hoc mechanisms, as witnessed by many changes in the course of its history. And there seems to be room for improvement on the mechanism as well as need for a more principled approach towards constructing anonymity markets.

Acknowledgement. We thank the anonymous reviewers for helpful comments. The second author is funded in part by Archimedes Privatstiftung, Innsbruck, and the Ger-

man Bundesministerium für Bildung und Forschung (BMBF) under grant agreement 16KIS0382.

Appendix

A Shapley Value Derivation for the 3-Player Example

Following the formula given in Eq. (2), the Shapley value for the taker, t, with the makers as players 1 and 2, is

$$\begin{aligned}\varphi_t(\omega,\boldsymbol{\lambda}) =& \frac{1}{3}\big(\omega(N)-\omega(\{1,2\})\big)+ && \text{(grand coalition)}\\ & \frac{1}{6}\big(\omega(\{t,1\})-\omega(\{1\})\big)+ && \text{(taker \& maker 1)}\\ & \frac{1}{6}\big(\omega(\{t,2\})-\omega(\{2\})\big)+ && \text{(taker \& maker 2)}\\ & \frac{1}{3}\big(\omega(\{t\})-\omega(\emptyset)\big). && \text{(taker alone)}\end{aligned} \tag{32}$$

Substituting in the worth function values, Eqs. 10–15,
(by convention, $\omega(\emptyset)=0$):

$$\varphi_t(\omega,\boldsymbol{\lambda}) = \frac{1}{3}\left(\frac{2}{3}D+\frac{4}{3}d\right)+\frac{1}{6}\left(\frac{1}{2}D+\frac{1}{2}d\right)+\frac{1}{6}\left(\frac{1}{2}D+\frac{1}{2}d\right)+\frac{1}{3}\left(\delta D-F\right). \tag{33}$$

Aggregating terms,

$$\varphi_t(\omega,\boldsymbol{\lambda}) = \left(\frac{2}{9}+\frac{1}{12}+\frac{1}{12}\right)D+\left(\frac{4}{9}+\frac{1}{12}+\frac{1}{12}\right)d+\frac{1}{3}\left(\delta D-F\right) \tag{34}$$

and simplifying:

$$= \frac{14}{36}D+\frac{22}{36}d+\frac{1}{3}\left(\delta D-F\right). \tag{35}$$

Using Eq. (2) to calculate the Shapley value for player 1, who is a maker:

$$\begin{aligned}\varphi_1(\omega,\boldsymbol{\lambda}) =& \frac{1}{3}\big(\omega(N)-\omega(\{t,2\})\big)+ && \text{(grand coalition)}\\ & \frac{1}{6}\big(\omega(\{t,1\})-\omega(\{t\})\big)+ && \text{(taker \& maker 1)}\\ & \frac{1}{6}\big(\omega(\{1,2\})-\omega(\{2\})\big)+ && \text{(both makers)}\\ & \frac{1}{3}\big(\omega(\{1\})-\omega(\emptyset)\big). && \text{(maker 1 alone)}\end{aligned} \tag{36}$$

Substituting in the worth function values:

$$\varphi_1(\omega, \boldsymbol{\lambda}) = \frac{1}{3}\left(\frac{2}{3}D + \frac{4}{3}d - \frac{1}{2}D - \frac{1}{2}d\right) + \frac{1}{6}\left(\frac{1}{2}D + \frac{1}{2}d - (\delta D - F)\right). \quad (37)$$

Aggregating terms and simplifying:

$$\varphi_1(\omega, \boldsymbol{\lambda}) = \left(\frac{2}{9} - \frac{1}{6} + \frac{1}{12}\right)D + \left(\frac{4}{9} - \frac{1}{6} + \frac{1}{12}\right)d - \frac{1}{6}(\delta D - F) \quad (38)$$

$$= \frac{5}{36}D + \frac{13}{36}d - \frac{1}{6}(\delta D - F). \quad (39)$$

By the symmetry property of the Shapley value, $\varphi_2(\omega, \boldsymbol{\lambda}) = \varphi_1(\omega, \boldsymbol{\lambda})$. □

B Proof of Theorem 2

The proof consists of three parts.

B.1 Shapley Value for the Taker

Proof. From Eq. (2), the coefficient on $\omega(\{t\}) - \omega(\emptyset) = \delta D - F$ in the Shapley value is $1/N = 1/(m+1)$. This is the first term in Eq. (28).

Given m makers, there are $\binom{m}{n} = \frac{m!}{n!\,(m-n)!}$ combinations of coalitions that can be expressed as $S = \{t, n\}$. Note also that $N = m + 1$. From Eq. (2), the coefficient on each coalition $\{t, n\}$ in the Shapley value is

$$\frac{(|S|-1)!\,(N-|S|)!}{N!} = \frac{\big((n+1)-1\big)!\,\big((m+1)-(n+1)\big)!}{(m+1)!} = \frac{n!\,(m-n)!}{(m+1)\,m!}. \quad (40)$$

For each coalition, $\{t, n\}$, the marginal contribution for the taker in the formula for φ_t is $\big(\omega(\{t, n\}) - \omega(\{n\})\big) = \omega(\{t, n\})$. In aggregate, the partial sum in the Shapley value for a specific n is the product of the following three terms: (i) the number of $\{t, n\}$ coalitions, (ii) the Shapley coefficient that is common to each $\{t, n\}$ coalition, (40), and (iii) $\big(\omega(\{t, n\}) - \omega(\{n\})\big) = \omega(\{t, n\})$, from Eq. (27):

$$\frac{m!}{n!\,(m-n)!} \times \frac{n!\,(m-n)!}{(m+1)\,m!} \times \omega(\{t, n\} = \frac{1}{m+1}\left(\frac{n}{n+1}D + \frac{n^2}{n+1}d\right). \quad (41)$$

Summing this over all possible $n = 1, \ldots, m$ yields the final two terms in Eq. (28). This completes the derivation of the Shapley value for the taker, φ_t/λ_t, given that $\lambda_t = 1$.

B.2 Shapley Value for the Makers

In deriving the Shapley value for maker i, φ_i, note that for any coalition, $\hat{S}$, where $t \notin \hat{S}, \omega(\hat{S}) = 0$. This simplifies the remaining steps for calculating the Shapley value for i to only those worth functions whose coalitions include a taker, t, as a member; i. e., $\omega(\{t, n\})$.

Proof. For any maker, i, there is one and only one $\{t, i\}$ coalition. The coefficient on this coalition in the Shapley value is

$$\frac{(2-1)!\,\big((m+1)-2\big)!}{(m+1)!} = \frac{(m-1)!}{(m+1)!} = \frac{1}{m\cdot(m+1)}. \tag{42}$$

As $\omega(\{t, i\}) = {}^1\!/_2 D + {}^1\!/_2 d$ and $\omega(\{t, i\} \setminus \{i\}) = \delta D - F$, the part of the calculation of φ_i that corresponds to the marginal contribution of i to $\{t, i\}$ is:

$$\frac{1}{m\cdot(m+1)}\Big(\omega(\{t,i\}) - \omega(\{t,i\}\setminus\{i\})\Big) = \frac{1}{m\cdot(m+1)}\left(\frac{1}{2}D + \frac{1}{2}d - (\delta D - F)\right). \tag{43}$$

From Eq. (2) the calculation of the Shapley value is now:

$$\begin{aligned}\varphi_i = {} & \frac{1}{m\cdot(m+1)}\left(\frac{1}{2}D + \frac{1}{2}d - (\delta D - F)\right) \\ & + \sum_{\substack{i\in\{\{t,n\}\mid n\geq 2\},\\ \{t,n\}\subseteq N}} \frac{(|S|-1)!\,(N-|S|)!}{N!} \times \Big(\omega(\{t,n\}) - \omega(\{t,n\}\setminus\{i\})\Big).\end{aligned} \tag{44}$$

The remainder of the coalitions where $i \in \{t, n\}$ require $n \geq 2$. For a given n, the number of coalitions for which maker i is a member, $i \in \{t, n\}$, is

$$\binom{m-1}{n-1} = \frac{(m-1)!}{(n-1)!\,\big((m-1)-(n-1)\big)!} = \frac{(m-1)!}{(n-1)!\,(m-n)!}. \tag{45}$$

Given n, the coefficient in the Shapley value for the marginal contribution, $\omega(\{t, n\}) - \omega(\{t, n\} \setminus \{i\})$, of maker i is

$$\frac{\big((n+1)-1\big)!\,\big((m+1)-(n+1)\big)!}{(m+1)!} = \frac{n!\,(m-n)!}{(m+1)!}. \tag{46}$$

To calculate $\omega(\{t, n\}) - \omega(\{t, n\} \setminus \{i\})$, use Eq. (27) and observe that

$$\omega(\{t,n\}\setminus\{i\}) = \omega(\{t,n-1\} = \frac{n-1}{n}D + \frac{(n-1)^2}{n}d. \tag{47}$$

Consequently, i's marginal contribution to the coalition $\{t, n\}$ is

$$\omega(\{t,n\}) - \omega(\{t,n\}\setminus\{i\}) = \frac{1}{n\cdot(n+1)}D + \frac{n^2+n-1}{n\cdot(n+1)}d. \tag{48}$$

Hence, when $n \geq 2$, for a given value of n the partial sum within the Shapley value corresponding to $\{t, n\}$ such that $i \in \{t, n\}$ is the product:

$$\underbrace{\frac{(m-1)!}{(n-1)!\,(m-n)!}}_{\text{Eq. 45}} \times \underbrace{\frac{n!\,(m-n)!}{(m+1)!}}_{\text{Eq. 46}} \times \underbrace{\left(\frac{1}{n\cdot(n+1)}D + \frac{n^2+n-1}{n\cdot(n+1)}d\right)}_{\text{Eq. 48}}$$
$$= \frac{1}{m\cdot(m+1)} \times \left(\frac{1}{n+1}D + \frac{n^2+n-1}{n+1}d\right). \tag{49}$$

Summing this over all possible $n = 2, \ldots, m$ yields:

$$\frac{D}{m\cdot(m+1)}\sum_{n=2}^{m}\frac{1}{n+1} + \frac{d}{m\cdot(m+1)}\sum_{n=2}^{m}\frac{n^2+n-1}{n+1}. \tag{50}$$

To derive φ_i, combine this with the Shapley value term for $\{t, i\}$, Eq. (43):

$$\varphi_i = \frac{1}{m\cdot(m+1)}\left(\frac{1}{2}D + \frac{1}{2}d - (\delta D - F)\right)$$
$$+ \frac{D}{m\cdot(m+1)}\sum_{n=2}^{m}\frac{1}{n+1} + \frac{d}{m\cdot(m+1)}\sum_{n=2}^{m}\frac{n^2+n-1}{n+1}. \tag{51}$$

Aggregating terms:

$$\varphi_i = \frac{D}{m\cdot(m+1)}\sum_{n=1}^{m}\frac{1}{n+1} + \frac{d}{m\cdot(m+1)}\sum_{n=1}^{m}\frac{n^2+n-1}{n+1} - \frac{\delta D - F}{m\cdot(m+1)}. \tag{52}$$

This completes the derivation of the Shapley value for a maker, φ_i/λ_i, given $\lambda_i = 1$.

B.3 Feasibility Check

The final step requires verification that φ_i/λ_i is feasible for $V(N)$. This is facilitated using the expressions of φ_t/λ_t and φ_i/λ_i in terms of harmonic numbers, as given in Eqs. (56) and (57) in Appendix C.

Proof. Recall that the fee, f, given in Eq. (31) was derived from the feasibility condition for φ_t/λ_t when $\lambda_t = 1$: $\varphi_t \leq \frac{m}{m+1}D - mf$, yielding $f = \frac{1}{m+1}D - \frac{1}{m}\varphi_t$. The feasibility condition for φ_i/λ_i when $\lambda_i = 1$ is

$$\varphi_i \leq \frac{m}{m+1}d + f = \frac{m}{m+1}d + \frac{1}{m+1}D - \frac{1}{m}\varphi_t. \tag{53}$$

Setting the two sides equal and substituting in the value for φ_t, Eq. (56):

$$\begin{aligned}\varphi_i = &\frac{m \cdot d}{m+1} + \frac{D}{m+1} \\ &- \frac{1}{m}\left(\frac{\delta D - F + D\,(m - H_{m+1} + 1)}{m+1}\right) \\ &+ \frac{1}{m}\left(\frac{d\left(H_{m+1} + \left(\frac{m}{2} - 1\right)(m+1)\right)}{m+1}\right),\end{aligned} \tag{54}$$

which reduces to

$$\varphi_i = \frac{D(H_{m+1} - 1) + d\left(\frac{m^2+m}{2} - H_{m+1} + 1\right) - \delta D + F}{m \cdot (m+1)}. \tag{55}$$

This is exactly the right side of Eq. (57). Hence the condition holds under the theorem.

C Alternative Form of Theorem 2 Using Harmonic Numbers

Let H_m denote the m-th harmonic number, i.e., $H_m = \sum_{n=1}^{m} \frac{1}{n}$. Using this shorthand, Eq. (28) of Theorem 2 can be rewritten as,

$$\frac{\varphi_t(\omega, \boldsymbol{\lambda})}{\lambda_t} = \frac{\delta D - F + D \cdot (m - H_{m+1} + 1) + d \cdot \left(H_{m+1} + \left(\frac{m}{2} - 1\right)(m+1)\right)}{m+1}; \tag{56}$$

and Eq. (29) becomes:

$$\frac{\varphi_i(\omega, \boldsymbol{\lambda})}{\lambda_i} = \frac{D \cdot (H_{m+1} - 1) + d \cdot \left(\frac{m^2+m}{2} - H_{m+1} + 1\right) - \delta D + F}{m \cdot (m+1)}. \tag{57}$$

References

1. Abramova, S., Schöttle, P., Böhme, R.: Mixing coins of different quality: a game-theoretic approach. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 280–297. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_18
2. Acquisti, A., Dingledine, R., Syverson, P.: On the economics of anonymity. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 84–102. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45126-6_7
3. Acquisti, A., Taylor, C., Wagman, L.: The economics of privacy. J. Econ. Lit. **54**(2), 442–492 (2016)

4. Acquisti, A., Varian, H.R.: Conditioning prices on purchase history. Market. Sci. **24**(3), 367–381 (2005)
5. Androulaki, E., Raykova, M., Srivatsan, S., Stavrou, A., Bellovin, S.M.: PAR: payment for anonymous routing. In: Borisov, N., Goldberg, I. (eds.) PETS 2008. LNCS, vol. 5134, pp. 219–236. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70630-4_14
6. Böhme, R., Christin, N., Edelman, B., Moore, T.: Bitcoin: economics, technology, and governance. J. Econ. Perspect. **29**(2), 213–238 (2015)
7. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: anonymity for bitcoin with accountable mixes. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 486–504. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_31
8. Chessa, M., Grossklags, J., Loiseau, P.: A game-theoretic study on non-monetary incentives in data analytics projects with privacy implications. In: Fournet, C., Hicks, M.W., Viganò, L. (eds.) Proceedings of the Computer Security Foundations Symposium (CSF), pp. 90–104. IEEE Computer Society (2015)
9. Chessa, M., Loiseau, P.: A cooperative game-theoretic approach to quantify the value of personal data in networks. In: Proceedings of the 12th Workshop on the Economics of Networks, Systems and Computation, no. 9. ACM, Cambridge (2017)
10. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P. (eds.) PET 2002. LNCS, vol. 2482, pp. 54–68. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36467-6_5
11. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: 13th USENIX Security Symposium. USENIX Association (2004)
12. Dwork, C.: Differential privacy: a survey of results. In: Agrawal, M., Du, D., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 1–19. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79228-4_1
13. Franz, E., Jerichow, A., Wicke, G.: A payment scheme for mixes providing anonymity. In: Lamersdorf, W., Merz, M. (eds.) TREC 1998. LNCS, vol. 1402, pp. 94–108. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053404
14. Friedman, E., Resnick, P.: The social cost of cheap pseudonyms. J. Econ. Manag. Strategy **10**(2), 173–199 (2001)
15. Gkatzelis, V., Aperjis, C., Huberman, B.A.: Pricing private data. Electron. Markets **25**(2), 109–123 (2015)
16. Harsanyi, J.C.: A simplified bargaining model for the n-person cooperative game. Int. Econ. Rev. **4**(2), 194–220 (1963)
17. Jansen, R., Johnson, A., Syverson, P.: LIRA: lightweight incentivized routing for anonymity. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). The Internet Society (2013)
18. Kleinberg, J., Papadimitriou, C.H., Raghavan, P.: On the value of private information. In: Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge (TARK), pp. 249–257. ACM, New York (2001)
19. Köpsell, S.: Low latency anonymous communication – how long are users willing to wait? In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 221–237. Springer, Heidelberg (2006). https://doi.org/10.1007/11766155_16
20. Maschler, M., Owen, G.: The consistent Shapley value for games without side-payments. In: Selten, R. (ed.) Rational Interaction, pp. 5–12. Springer, New York (1992). https://doi.org/10.1007/978-3-662-09664-2_2
21. Maxwell, G.: CoinJoin: Bitcoin privacy for the real world (2013). https://bitcointalk.org/index.php?topic=279249.0

22. Meiklejohn, S., Orlandi, C.: Privacy-enhancing overlays in bitcoin. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) FC 2015. LNCS, vol. 8976, pp. 127–141. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_10
23. Möser, M., Böhme, R.: Join me on a market for anonymity. In: Workshop on the Economics of Information Security (WEIS), Berkeley, CA (2016)
24. Möser, M., Böhme, R.: The price of anonymity: empirical evidence from a market for Bitcoin anonymization. J. Cybersecur. **3**(2), 127–135 (2017)
25. Möser, M., Böhme, R., Breuker, D.: An inquiry into money laundering tools in the Bitcoin ecosystem. In: Proceedings of the APWG E-Crime Researchers Summit, pp. 1–14. IEEE, San Francisco (2013)
26. Myerson, R.B.: Fictitious-transfer solutions in cooperative game theory. In: Selten, R. (ed.) Rational Interaction, pp. 13–33. Springer, Heidelberg (1992). https://doi.org/10.1007/978-3-662-09664-2_3
27. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Crypocurrency Technologies. A Comprehensive Introduction. Princeton University Press, Princeton (2016)
28. Owen, G.: Game Theory. W. B. Saunders, Philadelphia (1968)
29. Padilla, A.J., Pagano, M.: Sharing default information as a borrower discipline device. Eur. Econ. Rev. **44**(10), 1951–1980 (2000)
30. Pagano, M., Jappelli, T.: Information sharing in credit markets. J. Finance **48**(5), 1693–1718 (1993)
31. Peleg, B., Sudhölter, P.: Introduction to the Theory of Cooperative Games, 2nd edn. Kluwer Academic, Boston (2007)
32. Pfitzmann, A., Köhntopp, M.: Anonymity, unobservability, and pseudonymity - a proposal for terminology. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 1–9. Springer, Heidelberg (2001)
33. Saad, W., Han, Z., Debbah, M., Hjørungnes, A., Başar, T.: Coalitional game theory for communication networks: a tutorial (2009). https://arxiv.org/abs/0905.4057
34. Sandler, T.: Collective Action. University of Michigan Press, Ann Arbor (1992)
35. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Dingledine, R., Syverson, P. (eds.) PET 2002. LNCS, vol. 2482, pp. 41–53. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36467-6_4
36. Shapley, L.S.: A value for n-person games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games, vol. II, pp. 307–317. Princeton University Press, Princeton (1953)
37. Shapley, L.S.: Utility comparisons and the theory of games. In: La Décision: Aggrégation et Dynamique des Orders de Préférences, pp. 251–263. Éditions du Centre Nationale de la Recherche Scientifique, Paris (1969)
38. Shapley, L.S.: Utility comparisons and the theory of games. In: Roth, A.E. (ed.) The Shapley Value. Essays in Honor of Lloyd S. Shapley, pp. 307–319. Cambridge University Press (1988)
39. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertainty Fuzziness Knowl.-Based Syst. **10**(5), 571–588 (2002)
40. Zhu, H.: Do dark pools harm price discovery? Rev. Financ. Stud. **27**(3), 747–780 (2014)

A New Look at the Refund Mechanism in the Bitcoin Payment Protocol

Sepideh Avizheh[1(✉)], Reihaneh Safavi-Naini[1], and Siamak F. Shahandashti[2]

[1] University of Calgary, Calgary, Alberta, Canada
sepideh.avizheh1@ucalgary.ca
[2] University of York, York, UK

Abstract. BIP70 is the Bitcoin payment protocol for communication between a merchant and a pseudonymous customer. McCorry et al. (FC 2016) showed that BIP70 is prone to refund attacks and proposed a fix that requires the customer to sign their refund request. They argued that this minimal change will provide resistance against refund attacks. In this paper, we point out the drawbacks of McCorry et al.'s fix and propose a new approach for protection against refund attacks using the Bitcoin multisignature mechanism. Our solution does not rely on merchants storing refund requests, and unlike the previous solution, allows updating refund addresses through email. We discuss the security of our proposed method and compare it with the previous solution. We also propose a novel application of our refund mechanism in providing anonymity for payments between a payer and payee in which merchants act as mixing servers. We finally discuss how to combine the above two mechanisms in a single payment protocol to have an anonymous payment protocol secure against refund attacks.

1 Introduction

Since the introduction of Bitcoin in 2008 [15], it has been widely adopted by merchants as a payment method. By 2015, the number of merchants accepting Bitcoin was reported to surpass 100,000 [9] and it has continued to expand into new markets (see e.g. [10,11]). Bitcoin standards are developed through a process which involves a so-called (Standard Track) Bitcoin Improvements Proposal (BIP) being proposed, discussed, ratified, and adopted by the Bitcoin community. BIP70 [1] is the Bitcoin Payment Protocol standard that defines the communications between a pseudonymous customer and a merchant with a public key certificate. The protocol, provides a number of properties to improve the interaction between the two entities (e.g. allows the merchant's address to be human-readable) and also provides the necessary guarantees (e.g. a proof of payment to the customer that can be used for dispute resolution). One important feature of the protocol is that the customer can specify refund addresses that

The full version of this paper is available at https://arxiv.org/abs/1807.01793 [3].

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 369–387, 2018.
https://doi.org/10.1007/978-3-662-58387-6_20

will be used by the merchant in the case of refunds for cancelled orders or overpayments.

McCorry, Shahandashti and Hao however showed two *refund attacks* on the BIP70 protocol [14]. These attacks are referred to as the Silkroad Trader and Marketplace Trader attacks, and exploit the inadequacies of the refund mechanism in the standard, including for example the fact that the protocol only provides one-way authentication. In the Silkroad Trader attack, a malicious customer uses the refund mechanism to relay a payment to an illicit merchant (the Silkroad Trader) through an honest merchant. This is by simply declaring the Bitcoin address of the Silkroad Trader as the refund address, and later asking for a refund. McCorry et al. [14] discuss the steps required for the attack in detail and showed its feasibility by successfully carrying out the attack in real-life payment scenarios. The authors also described a second attack, called Marketplace Trader attack, in which a rogue trader plays the role of a Man-in-the-Middle (MITM) between the customer and a reputable merchant and effectively direct the customer's payment to its own address after using the inadequate authentication of the protocol to change the refund address. In both these attacks, the analysis of blockchain data will not reveal the attacks. In McCorry et al.'s solution the customer signs the refund addresses by the public key they have used in the payment (the signature is called a *proof of endorsement*) and so effectively binds the refund addresses to the customer address. This prevents the first attack since the customer cannot deny their link to the Silkroad trader anymore. McCorry et al. [14] argue that this measure also discourages the second attack since merchants will become more reluctant to update the refund address through unauthenticated channels such as email.

McCorry et al.'s solution, although minimally changing the protocol, introduces a major data management challenge for the merchant. This is because protection against the Silkroad trader attack requires the merchant to maintain a database of the proofs of endorsement and transactions that are signed by the customer. These transactions include the refund information including the amount and the refundee address that the merchant must use, and the amount and address of the customer that the merchant has received the bitcoins from. Using the stored data, the merchant can "prove" that they have followed the customer's request and have not colluded with the customer in transferring money to the refundee (Silkroad Trader). We refer to this as an *explicit log solution* where all the relevant information must be stored and kept indefinitely by the merchant. The stored data are also privacy sensitive and so the merchant must adopt extra measures to secure the storage. The data must be kept in the database indefinitely as proof may become necessary at any time in the future. The database must be securely backed up (e.g. using cloud services) to ensure data is available when required.

Our Contributions. We introduce *implicit logging* that requires the merchant to only store a number of indexes to the blockchain that will be used to recover the required proof of innocence, when needed. The solution works as follows: a refund operation consists of two transactions produced by the merchant. The

first transaction is a two output transaction with the refund amount that requires the signatures of the refundee and the customer to release the fund. The second transaction has the same value and is issued to the customer, with a time lock that allows the transaction to be released only after a specified time passes. The refundee requires the signature of the customer to receive their refund, which implies that they endorse the refund. If the customer does not endorse the refund (e.g. in the case of a Marketplace Trader attack), the first transaction will not be redeemed in time and hence the customer will be able to use the second (time-locked) transaction to redeem the bitcoins.

The merchant will store the indexes of the two transactions that are issued for a refund request, together with the indexes of the original payment transaction and the redeemed transaction. In the case of multiple refundees, all their details will be included in the first transaction and so the same amount of storage will be required.

The solution is robust in the sense that transactions that are associated with a refund and contain the proof of the relation between the customer and the refundee are kept on the blockchain and are immutable. If the database that contains the indexes of the transactions is corrupted, the merchant can still recover the information about the transactions that they have issued or received, by searching the blockchain for transactions that include their public key information. The solution preserves the privacy of the refund transaction against an adversary who has access to the blockchain and eavesdrops on the communication between the customer and the merchant, in the sense that the linkage between the customer and the refundee can only be revealed by the customer or the merchant. We will discuss how the scheme can be modified when there are more than one transaction issuers (this was considered by McCorry et al.). The details are provided in the full version of the paper [3].

We show that this solution also provides protection against Marketplace Trader attack without putting any restriction on BIP70. This is in contrast with McCorry et al.'s solution that requires the refund address not be accepted through email. This restriction is likely to be ignored in practice, rendering McCorry et al.'s solution incapable of preventing Marketplace Trader attacks.

In Sect. 4 we consider a novel application of the refund mechanism in providing payment anonymity. The main observation is that refund addresses in Bitcoin can effectively provide a level of indirection that if carefully used, can decouple the payer and payee. In our proposal, the merchant provides a mixing service that allows the customers to pay for other services and to other merchants using a combination of overpayment and refund. By "mixing" transactions of multiple customers, the linkage of transactions using their payer and payee fields, as well as values, will be removed. We define the communication protocol between the customer and the mixing service based on BIP70. In the full version of the paper [3] we discuss how the above two mechanisms can be combined to provide an anonymous payment protocol with security against refund attacks.

2 Preliminaries

2.1 The BIP70 Payment Protocol

BIP70 [1] is a Bitcoin application layer payment protocol that defines the sequence of messages communicated between a customer and a merchant. BIP70 consists of three messages: *payment request*, *payment*, and *payment acknowledgment*. It proceeds as follows.

After the customer selects an item from the merchant's website and clicks to pay, the merchant responds by sending a *payment request* message. This message contains *payment details*, the information related to merchant's X.509 certificate (PKI type and PKI data), as well as the signature of the merchant on the hash of the payment request. Here *payment details* consists of the Bitcoin address that the customer should send the bitcoins to, the time that request has been created, an expiration time, a memo containing notes to the customer, a payment URL, and finally the merchant data which is used by the merchant to identify the payment request.

The customer's Bitcoin wallet subsequently verifies the signature and the merchant's identity, the information in the payment details, such as the time of the request creation and expiry, displays the merchant's identity, the amount to pay, and the memo to the customer and asks the customer whether they want to continue. If confirmed, the wallet will create the necessary Bitcoin transactions for the payment and broadcast them to the Bitcoin peer-to-peer (P2P) network. Then, a *payment* message is sent to the merchant. This message consists of the merchant data from the *payment details* in payment request, one or more valid Bitcoin transactions, the $refund\_to$ field which specifies a set of refund amount and address pairs to be used in case of a refund request, and a note for the merchant (memo).

When the merchant receives the payment message, it verifies that the transactions satisfy the payment conditions, broadcasts the transactions, and sends back a *payment acknowledgment* message. This message contains a copy of the payment message and a final memo including a note on the status of the transaction.

BIP70 does not specify how the payment request message should be downloaded, but requires that the payment and payment acknowledgment messages are communicated over a secure channel (such as HTTPS).

BIP70 does not explicitly define a refund protocol. It is implicitly assumed that if the customer requests a refund identifying the payment by the *merchant data* field, the merchant issues a refund transaction which sends the refund amounts to the corresponding refund addresses specified in the $refund\_to$ field of the *payment* message.

Figure 1 shows the communication flow in BIP70 and its implicit refund procedure. Note that besides communicating with each other, both the customer and the merchant are assumed have access to the Bitcoin P2P network. Both are able to broadcast transactions to the Bitcoin P2P network. Upon receiving a transaction, the Bitcoin P2P network decides whether to add the transaction

into the distributed ledger (i.e. the Bitcoin blockchain) through the Bitcoin consensus mechanism. Both the customer and the merchant are also able to check whether their broadcasted transaction has been included in the blockchain. It is important to distinguish the communications in the application layer payment protocol and those in the P2P network. To simplify our diagrams, we do not explicitly show the P2P network communications.

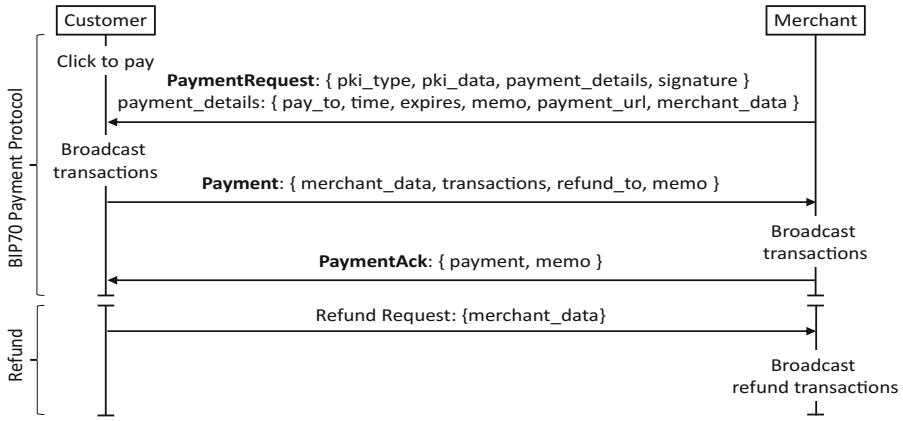


Fig. 1. The BIP70 payment protocol and its refund procedure. Note that the Bitcoin P2P network to which the transactions are broadcast is not explicitly shown here.

2.2 Refund Attacks

McCorry et al. propose two attacks on the refund process of BIP70 [14]. These attacks work even if a secure channel such as HTTPS is used for communication between parties. We briefly describe these two attacks in the following.

Silkroad Trader Attack. The refund addresses provided by the customer (in the *refund_to* field) are in no way endorsed and can be repudiated at a later time. This means that a malicious customer may abuse the refund mechanism to relay their payment to an illicit trader (here called the Silkroad trader) through an honest merchant. The customer simply provides the illicit trader's address as the refund address to the merchant and thus when a refund is requested, the merchant will send the refund to the illicit Trader. The customer can later deny abusing the refund mechanism and the merchant will have no way to prove they have been cheated. Figure 2 shows the interaction among parties in this attack.

Marketplace Trader Attack. Some merchants allow customers to specify new refund addresses upon a refund request. The customer requesting the refund is not authenticated. This means that any entity who has knowledge of the payment identifier (specified in the *merchant data* field of the payment details in the payment request message) can request a refund to any arbitrary account.

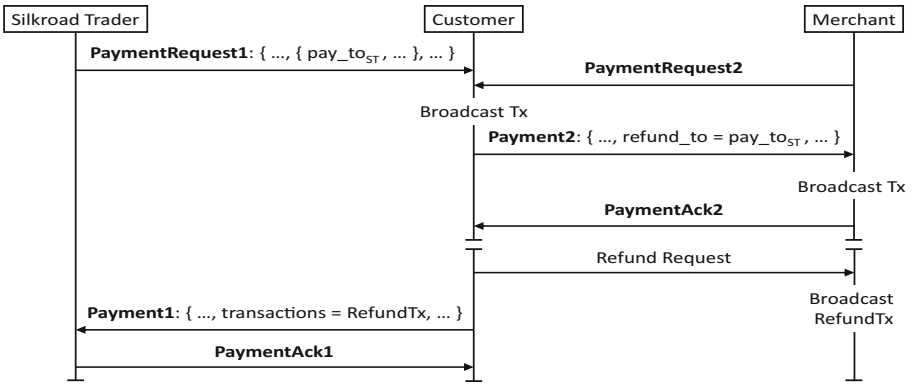


Fig. 2. The Silkroad Trader attack.

This is the basis for the Marketplace Trader attack, in which a rogue trader acts as relaying man-in-the-middle for the payment request message between the merchant and the customer. Hence, the rogue trader is able to find out *merchant data*. At a later time, the rogue trader requests a refund to an arbitrary address and is able to steal the funds. Figure 3 shows the interactions among the parties in this attack.

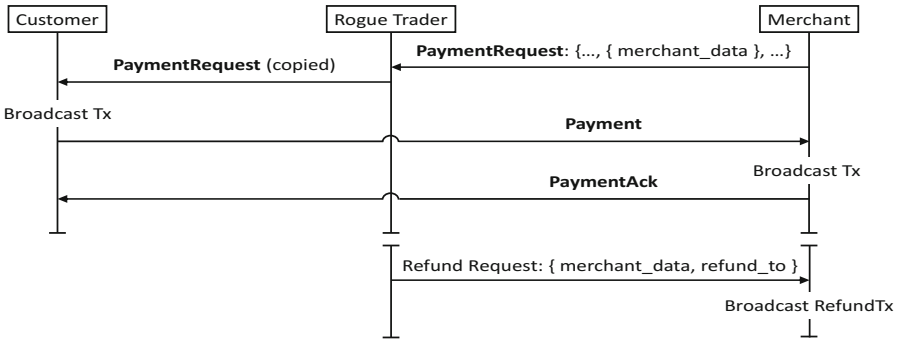


Fig. 3. The Marketplace Trader attack.

2.3 McCorry et al.'s Solution to Refund Attacks

McCorry et al. propose to include in the payment message a "proof of endorsement" for refund addresses. To do this, each customer address involved in the payment protocol is required to produce a digital signature on (and therefore "endorse") a corresponding refund address. Employing this solution, at the end of a successful payment protocol, the merchant will be in possession of a proof of

endorsement for each refund address. Such a proof can be presented and verified by a third party in case of a Silkroad Trader attack to implicate the malicious customer. Besides, since such a proof of endorsement is valuable for merchants, McCorry et al. argue that it will discourage merchants to accept new refund addresses unless accompanied by a proof of endorsement, resulting in reducing the possibility of Marketplace Trader attacks.

In McCorry et al.'s solution, for guaranteed protection against attacks, merchants will need to store payment transactions as well as payment requests and payment messages which are required to verify the proof of endorsement. Therefore, the actual storage overhead of McCorry et al.'s solution is much larger than only keeping proofs of endorsement. As noted earlier, maintaining a secure and robust database to store proof of endorsement messages is a security bottleneck of the system and can particularly become expensive for smaller merchants with limited resources.

2.4 Multisignature and Time-Locked Transactions in Bitcoin

Although it is convenient to think of Bitcoin transactions as sending funds to certain account addresses, technically what the transaction specifies is a set of redemption criteria in a certain script language. Any subsequent transaction which satisfies the redemption criteria may authorize the transfer of funds made available in the original transaction.

The most popular script is "Pay to Public Key Hash" (P2PKH), which requires a signature corresponding to an address, hence effectively sending the bitcoins to the address. Typical Bitcoin transactions use this script.

Another popular and more versatile script is "Pay to Script Hash" (P2SH), which requires satisfying a script, the hash of which is listed. P2SH can be used to implement a diverse range of transactions including *multisignature* transactions. A k-of-n multisignature transaction requires k signatures corresponding to k addresses within a set of n specified addresses to be present to redeem the funds in the transaction.

An interesting script which can be combined with the ones discussed above is one that effectively freezes the transaction funds until a time in the future to create a so-called *time-locked* transaction. The funds in a time-locked transaction cannot be spent by any other transaction until a certain (absolute or relative) time in the future.

3 A New Approach to Protection Against Refund Attacks

We propose a solution to refund attacks that requires the merchant to store a fixed number of indexes (and so constant size) in each run of BIP70 protocol. The solution is robust to possible damages to database content, and ensures privacy of refund transaction from outsiders.

BIP70 requires the *payment* and *payment acknowledgment* to be sent over a secure channel, however does not specify such a requirement for the *payment request* message [1]. Therefore, in general we consider two types of attackers:

- **Online attacker**, intercepts the communication channel and sees all the input/output messages of a merchant; they also have access to blockchain data.
- **Offline attacker**, has only access to the blockchain data.

To simplify our description, we first assume BIP70 communication is over HTTPS and so we only need to consider an offline attacker. We will then show how to secure the protocol against an online attacker.

Our goal is to provide the following properties for the refund mechanism:

Implicit log. The merchant only stores indexes of transactions in a protocol run. This has the following advantages:
  - **Constant storage size per protocol run.** The local storage size for a customer's payment in a protocol run is constant.
  - **Robustness.** The refund mechanism will work correctly and reliably in the case of a dispute, even if the merchant's local database is corrupted or lost. In the worst case when none of the locally stored indexes are accessible, the merchant can recover the required proofs by searching the blockchain using their own public key information.

Refund privacy. An offline adversary with access to the blockchain, or an online adversary with access to the blockchain and the communication link between the merchant and the customer, cannot reveal the linkage between the customer and the refundee. Note that BIP70 does not require secure communication between the customer and the merchant and so an online adversary can access unencrypted communication between the two. We note that the merchant's local database must be kept secure for customers' privacy.

We also aim to conform with BIP70 specifications and avoid extra restrictions including *not* accepting refund addresses by email. Note that Refund addresses are valid for two months from the time of the payment [1], and during this period the customer should be allowed to change the refund addresses for example when an existing refundee has lost their wallet. Coinbase and Bitpay [7,8] both accept refund address updates via email.

3.1 Our Solution

Our proposed refund mechanism works as follows. The merchant creates a 2-of-2 multisignature transaction, and hence binds the refund amount to both the customer and refundee. Then, to make the protocol robust in case one of the addresses is not available, a second transaction is created. This second transaction is a time-locked transaction, and the customer is its only recipient. Merchant uses a lock time for this transaction to give priority to the first transaction. If the

customer and the refundee could not collaborate to redeem the refund, the customer is able to claim them after the lock time. Note that the lock time creates a delay in the system only if the customer does not know the refundee, e.g. in the case of a Marketplace Trader attack. In other words, the second transaction is a backup for system robustness (see Fig. 4).

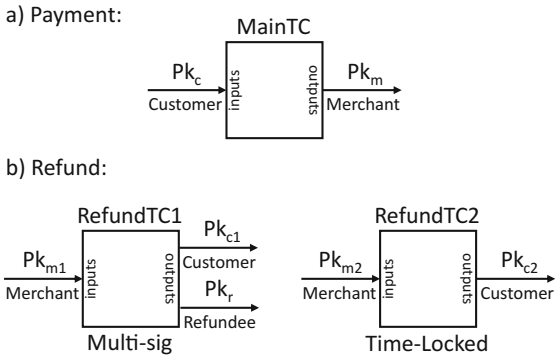


Fig. 4. (a) The main transaction. (b) The proposed Refund mechanism, in which merchant locks the transaction to the customer and the refundee and they may redeem this transaction if they collaborate. The merchant also issues RefundTC2 for robustness to ensure that the refund can be unlocked by customer in case the first transaction is not redeemed.

In addition, to preserve refund privacy against offline attackers[1] the merchant deterministically creates fresh addresses from the public key of the customer and then masks them with a Diffie-Hellman key, generated using the fresh address of the customer and the private key of the merchant. This is to ensure that only the merchant or the customer who are able to re-generate the Diffie-Hellman key can discover the linkage between the refund key and the payment transaction. To derive fresh addresses we assume the customer has a deterministic wallet based on BIP32 [19]. Most Bitcoin wallets support BIP32 and so this is a reasonable assumption. In the rest of the paper, by deterministic wallet we mean a BIP32 wallet with a master key and key hierarchy (keys can be specified with an index).

Based on BIP32, a deterministic wallet generates a tree of public/private key pairs on elliptic curve E, e.g. for a 1-level tree, it creates 2^{31} *hardened* and 2^{31} *non-hardened* keys. Hardened keys are public keys for which the associated private keys can only be known before the generation of the public key. Non-hardened keys however allow anyone to derive a valid child public key from the parent public key, while the owner of the parent (master) private key can generate the respective child private key. In our protocol, the customer address in the payment transaction is a non-hardened public key which is used as a

[1] Bitcoin transactions use fresh addresses (address freshness) [5] to protect the privacy of the address owner as well as others.

parent key by the merchant to derive child public keys. The customer knows the respective private keys and can also create the Diffie-Hellman key using the child private key and the public key of the merchant. Hardened keys can be used for refund addresses. The step by step process is given below:

Key generation. The customer wallet software generates a tree of public/private key pairs using BIP32 [19]. Each private key is an element of F_q, where $q = p^n$ is a prime power. Each public key is a point on an elliptic curve E (specified for bitcoin) over F_q. H_l denotes the 256 leftmost bits of HMAC-SHA512 which is used for computing the child key; the rightmost bits of HMAC-SHA512 are used as the next level chain code. Let $Pk_c = cP$ be a non-hardened public key. A child public key can be derived by anyone using this parent public key as follows: $Pk'_c = Pk_c + H_l(ch_c, Pk_c||index)P$; where Pk'_c is a child public key, $Pk'_c = c'P$, but only the customer who knows the parent private key can compute the child private key c'. ch_c denotes the chain code which is the 256 rightmost side of the parent hash, and *index* is the index of the generated child key in the tree. (Pk_c, ch_c) is called the *extended pubic key* (see [19] for details). We also use H^* to denote a collision resistant hash function which maps a point on Elliptic curve E to F_q, this is used for computing the Diffie-Hellman component in child keys as we describe it below.

Click to pay. The customer visits the merchant website and chooses an item, then declares their intent to pay (e.g. by clicking a "pay" button).

Payment request. The merchant sends the payment request including their public key, $Pk_m = mP$. This public key is unique for each transaction.

Payment message. The customer after authenticating the merchant, puts together a payment transaction, *MainTC*, that transfers the cost of the chosen item to the merchant, and uses a non-hardened extended public key (with external chain) $(Pk_c = cP, ch_c)$, as a data output (using the OP_Return opcode). Finally, the customer creates a *Payment* message based on *MainTC*, and specifies their extended public key, n refund addresses, $(Pk_{r_1}, Pk_{r_2}, \ldots, Pk_{r_n})$, and the amount of refund for each address.

Payment ack. The merchant detects *MainTC*, and returns a *PaymentAck* message to the customer.

Refund request. Within a 2 month period from the payment request [1], the customer can use the addresses provided in *Payment.refund_to* field to receive their refund. In this case, the merchant does the following:

1. Generates two new keys, $Pk_{m_1} = m_1P$ and $Pk_{m_2} = m_2P$.
2. Derives child keys from the customer extended public key as described earlier, Pk'_{c_i} for $1 \leq i \leq n+1$.
3. Masks the child keys as $Pk''_{c_i} = Pk'_{c_i} + H^*(m_1 Pk'_{c_i})P$ for $1 \leq i \leq n$, and $Pk''_{c_i} = Pk'_{c_i} + H^*(m_2 Pk'_{c_i})P$ for $i = n+1$.
4. Creates and broadcasts two transactions, RefundTC1 (in which the merchant's public key is Pk_{m1}) and RefundTC2 (in which the merchant's public key is Pk_{m2}). RefundTC1 is a P2SH transaction that can be redeemed by providing signatures from both Pk''_{c_i} and Pk_{r_i} for $1 \leq i \leq n$.

RefundTC2 is a P2PKH transaction that can be redeemed by the private key corresponding to $Pk''_{c_{n+1}}$ after a specified lock time period (e.g. one week).

3.2 Protection Against Silkroad Trader Attacks

In the Silkroad Trader attack, the customer wants to remove their link to the Silkroad Trader using a victim merchant. In our approach the refund transaction can be only redeemed if the customer and the refundee collaborate. Hence, the redemption of the refund transaction constitutes an evidence of the linkage between the customer and the refundee. This linkage, however, is hidden from those who observe the Bitcoin blockchain and so the merchant is the only one who knows this linkage. The merchant can prove that a payment transaction and a refund transaction are linked to each other by first deriving the child keys from the payment transactions and adding the respective Diffie-Hellman key (based on the public key of the merchant and the customer's child key) to them, then using the refundees' public key and the customer's masked child keys to create the P2SH address and show that it matches the address within the refund transaction; finally showing that the refund transaction has been redeemed by a pair of collaborating customer and refundee, thus establishing the evidence of the linkage. The merchant does not need to store the transactions in their database and the proofs are robustly preserved on the blockchain.

3.3 Protection Against Marketplace Trader Attacks

If the customer provides a refund address during the BIP70 protocol run and later update it via email, the merchant just uses the newest refund address and locks the refund amount to both the refundee and the customer. If the customer and the refundee collaborate to redeem the refund, the refund is finalized; otherwise, if the rogue trader sends their own address to the merchant, they cannot later claim the bitcoins since the customer will not collaborate with an unknown refundee to sign the refund transaction. In case of such an attack, the customer will be able to claim the refund after the lock time expires. Thus, the approach protects the customer against Marketplace Trader attacks and at the same time provides the possibility to update the refund addresses.

3.4 Communication over HTTP

BIP70 does not restrict the customer and merchant to use HTTPS. Our proposed solution to refund attacks although by design hide the relation between the customer and the merchant, *payment* messages include information such as the refundee's address which can be used by an online attacker to find the corresponding refund transactions by searching the blockchain for the transactions that contain the address and so trace refundees transactions. To protect against these attacks, the customer may use HTTPS, or Diffie-Hellman key agreement

on the merchant's public key and their own private key, to generate a key that will be used to encrypt sensitive messages such that they are decryptable by the merchant. Alternatively, merchant can follow the key generation algorithm of our scheme for both customers and refundees (for refundees derive child keys of the refund addresses and then mask them with Diffie-Hellman key on the refundees child key and their own private key). In this case refundee can still find the corresponding private key to claim the refund, but attacker cannot detect the refund transactions.

3.5 Analysis

In the following, we show how each of the mentioned properties will be satisfied.

Implicit Logging. A merchant may store information that are communicated during a payment protocol for various reasons, including bookkeeping, refund or exchange, or statistics about customers and products. Here we do not consider bookkeeping that is mainly for accounting purposes or the ability to honour refund or exchange policies. Nor we consider data storage that are for statistical analysis purposes. As noted in [14], using the Bitcoin Payment Protocol requires the merchant to store evidence to protect them against refund attacks. This information must be kept for a sufficiently long time to be effective in providing such protection.

Consider a single run of the payment protocol. Transactions which are the "evidence for innocence" (of the merchant) are stored on the blockchain. The merchant must just store the transaction indexes, that is, the transaction ids (txid) of the *MainTC*, both the refund transactions *RefundTC1* and *RefundTC2*, and *Redeem transaction* by which the customer (and possibly the refundee) redeem the refund. Table 1 shows a full refund record in the merchant's database. Each transaction index is 32 bytes and so $4 \times 32 = 128$ bytes are needed for the four transactions ids (*MainTC*, *RefundTC1*, *RefundTC2*, and *Redeem transaction*). Note that the merchant can always use a deterministic approach for the child key indexes, for example always start from index 0 and increment it for each new child key, and so they do not need to be stored. It is not difficult to see that storage size is independent of the number of refundees.

Table 1. One refund record in the merchant's database.

MainTC txid	RefundTC1 txid	RefundTC2 txid	Redeem txid
32 bytes	32 bytes	32 bytes	32 bytes

In the case of a dispute, the merchant can retrieve all the mentioned transactions from the blockchain using their stored transaction ids, and then use the chain code, index, and the public key of the customer, to derive the related child key, and use their own private key m_i, $i = 1, 2$, to re-create the masked address,

$Pk''_c = Pk'_c + H^*(m_i Pk'_c)P$. If the output of the P2PKH transaction with address Pk''_c is a spent output, it indicates that the customer has redeemed the refund without the collaboration of the refundee. If the redeemed transaction is the P2SH transaction, Pk''_c and the refund addresses in the *Redeem transaction* are used to re-generate the P2SH address. If the address matches the *RefundTC2* and it is a spent output we know that the owner of the masked child public key Pk''_c (or more precisely the owner of the public key Pk_c) and refundee have collaborated with each other to redeem the refund.

Minimal Storage Size. In the following we compare the storage cost of our proposed protocol with that of McCorry et al. [14]. The comparison summary can be found in Table 2. In McCorry et al.'s solution, the proof of endorsement is a signature that must be locally stored. Verification of this signature requires information about the main transaction and the communicated messages including the refund addresses, refund values, the memo from the customer, and the payment request message. The merchant also needs to store the transaction ids for both the payment and refund transactions. Let L_S denote the size of the proof of endorsement signature. The size of a transaction input with one signer is at least 146 bytes[2]. Other values are, refund address which is 34 bytes, refund value which is 8 bytes, memo and payment request message sizes are denoted by L_{pay} and can reach 50,000 bytes. Finally a transaction id is 32 bytes. Thus in total, for one refundee, $252 + L_S + L_{pay}$ bytes must be stored at the merchant side and this cost grows linearly with the number of refundees. The total storage for n refundees will be $210 + 42n + L_S + L_{pay}$ bytes which is significantly higher than our scheme. Note that we are not considering the size of the merchants' keys in our calculations.

Table 2. Storage size (in bytes) of our approach vs. [14]

Scenario	McCorry et al. [14]	Our approach
1 refundee	$252 + L_S + L_{pay}$	128
n refundees	$210 + 42n + L_S + L_{pay}$	128

Robustness. The payment protocol must work correctly in case of a dispute or when the information stored in the merchant's database is corrupted or lost. In [14], if the local database that stores the signature (proof of endorsement) is corrupted, the evidence of the collusion will be irreversibly lost and the merchant will become vulnerable to refund attacks. In our proposed approach however the merchant can exhaustively search on all their keys to retrieve the database records. To do so, the merchant re-generates all the private/public

[2] Previous transaction hash is 32 bytes, previous Tx-out index is 4 bytes, Tx-in script length is 1–9 bytes, public key is 33 bytes in compressed format, signature is 72 bytes, sequence number is 4 bytes.

keys using their wallet (through the master key) and then uses a blockchain explorer (e.g. blockchain.info) to search for the transactions that contain these public keys. We denote the search complexity on blockchain with O_S. Assuming the number of keys used by the merchant to be 2^k, the search complexity becomes $2^k O_S$. The retrieved transactions are then identified as *MainTC*, *RefundTC1*, and *RefundTC2* based on their types and the merchant role as a sender or receiver of each transaction. If the merchant is the recipient, the transaction is *MainTC*. If the merchant is the sender and the transaction is P2SH, it is *RefundTC1*, otherwise if the merchant is the sender and the transaction is P2PKH, the transaction is *RefundTC2*. *Redeem transactions* are also found by searching the output addresses in *RefundTC1* and *Refund TC2*. If the number of these transactions is ℓ the search complexity increases to $2^k.\ell.O_S$. After this classification, the merchant can follow the steps below to retrieve the database:

1. The merchant chooses one specific *MainTC* transaction and stores its index in the database.
2. To find the related refund transactions, the merchant should reconstruct the masked child key of the customer Pk_c''. First, merchant generates the child key of the customer, Pk_c', using the chain code stored in *MainTC* (for the index, merchant can always try the first index, i.e. pick index 0 and 1). Then, they mask the child key with the Diffie-Hellman component, $H(m_i Pk_c')P$ to generate Pk_c''. For this, the merchant should try all the private keys m_i, for $1 \leq i \leq 2^k$.
   - If Pk_c'' matches the key inside *RefundTC2*, the used index is stored in the database as *RefundTC2* txid. If this transaction is a spent transaction, merchant stores the corresponding *Redeem transaction* index in the database and halts, since the proof is fully retrieved and customer have spent the transaction alone.
   - Otherwise, $RefundTC1$ may have been spent, so the merchant uses Pk_c'' to find the corresponding *Redeem transaction* and uses the refund keys in that transaction to reconstruct the P2SH address. The corresponding *RefundTC1* is the one which contains the mentioned P2SH address. The index of this transaction is stored in the database and the proof is fully retrieved.
   - However, if *Redeem transaction* for a specific customer does not exist at all, it shows that customer has not redeemed the refund value yet, hence the merchant should wait for the customer to claim the refund and subsequently retrieve the database record.

The complexity of reconstruction depends on the number of keys the merchant has used, the key generation, the search operation, and the number of refund transactions. If key generation (calculating the point arithmetic and hashing related to masked child key) takes O_K, and the total number of customers is t, then the total complexity is upper bounded by $2^k(2tO_K + \ell O_S)$. The factor 2 for O_K is because we should follow this procedure for both refund transactions *RefundTC1* and *RefundTC2*.

Privacy. To show that our scheme guarantees privacy, that is, only the merchant and the customer can reveal the linkage between the customer and the refundee, we consider two types of attackers: online attackers and offline attackers. Since an online attacker has access to both the communication channel and the blockchain, and hence is stronger, we only provide the justification for privacy for an online attacker. Privacy against offline attackers who only have access to the blockchain is straightforward.

Suppose that the channel is not TLS-protected (worst case), so the online attacker can intercept all the communicated messages between the customer and the merchant. Through *PaymentRequest* and *Payment* message, the attacker discovers the public key of the merchant Pk_m^*, the extended public key of the customer (ch_c^*, Pk_c^*), and the public key of the refundee Pk_r^*. The goal of the attacker is to link the customer address Pk_c^* in MainTC to the refund address Pk_r^*. For simplicity we assume that *RefundTC1* and *Redeem transaction* is also given to the attacker; this simulates the case where the refundee is the attacker. In practice this assumption may not be true, and the attacker needs to also guess the transactions. Overall, the attacker observes the information given in Table 3.

Table 3. View of an online attacker.

Information	Key	Source(s)
Extended public key of customer	(ch_c^*, Pk_c^*)	MainTC
Public key of refundee	Pk_r^*	Payment message, Redeem transaction
Public key of merchant	$Pk_{m_1}^*$	RefundTC1
Public key of customer	Pk_c''	Redeem transaction

From *Redeem transaction*, the attacker can link the customer address Pk_c'' to Pk_r^*. The reason is that to claim the bitcoins both the customer and the refundee need to sign the *Redeem transaction*. Hence, a spent refund transaction shows that the customer with Pk_c'' knows the refundee with Pk_r^*. Thus, to link the customer Pk_c^* to refundee Pk_r^*, the attacker needs to just find whether Pk_c'' is derived from Pk_c^*.

Pk_c'' is generated as follows $Pk_c'' = Pk_c' + H^*(m_1^* c' P)P$, where Pk_c' is the child key derived from the customer address $Pk_c' = Pk_c^* + H_l(ch_c^*, Pk_c^* || index)P$. The attacker knows Pk_c^* and ch_c^*, so they can guess Pk_c' by trying different indexes with probability $\frac{1}{n}$, assuming the number of refundees is n. To link the child key Pk_c' to Pk_c'', the attacker needs to solve the decisional Diffie-Hellman problem (DDH) given $m_1^* c' P$, $Pk_{m_1}^* = m_1^* P$, and $Pk_c' = c' P$. Since solving DDH is hard (say with ϵ representing the probability of solving DDH), the attacker's probability of success will be $P_{success} = \frac{\epsilon}{n}$ which is negligible.

4 Bitcoin User Anonymity Using Refund Mechanism

Despite using pseudonym for senders and receivers of transaction, it has been shown that transactions can be linked [2,16,17] and combined with other data possibly reveal user identities. There have been a number of approaches for providing anonymity [4,12,13,18]. Stealth address schemes [18] guarantee address anonymity against an online attacker who intercepts the communication link and sees the Bitcoin address of the payee when it is sent to payer to create the transaction. By stealth address technique, payer adds a Diffie-Hellman key to the payee's address in a way that the corresponding private key is still known by payee. In CoinSwap [13] a party uses an intermediary node to send the payment to payee; the goal is anonymity against online attacker. In CoinJoin [12] a number of parties agree to create one transaction together, they also use values with equal worth to provide value anonymity against an offline attacker. In Fair Exchange [4] two people exchange their bitcoins with each other to achieve coins with a history that is unrelated to them, to resist against an offline attacker. Each of these solutions can be considered as a traditional mixing service. Users can also mix their coins through a mix server (e.g. bitmixer.io [6]), which receives their coins and pay them back a fresh coin, although mixer receives a fee from the user. This technique, however has a problem, user should trust the mix server that they will not steal their money.

Refund mechanism provides a level of indirection that can be used for adding anonymity to Bitcoin users. We propose to use merchants as a trusted mixing servers by using a modified BIP70 protocol refund's policy. To use this service, the customer visits the website of a merchant and selects an item for purchase. By using overpayment and the recipients' addresses as the refund addresses, the sender can send payment to refundees in an anonymous way. Alternatively, they can send the desired amount to the merchant and later cancel their order for the refund. In these situations, merchant acts as an intermediary to allow the customer to pay the bitcoins to the recipients indirectly. Merchant can also split the value to smaller chunks and mix the refund transactions of different entities to provide value and time anonymity respectively (see Fig. 5). Reputable merchants are generally trusted and are expected to follow the protocol. Note that the merchant does not know if the refundee in a refund transaction is a customer and cannot relate the output bitcoin addresses to the user. Merchants can benefit for such service by requesting a fee for it.

In our proposed protocol, the merchant receives inputs from customers' transaction, and issues transaction with outputs based on the addresses in the *Payment.refund_to* field. Refund addresses are extended public keys of refundees. Merchant generates child keys of the respective refund addresses, split the values of refund to smaller equal chunks and sends the partitioned values to child keys. Merchant considers the refund address as a parent key and uses its child keys for refund transactions to have a fresh address for each chunk. To provide confidentiality for the parent refund key (this is needed because we assume online attacker exists and the communication is HTTP), the merchant encrypts the child keys using the Diffie-Hellman key generated by the public key of the

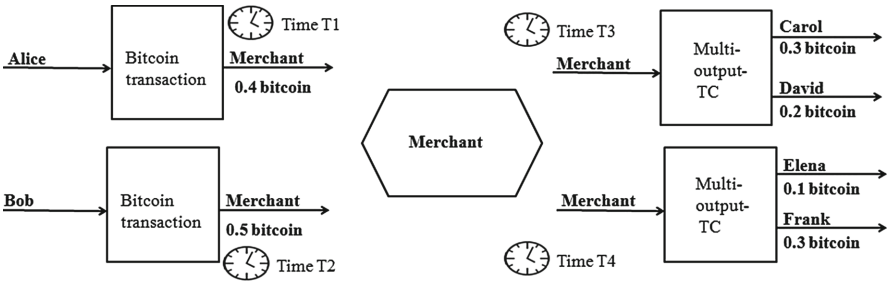


Fig. 5. Alice wants to send 0.4 Ƀ to Carol and Elena, and Bob 0.5 Ƀ to David and Frank. Alice pays the bitcoins to merchant, and introduces Carol and Elena as refundees, through BIP70. Bob also pays the bitcoins to merchant and introduces David as refundee. Merchant creates a few refund transactions and mixed their outputs and send them to refundees.

merchant and the private key of the customer. For secure mixing, the time relationship between the input and the output of the mix must be protected. Otherwise an adversary who intercepts the merchant's channel can link the two using the time information of the merchant input and output. In the following protocol the merchant mixes the refunds of different customers and hides the time relation between inputs and outputs of the mix service.

Key generation. We assume that Refundee is using BIP32 [19] wallets, and sends their extended public key to the customer. Customer generates a private/public key pair as $Pk_c = cP$.

Click to pay. The customer visits the merchant website and chooses an item, then clicks on "pay".

Payment request. The merchant sends the payment request message including their public key, $Pk_m = mP$. This public key is unique for each transaction.

Payment message. After authenticating the merchant, the customer picks a public key, Pk_c, and generates *MainTC* which pays the cost of the chosen item to Merchant. Then, the customer creates a payment message with extended refund keys, $(Pk_{r_i}, ch_{r_i}), \forall 1 \leq i \leq n$, for n refund addresses, and the amount of refund value for each; then encrypts the *Payment.refund_to* field using Diffie-Hellman key $H^*(cPk_m)$ (this is not required if channel is TLS-protected).

Payment ack. The merchant detects *MainTC*, decrypts the refund addresses, and returns an acknowledgment message, *PaymentAck*, to customer.

Refund request. Within a predetermined distance from payment request (can be defined in *Payment.refund_to* field), customer can use the addresses provided in $refund\_to$ field to receive the refund. In this case, the merchant

1. Splits each refund value (for different customers) to k partition; for example, v_1 is divided to $v_{11}, v_{12}, \ldots v_{1k}$, and v_2 to $v_{21}, v_{22}, \ldots v_{2k}$ and so on. The goal is to have equal amounts.
2. Derives child keys of each refund address, $Pk'_{r_{ij}}$ $\forall 1 \leq i \leq n$, and $\forall 1 \leq j \leq k$.

3. Creates a few transactions. For each child key merchant creates an output that pays its chunk of refund value to the corresponding masked child key $Pk''_{r_{ij}} = Pk'_{r_{ij}} + H^*(m^* Pk'_{r_{ij}})$, $\forall 1 \leq i \leq n$, and $\forall 1 \leq j \leq k$. While mixing the outputs of different customers in each transaction, the merchant broadcasts the transactions to the Bictoin network.

5 Concluding Remarks

We proposed a new approach to mitigate Refund attacks against BIP70 using implicit logging which requires the merchant to only store indexes of four transactions for each run of the protocol. Our approach provides a solution that is robust against possible corruption of the merchant's local database, and preserves the privacy of the refund (i.e. the link between the customer and the refundee). We also showed that refund mechanism can be used to provide anonymity for payers and payees, through merchant acting as a mix server, and provided the communication protocol between the customer and the mixing service based on BIP70. This is a novel approach for providing anonymity for bitcoin transaction that need careful evaluation. This will be an interesting direction for future work.

References

1. Andresen, G., Hearn, M.: BIP 70, July 2013. https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki. Accessed Feb 2017
2. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 34–51. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_4
3. Avizheh, S., Safavi-Naini, R., Shahandashti, S.F.: A new look at the refund mechanism in the bitcoin payment protocol (2018). https://arxiv.org/abs/1807.01793
4. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better — how to make bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_29
5. bitcoinwiki: Address reuse, April 2017. https://en.bitcoin.it/wiki/Address_reuse. Accessed May 2017
6. bitmixer: High volume bitcoin mixer (2014). https://bitmixer.io/. Accessed Sept 2017
7. BitPay: Can bitpay refund my order? (2015). https://support.bitpay.com/hc/en-us/articles/203411523-Can-BitPay-refund-my-order-. Accessed Feb 2017
8. Coinbase: How can i refund a customer with the API? (2015). https://support.coinbase.com/customer/en/portal/articles/1521752-how-can-i-refund-a-customer-with-the-api-. Accessed Feb 2017
9. Cuthbertson, A.: Bitcoin now accepted by 100,000 merchants worldwide, February 2015. http://www.ibtimes.co.uk/bitcoin-now-accepted-by-100000-merchants-worldwide-1486613. Accessed Mar 2017
10. Das, S.: 6,000 South Korean outlets to make cryptocurrencies available by Q2 2018, March 2018. https://www.ccn.com/6000-south-korean-outlets-to-make-cryptocurrencies-available-by-q2-2018. Accessed Apr 2018

11. Helms, K.: Bitcoin to be accepted at 260,000 stores in Japan by this summer, April 2017. https://news.bitcoin.com/bitcoin-accepted-260000-stores-summer. Accessed Apr 2018
12. Maxwell, G.: CoinJoin: bitcoin privacy for the real world (2013). https://bitcointalk.org/index.php
13. Maxwell, G.: CoinSwap: transaction graph disjoint trustless trading, October 2013
14. McCorry, P., Shahandashti, S.F., Hao, F.: Refund attacks on bitcoin's payment protocol. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 581–599. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_34
15. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
16. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: Altshuler, Y., Elovici, Y., Cremers, A., Aharony, N., Pentland, A. (eds.) Security and Privacy in Social Networks, pp. 197–223. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-4139-7_10
17. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_2
18. Todd, P.: January 2014. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html. Accessed Mar 2017
19. Wuille, P.: February 2017. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki. Accessed Feb 2017

Anonymous Reputation Systems Achieving Full Dynamicity from Lattices

Ali El Kaafarani[1(✉)], Shuichi Katsumata[2], and Ravital Solomon[1]

[1] University of Oxford, Oxford, UK
{ali.elkaafarani,ravital.solomon}@maths.ox.ac.uk
[2] National Institute of Advanced Industrial Science and Technology (AIST), The University of Tokyo, Tokyo, Japan
shuichi_katsumata@it.k.u-tokyo.ac.jp

Abstract. In this work, we revisit the *Anonymous Reputation Systems* presented by Blömer et al. in (FC'15). An anonymous reputation system allows users to review/rate products that they have purchased. The main security guarantee that such systems ensure is *privacy*, *i.e.,* users are allowed to *anonymously* write reviews for any products which they have purchased. However, to avoid abuse/misuse cases, a *review-once-policy* is also enforced, *i.e.,* if a user tries to write a second review for the same product, his reviews will be publicly linkable. Therefore, the system manager can revoke this user from the system.

The contribution of this paper is threefold. First, we strengthen and re-formalize the security model for reputation systems of Blömer et al. so that it captures more accurately real-life threats. In particular, our security model captures all possible framing scenarios including when the adversary tries to produce a review that links to another review produced by an honest user. Without this security notion, an adversary can exploit this vulnerability in order to revoke or partially de-anonymize a particular user. Second, our reputation system is fully dynamic so that users and items can be added and revoked at any time. This is an attractive and should possibly be a default feature for reputations systems to have, since the system manager will not know the users/items in the time of setup of the system. Finally, we propose the first construction of a reputation system based on lattice assumptions that are conjectured to be resistant to quantum attacks by incorporating a lattice-based tag scheme.

1 Introduction

Since 2000, a tremendous effort has been made to improve the state-of-the-art of reputation systems[1], trying to build the best possible system that helps both consumers and sellers establish mutual trust on the internet. A reputation system allows users to anonymously rate or review products that they bought over the

[1] In this paper, we will use the terms reputation systems and anonymous reputation systems interchangeably.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 388–406, 2018.
https://doi.org/10.1007/978-3-662-58387-6_21

internet, which would help people decide what/whom to trust in this fast emerging e-commerce world. In 2000, Resnick et al. in their pioneering work [RKZF00] concluded their paper on reputation systems with an allusion to democracy. They envisioned what would Winston Churchill (British prime minister during WWII) comment on reputation systems as he did on democracy. They claim that he might say the following: *"Reputation systems are the worst way of building trust on the Internet, except for all those other ways that have been tried from time-to-time."* Sixteen years later, Zhai et al., in their interesting work [ZWC+16], are still asking the intriguing and challenging question; *"Can we build an anonymous reputation system?"* This clearly shows how challenging and difficult it is to build a useful, secure, and deployable reputation system.

Why reputation systems? Because they simulate what used to happen before the internet era; people used to make decisions on what to buy and from whom, based on personal and corporate reputations. However, on the internet, users are dealing with total strangers, and reputation systems seem to be a suitable solution for building trust while maintaining privacy. Without a doubt, privacy has become a major concern for every internet user. Consumers want to rate products that they buy on the internet and yet keep their identities hidden. This is not merely paranoia; Resnick and Zeckhauser showed in [RZ02] that sellers on eBay discriminate against potential customers based on their review history. This discrimination could take the form of *"Sellers providing exceptionally good service to a few selected individuals and average service to the rest"*, as stated in [Del00]. Therefore, anonymity seems to be the right property for a reputation system to have. However, on the other hand, we cannot simply fully anonymize the reviews, since otherwise malicious users can for example create spam reviews for the purpose of boosting/reducing the popularity of specific products, thus defeating the purpose of a reliable reputation system. Therefore, reputation systems must also enforce *public linkability*, *i.e.,* if any user misuses the system by writing multiple reviews or rating multiple times on the same product, he will be detected, and therefore revoked from the system.

Different cryptographic tools have been used to realize reputation systems, including *Ring Signatures* (*e.g.,* [ZWC+16]), *Signatures of Reputations* (*e.g.,* [BSS10]), *Group Signatures* (*e.g.,* [BJK15]), *Blockchains* (*e.g.,* [SKCD16]), *Mix-Nets* (*e.g.,* [ZWC+16]), *Blind Signatures* (*e.g.,* [ACBM08]), etc., each of which improves on one or multiple aspects of reputation systems that are often complementary and incomparable. Other relevant works include a long line of interesting results presented in [Del00, JI02, KSGM03, DMS03, Ste06, ACBM08, Ker09, YSK+09, GK11, VMG+12, CSK13, MKKSM13, MK14].

Why Group Signatures. In this work, we choose to move forward and strengthen the state-of-the-art of reputation systems built from group signatures presented in [BJK15] in three orthogonal directions (see details in next paragraph). Undeniably, group signatures are considered to be one of the most well-established type of anonymous digital signatures, with a huge effort being made to generically formalize such an intriguing tool

(see for instance, [CVH91, Cam97, AT99, BMW03, BBS04, BS04, CG04, BSZ05, BW06, BCC+16, LNWX17]), and therefore, building a reputation system from group signatures seems one of the advanced and safe options.

Although anonymous reputation systems share some of their security properties with group signatures, they do have their unique setting that requires a different and more challenging security model. For instance, a unique security property that is required by reputation systems is *public-linkability*; adding public-linkability will surely affect the way we would define the *anonymity* and *non-frameability* properties. For example, public-linkability can be easily seen to harm the standard anonymity notion for group signatures. Furthermore, a new framing threat arises when using any linking technique within an anonymous system (see details in Sect. 3.2). Therein lie the main subtleties of reputation systems' design, and that is why it has been difficult to define an acceptable security model for such systems so far even though reputation systems have been a hot topic for the last decade and one of the most promising applications of anonymous digital signatures.

Contribution. In our work, we substantially boost the line of work of reputation systems built from group signatures by providing a reputation system that affirmatively addresses three main challanges simultanouesly; namely, we give a rigorous security model, achieve full dynamicity (*i.e.,* users can join and leave at any moment), and equip this important topic with an alternative construction to be ready for the emerging post-quantum era. In more details, we first strengthen and re-formalize the security model for anonymous reputation systems presented in [BJK15] to fully capture all the real-life threats. In particular, we identify an essential security notion[2] uncalled in the presentation of [BJK15]; we capture and formalize the framing scenario where the adversary tries to produce a review that links to another review produced by an honest user. We believe this to be one of the central security notions to be considered in order to maintain a reliable anonymous reputation system, as an adversary otherwise can exploit this vulnerability for the purpose of revoking or partially de-anonymizing a particular user. Also, our security model captures the notion of tracing soundness. It is indeed an important security property as it ensures that even if *all* parties in the system are fully corrupt, no one but the actual reviewer/signer can claim authorship of the signature. Additionally, in our security model, we are able to put less trust in the managing authorities, namely, the tracing manager does not necessarily have to be honest as is the case with [BJK15]. Second, our reputation system is *fully dynamic* where users/items can be added and revoked at any time. This is an attractive and should possibly be

[2] We would like to emphasize that the scheme of [BJK15] is secure according to their formalization, and we do not claim their scheme to be wrong in their proposed security model. We view one of our contribution as identifying a security hole which was not captured by the previous security model for reputation systems [BJK15], and providing a more complete treatment of them by building on the ideas of the most up-to-date security model for group signatures [BCC+16].

a default feature for a reputation system to have, due to its dynamic nature, *i.e.,* the system manager will not have the full list of users and items that will be participating in the system upon the setup of the system. Finally, we give a construction of a reputation system that is secure w.r.t our strong security model based on lattice assumptions. To the best of our knowledge, this is the first reputation system that relies on non number-theoretic assumptions, and thereby not susceptible to quantum attacks.

2 Preliminaries

2.1 Lattices

For positive integers n, m such that $n \leq m$, an integer n-dimensional lattice Λ in $\mathbb{Z}^m$ is a set of the form $\{\sum_{i\in[n]} x_i \mathbf{b}_i | x_i \in \mathbb{Z}\}$, where $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ are n linearly independent vectors in $\mathbb{Z}^m$. Let $D_{\mathbb{Z}^m,\sigma}$ be the discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter $\sigma > 0$. In the following, we recall the definition of the Short Integer Solution (SIS) problem and the Learning with Errors (LWE) problem.

Definition 1 (SIS). *For integers $n = n(\lambda), m = m(n), q = q(n) > 2$ and a positive real β, we define the* short integer solution *problem $\mathsf{SIS}_{n,m,q,\beta}$ as the problem of finding a vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = \mathbf{0} \mod q$ and $\|\mathbf{x}\|_\infty \leq \beta$ when given $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ as input.*

When $m, \beta = \mathsf{poly}(n)$ and $q > \sqrt{n}\beta$, the $\mathsf{SIS}_{n,m,q,\beta}$ problem is at least as hard as SIVP_γ for some $\gamma = \beta \cdot \tilde{O}(\sqrt{nm})$. See [GPV08, MP13].

Definition 2 (LWE). *For integers $n = n(\lambda), m = m(n), t = t(n)$, a prime integer $q = q(n) > 2$ such that $t < n$ and an error distribution over $\chi = \chi(n)$ over $\mathbb{Z}$ we define the decision* learning with errors problem $\mathsf{LWE}_{n,m,q,\chi}$ *as the problem of distinguishing between $(\mathbf{A}, \mathbf{A}^\top\mathbf{s} + \mathbf{x})$ from $(\mathbf{A}, \mathbf{b})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \leftarrow \chi^n$, $\mathbf{x} \leftarrow \chi^m$ and $\mathbf{b} \leftarrow \mathbb{Z}_q^m$. We also define the search* first-are-errorless learning with errors problem $\mathsf{faeLWE}_{n,t,m,q,\chi}$ *as the problem of finding a vector $\mathbf{s} \in \mathbb{Z}_q^n$ when given $\mathbf{b} = \mathbf{A}^\top\mathbf{s} + \mathbf{x} \mod q$ as input, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \leftarrow \chi^n$ and $\mathbf{x} \leftarrow \{0\}^t \times \chi^{m-t}$, i.e., the first t samples are noise-free.*

[ACPS09] showed that one can reduce the standard LWE problem where $\mathbf{s}$ is sampled from $\mathbb{Z}_q^n$ to the above LWE problem where the secret is distributed according to the error distribution. Furthermore, [ALS16] showed a reduction from $\mathsf{LWE}_{n-t,m,q,\chi}$ to $\mathsf{faeLWE}_{n,t,m,q,\chi}$ that reduces the advantage by at most 2^{n-t-1}. When $\chi = D_{\mathbb{Z},\alpha q}$ and $\alpha q > 2\sqrt{2n}$, the $\mathsf{LWE}_{n,m,q,\chi}$ is at least as (quantumly) hard as solving SIVP_γ for some $\gamma = \tilde{O}(n/\alpha)$. See [Reg05, Pei09, BLP+13]. We sometimes omit the subscript m from $\mathsf{LWE}_{n,m,q,\chi}, \mathsf{faeLWE}_{n,m,q,\chi}$, since the hardness of the problems hold independently from $m = \mathsf{poly}(n)$. In the following, in case $\chi = D_{\mathbb{Z},\beta}$, we may sometimes denote $\mathsf{LWE}_{n,m,q,\beta}, \mathsf{faeLWE}_{n,m,q,\beta}$.

2.2 Tag Schemes

We recall here the lattice-based *linkable indistinguishable tag* (LWE-LIT) scheme presented in [EE17]. Let m, ω, q be positive integers with $m = 3\omega$ and $q > 2$ a prime. Assume they are all implicitly a polynomial function of the security parameter n, where we provide a concrete parameter selection in our construction (see Sect. 4). Let $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q^{m \times \omega}$ be a hash function modeled as a random oracle in the security proofs. Let $\mathcal{K} = \mathbb{Z}_q^m \cap [-\beta, \beta]^m$ be the key space for some positive integer $\beta < q$, $\mathcal{T} = \mathbb{Z}_q^m$ be the tag space, and $\mathcal{I} = \{0,1\}^*$ be the message space. Finally, let β' be some positive real such that $\beta > \beta' \omega(\sqrt{\log n})$. Then, the lattice-based linkable indistinguishable tag scheme is defined by the following three PPT algorithms $\mathsf{LIT} = (\mathsf{KeyGen}_{\mathsf{LIT}}, \mathsf{TAG}_{\mathsf{LIT}}, \mathsf{Link}_{\mathsf{LIT}})$:

$\mathsf{KeyGen}_{\mathsf{LIT}}(1^n)$: The *key generation* algorithm takes as input the security parameter 1^n, it samples a secret key $\mathsf{sk} \leftarrow \mathcal{D}_{\mathbb{Z}^m, \beta'}$ until $\mathsf{sk} \in \mathcal{K}$.[3] It then outputs sk.

$\mathsf{TAG}_{\mathsf{LIT}}(I, \mathsf{sk})$: The *tag generation* algorithm takes as input a message $I \in \mathcal{I}$ and a secret key $\mathsf{sk} \in \mathcal{K}$, and samples an error vector $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^\omega, \beta'}$. It then outputs a tag $\tau = \mathcal{H}(I)^\top \mathsf{sk} + \mathbf{e} \in \mathcal{T}$.

$\mathsf{Link}_{\mathsf{LIT}}(\tau_0, \tau_1)$: The *linking* algorithm takes as input two tags τ_0, τ_1, and outputs 1 if $\|\tau_0 - \tau_1\|_\infty \leq 2\beta$ and 0 otherwise.

We require one additional algorithm only used during the security proof.

$\mathsf{IsValid}_{\mathsf{LIT}}(\tau, \mathsf{sk}, I)$: This algorithm takes as input a tag τ, a secret key sk and a message I, and outputs 1 if $\|\tau - \mathcal{H}(I)^\top \mathsf{sk}\|_\infty \leq \beta$ and 0 otherwise.

The tag scheme (LIT) must satisfy two security properties, namely, the tag-indistinguishability and linkability. Informally speaking, tag-indistinguishability ensures that an adversary $\mathcal{A}$ cannot distinguish between two tags produced by two users (of his choice) even given access to a tag oracle. Linkability means that two tags must "link" together if they are produced by the same user on the same message. In the context of reputation systems, the messages associated to the tag will correspond to the items that the users buy. Therefore, when the users write two anonymous reviews on the same item, the tags will help us link the two reviews.

Tag-Indistinguishability. A tag-indistinguishability for a LIT scheme is defined by the experiment in Fig. 1 and a tag oracle as in Fig. 2. We define the advantage of an adversary $\mathcal{A}$ breaking the tag-indistinguishability as follows:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Tag}}(n) = \left| \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Tag},0}(n) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Tag},1}(n) = 1] \right|$$

We say that a LIT scheme is tag-indistinguishable if for all polynomial time adversary $\mathcal{A}$ the advantage is negligible.

The proof of the following Theorem 1 will be provided in the full version.

[3] The expected number of samples required will be a constant due to our parameter selection. In particular, we have $\Pr[|x| > \beta' \omega(\sqrt{\log n})] = \mathsf{negl}(n)$ for $x \leftarrow D_{\mathbb{Z}, \sqrt{2}\beta'}$.

Experiment: $\mathsf{Exp}^{\mathsf{Tag},b}_{\mathcal{A}}(n)$

$\mathsf{sk}_j \leftarrow \mathsf{KeyGen}_{\mathsf{LIT}}(1^n)$ for $j = 0, 1$.
$V_0, V_1 \leftarrow \emptyset$
$(I^*, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Tag}}(\cdot,\cdot),\mathcal{H}(\cdot)}(1^n)$
$\tau^* \leftarrow \mathsf{TAG}_{\mathsf{LIT}}(I^*, \mathsf{sk}_b)$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Tag}}(\cdot,\cdot),\mathcal{H}(\cdot)}(\tau^*, \mathsf{st})$
If either $(0, I^*)$ or $(1, I^*)$ was submitted to $\mathcal{O}_{\mathsf{Tag}}$ **return** 0
If $b^* = b$ **return** 1, **else return** 0

Fig. 1. Tag-indistinguishability

Oracle: $\mathcal{O}_{\mathsf{Tag}}(j, I)$

If $j \notin \{0, 1\}$ **return** $\perp$
If $\exists \tau$ such that $(I, \tau) \in V_j$ **return** τ
Else $\tau \leftarrow \mathsf{TAG}_{\mathsf{LIT}}(I, \mathsf{sk}_j)$
$V_j \leftarrow V_j \cup \{(I, \tau)\}$
return τ

Fig. 2. Description of the tag oracle

Theorem 1 (tag-indistinguishability). *For any efficient adversary $\mathcal{A}$ against the tag-indistinguishability experiment of the* LWE-LIT *scheme as defined above, we can construct an efficient algorithm $\mathcal{B}$ solving the* $\mathsf{LWE}_{m,\omega Q,q,\beta'/\sqrt{2}}$ *problem with advantage:*

$$\mathsf{Adv}^{\mathsf{LWE}_{m,\omega Q,q,\beta'/\sqrt{2}}}_{\mathcal{B}}(n) \geq \mathsf{Adv}^{\mathsf{Tag}}_{\mathcal{A}}(n) - \mathsf{negl}(n),$$

where Q denotes the number of random oracle queries made by $\mathcal{A}$. In particular, assuming the hardness of $\mathsf{LWE}_{m,\omega Q,q,\beta'/\sqrt{2}}$*, the advantage of any efficient adversary $\mathcal{A}$ is negligible.*

Linkability. A linkability of a LIT scheme is defined by the experiment in Fig. 3. We define the advantage of an adversary $\mathcal{A}$ breaking the linkability as $\mathsf{Adv}^{\mathsf{Link}}_{\mathcal{A}}(n) = \Pr[\mathsf{Exp}^{\mathsf{Link}}_{\mathcal{A}}(n) = 1]$. We say that a LIT scheme is non-linkable if for all adversary $\mathcal{A}$ the advantage is negligible.

Experiment: $\mathsf{Exp}^{\mathsf{Link}}_{\mathcal{A}}(n)$

$(\tau_0, \tau_1, I, \mathsf{sk}) \leftarrow \mathcal{A}^{\mathcal{H}(\cdot)}(1^n)$
If $\mathsf{IsValid}_{\mathsf{LIT}}(\tau_b, \mathsf{sk}, I) = 1$ for $b \in \{0, 1\}$ and $\mathsf{Link}_{\mathsf{LIT}}(\tau_0, \tau_1) = 0$ **return** 1
Else return 0

Fig. 3. Linkability

Theorem 2 (Linkability). *For* any *adversary $\mathcal{A}$ against the linkability experiment of the* LWE-LIT *scheme as defined above, the advantage* $\mathsf{Adv}^{\mathsf{Link}}_{\mathcal{A}}(n)$ *is negligible.*

Proof. Suppose, towards a contradiction, that an adversary $\mathcal{A}$ wins the linkability experiment. In particular, $\mathcal{A}$ outputs $(\tau_0, \tau_1, I, \mathsf{sk})$ such that the following three conditions hold: $\|\tau_0 - \mathcal{H}(I)^\top \mathsf{sk}\| \leq \beta$, $\|\tau_1 - \mathcal{H}(I)^\top \mathsf{sk}\| \leq \beta$, and $\|\tau_0 - \tau_1\| > 2\beta$. From the first two inequalities, we have

$$\begin{aligned}\|\tau_0 - \tau_1\| &= \|(\tau_0 - \mathcal{H}(I)^\top \mathsf{sk}) + (-\tau_1 + \mathcal{H}(I)^\top \mathsf{sk})\| \\ &\leq \|\tau_0 - \mathcal{H}(I)^\top \mathsf{sk}\| + \|\tau_1 - \mathcal{H}(I)^\top \mathsf{sk}\| \leq 2\beta,\end{aligned}$$

by the triangular inequality. However, this contradicts the third inequality.

2.3 Group Signatures

In a group signature, a group member can anonymously sign on behalf of the group, and anyone can then verify the signature using the group's public key without being able to tell which group member signed it. A group signature has a group manager who is responsible for generating the signing keys for the group members. There are two types of group signatures: the static type and the dynamic type. In the static type [BMW03], the group members are fixed at the setup phase. In this case, the group manager can additionally trace a signature and reveal which member has signed it. In the dynamic type [BSZ05, BCC+16], users can join/leave the system at anytime. Now a group has two managers; the group manager and a separate tracing manager who can open signatures in case of misuse/abuse. Briefly speaking, a group signature has three main security requirements; *anonymity*, *non-frameability*, and *traceability*. Anonymity ensures that an adverary cannot tell which group member has signed the message given the signature. Non-frameability ensures that an adversary cannot produce a valid signature that traces back to an honest user. Finally, traceability ensures that an adversary cannot produce a valid signature that does not trace to an user.

In our work, we build on the recent lattice-based fully dynamic group signature scheme of [LNWX17] to construct our reputation system. We briefly sketch how the group signature scheme of [LNWX17] works; a group manager maintains a Merkle-tree in which he stores members' public keys in the leaves where the exact position are given to the signers at join time. The leaves will be hashed to the top of the tree using an accumulator instantiated using a lattice-based hash function. The relevant path to the top of the tree will be given to each member where the top of the tree itself is public. In order to sign, a group member has to prove in zero-knowledge that; first, he knows the pre-image of a public key that has been accumulated in the tree, and that he also knows of a path from that position in the tree to its root. Additionally, they apply the Naor-Yung double-encryption paradigm [NY90] with Regev's LWE-based encryption scheme [Reg05] to encrypt the identity of the signer (twice) w.r.t the tracer's public key to prove anonymity. To summarize, a group signature would be of the form (Π, c_1, c_2), where Π is the zero-knowledge proof that the signer is indeed a member of the group (*i.e.,* his public key has been accumulated into the Merkle-tree), and the encrypted identity in both c_1 and c_2 is a part of the path that he uses to get to the root of the Merkle-tree. Note that this implies that the ciphertexts (c_1, c_2) are bound to the proof Π.

3 Syntax and Security Definitions

We formalize the syntax of reputation systems following the sate-of-the-art formalization of dynamic group signatures of [BCC+16]. We briefly explain the two major differences that distinguish between a reputation system from a group signature scheme. First, a reputation system is in essence a *group of group signature schemes* run in parallel, where we associate each item uniquely to one instance of the group signature scheme. Second, we require an additional algorithm Link in order to publicly link signatures (i.e., reviews), which is the core functionality provided by reputation systems. We now define reputation systems by the following PPT algorithms:

$\mathsf{RepSetup}(1^n) \to \mathsf{pp}$: On input of the security parameter 1^n, the setup algorithm outputs public parameters pp.

$\mathsf{KeyGen}_{\mathsf{GM}}(\mathsf{pp}) \leftrightarrow \mathsf{KeyGen}_{\mathsf{TM}}(\mathsf{pp})$: This an interactive protocol between the group manager GM and the tracing manager TM. If completed successfully, $\mathsf{KeyGen}_{\mathsf{GM}}$ outputs the GM's key pair $(\mathsf{mpk}, \mathsf{msk})$ and $\mathsf{KeyGen}_{\mathsf{TM}}$ outputs the TM's key pair $(\mathsf{tpk}, \mathsf{tsk})$. Set the system public key to be $\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$.

$\mathsf{UKgen}(1^n) \to (\mathsf{upk}, \mathsf{usk})$: On input of the security parameter 1^n, it outputs a key pair $(\mathsf{upk}, \mathsf{usk})$ for a user. We assume that the key table containing the various users' public keys upk is publicly available.

$\mathsf{Join}(\mathsf{info}_{t_{\mathsf{current}}}, \mathsf{gpk}, \mathsf{upk}, \mathsf{usk}, \mathsf{item}) \leftrightarrow \mathsf{Issue}(\mathsf{info}_{t_{\mathsf{current}}}, \mathsf{msk}, \mathsf{upk}, \mathsf{item})$: This is an interactive protocol between a user upk and the GM. Upon successful completion, the GM issues an identifier $\mathsf{uid}_{\mathsf{item}}$ associated with item to the user who then becomes a member of the group that corresponds to item[4]. The final state of the Issue algorithm, which would always include the user public key upk, is stored in the user registration table reg at index $(\mathsf{item}, \mathsf{uid}_{\mathsf{item}})$ which is made public. Furthermore, the final state of the Join algorithm is stored in the secret group signing key $\mathsf{gsk}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}]$.

$\mathsf{RepUpdate}(\mathsf{gpk}, \mathsf{msk}, R, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{reg}) \to (\mathsf{info}_{t_{\mathsf{new}}}, \mathsf{reg})$: This algorithm is run by the GM to update the system info. On input of the group public key gpk, GM's secret key msk, a list R of active users' public keys to be revoked, the current system info $\mathsf{info}_{t_{\mathsf{current}}}$, and the registration table reg, it outputs a new system info $\mathsf{info}_{t_{\mathsf{new}}}$ while possibly updating the registration table reg. If no changes have been made, output $\perp$.

$\mathsf{Sign}(\mathsf{gpk}, \mathsf{gsk}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}], \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M}) \to \Sigma$: On input of the system's public key gpk, user's group signing key $\mathsf{gsk}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}]$, system info $\mathsf{info}_{t_{\mathsf{current}}}$ at epoch t_{current}, an item, and message M, it outputs a signature Σ. If the user owning $\mathsf{gsk}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}]$ is not an active member at epoch t_{current}, the algorithm outputs $\perp$.

[4] Here our syntax assumes that the items to be reviewed have been already communicated to the GM from the respective service providers. We merely do this to make our presentation simple and we emphasize that our construction is general in the sense that the GM does not need to know neither the number of items nor the items themselves ahead of time. Items can dynamically be added/removed from the system by GM when it is online.

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M}, \Sigma) \rightarrow 1/0$: On input of the system's public key gpk, system info $\mathsf{info}_{t_{\mathsf{current}}}$, an item, a message M, and a signature Σ, it outputs 1 if Σ is valid signature on M for item at epoch t_{current}, 0 otherwise.

$\mathsf{Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{reg}, \mathsf{item}, \mathsf{M}, \Sigma) \rightarrow (\mathsf{uid}_{\mathsf{item}}, \Pi_{\mathsf{Trace}})$: On input of the system's public key gpk, the TM's secret key tsk, the system information $\mathsf{info}_{t_{\mathsf{current}}}$, the user registration table reg, an item, a message M, and a signature Σ, it outputs the identifier of the user $\mathsf{uid}_{\mathsf{item}}$ who produced Σ and a proof Π_{Trace} that attests to this fact. If the algorithm cannot trace the signature to a particular group member, it returns $\perp$.

$\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}_{\mathsf{item}}, \Pi_{\mathsf{Trace}}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M}, \Sigma) \rightarrow 1/0$: On input of the system's public key gpk, a user's identifier $\mathsf{uid}_{\mathsf{item}}$, a tracing proof Π_{Trace} from the Trace algorithm, the system info $\mathsf{info}_{t_{\mathsf{current}}}$, an item, a message M and signature Σ, it outputs 1 if Π_{Trace} is a valid proof that $\mathsf{uid}_{\mathsf{item}}$ produced Σ and 0 otherwise.

$\mathsf{Link}(\mathsf{gpk}, \mathsf{item}, (m_0, \Sigma_0), (m_1, \Sigma_1)) \rightarrow 1/0$: On input of the system's public key gpk, an item, and two message-signature pairs, it returns 1 if the signatures were produced by the same user on behalf of the group that corresponds to item, 0 otherwise.

$\mathsf{IsActive}(\mathsf{info}_{t_{\mathsf{current}}}, \mathsf{uid}_{\mathsf{item}}, \mathsf{reg}, \mathsf{item}) \rightarrow 1/0$: this algorithm will only be used in the security games. On input of the system $\mathsf{info}_{t_{\mathsf{current}}}$, a user's identifier $\mathsf{uid}_{\mathsf{item}}$, the user registration table reg, and an item, it outputs 1 if $\mathsf{uid}_{\mathsf{item}}$ is an active member of the group for item at epoch t_{current} and 0 otherwise.

3.1 Discussion on the Security Model of FC'15 Reputation System

Blömer et al. [BJK15] constructed an anonymous reputation system from group signatures based on number-theoretical assumptions. In their work, they claim to formalize reputation systems following the formalization of partially dynamic group signature schemes presented by Bellare et al. [BSZ05], *i.e.,* they have two managers, the *group manger* and *key issuer*[5]. However, one can notice that the security model is in fact strictly weaker than that of [BSZ05]; the major difference being the assumption that the opener/tracer is *always* honest. Furthermore, in their public-linkability property, the key issuer (the GM in our case) is assumed to be honest. Another observation, which we believe to be of much bigger concern, is that their security notion for reputation systems does not fully capture all the real-life threats. In particular, their strong-exculpability property (which is essentially the notion of non-frameability), does not capture the framing scenario where the adversary outputs a signature that *links* to an honest user; it only captures the scenario where the adversary outputs a signature that traces to an honest user. Note that the former attack scenario does not exist in the context of group signatures since no tag schemes are being used there, *i.e.,* the whole notion of linkability does not exist. However, it is a vital security requirement in the reputation system context as an adversary could try to generate a review that links to an honest user's review so that the GM may decide to revoke or

[5] Note that [BJK15] does not completely follow the notation used in [BSZ05], *i.e.,* their group manager is in fact the tracer in [BSZ05].

de-anonymize the honest user. In our work, we provide a formal definition of reputation systems that models more accurately these real-life threats, which in particular, solve the aforementioned shortcomings of [BJK15].

3.2 Security Definitions

We provide a formal security definition following the experiment type definition of [BCC+16, LNWX17] for fully dynamic group signatures, which originates to [BSZ05]. Anonymity, non-frameability and public-linkability are provided in Fig. 4, whereas the rest of the security experiment together with the oracles used therein are provided in the full version of the paper. One of the main differences between theirs and ours is that, we require the *public-linkability* property, which does not exist in the group signature setting. Moreover, the existence of the tag scheme further affects the anonymity and non-frameability properties, which are depicted in Fig. 4; for the former, an adversary should not be allowed to ask for signatures by the challenge users on the challenge item, otherwise he could trivially win the game by linking the signatures. In the latter, an additional attack scenario is taken into consideration, *i.e.,* when an adversary outputs a review that links to an honest user's review.

In our formalization, we only require TM to be honest in the anonymity experiment, which is inevitable as otherwise the adversary could trivially win the game. Also, our public linkability holds unconditionally, and therefore, GM can be assumed to be corrupt there. We now present the security properties of our reputation system.

Correctness. A reputation system is correct if reviews produced by honest, non-revoked users are always accepted by the Verify algorithm and if the honest tracing manager can always identify the signer of such signatures where his decision will be accepted by a Judge. Additionally, two reviews produced by the same user on the same item should always link.

Anonymity. A reputation system is anonymous if for any PPT adversary the probability of distinguishing between two reviews produced by any two honest signers is negligible even if the GM and all other users are corrupt, and the adversary has access to the Trace oracle.

Non-frameability. A reputation system is non-frameable if for any PPT adversary it is unfeasible to generate a valid review that traces or links to an honest user even if it can corrupt all other users and chose the keys for GM and TM.

Traceability. A reputation system is traceable if for any PPT adversary it is unfeasible to produce a valid review that cannot be traced to an active user at the chosen epoch, even if it can corrupt any user and can choose the key of TM[6].

[6] The group manager GM is assumed to be honest in this game as otherwise the adversary could trivially win by creating dummy users.

Experiment: $\mathsf{Exp}^{\mathsf{Anon}\text{-}b}_{\mathsf{rep\text{-}sys},\mathcal{A}}(n)$

$\mathsf{pp} \leftarrow \mathsf{RepSetup}(1^n)$; $\mathsf{HUL}, \mathsf{CUL}, \mathsf{BUL}, \mathsf{SL}, \mathsf{CL} := \emptyset$
$(\mathsf{st}, \mathsf{info}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathcal{A}^{(\cdot \leftrightarrow \mathsf{KeyGen}_{\mathsf{TM}}(\mathsf{pp}))}(\mathsf{pp})$
if $\mathsf{KeyGen}_{\mathsf{TM}}$ did not accept or $\mathcal{A}$'s output is not well-formed, **return 0**
$\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$b^* \leftarrow \mathcal{A}^{\mathsf{AddU},\mathsf{CrptU},\mathsf{SndToU},\mathsf{RevealU},\mathsf{Trace},\mathsf{MReg},\mathsf{Chal}_\mathsf{b},\mathsf{Sign}}(\mathsf{st}, \mathsf{gpk})$
if $|\mathsf{CL}| \neq 1$ **return** 0, otherwise, let $\mathsf{CL} = \{(\mathsf{uid}_0, \mathsf{uid}_1, \mathsf{item}^*, \mathsf{M}, \Sigma)\}$
if $(-, \mathsf{uid}_b, -, \mathsf{item}^*, -, -) \in \mathsf{SL}$ for $b = 0$ or 1, **return** 0
else return b^*

Experiment: $\mathsf{Exp}^{\mathsf{non\text{-}frame}}_{\mathsf{rep\text{-}sys},\mathcal{A}}(n)$

$\mathsf{pp} \leftarrow \mathsf{RepSetup}(1^n)$; $\mathsf{HUL}, \mathsf{CUL}, \mathsf{BUL}, \mathsf{SL} := \emptyset$
$(\mathsf{st}, \mathsf{info}, \mathsf{msk}, \mathsf{mpk}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathcal{A}(\mathsf{pp})$
if $\mathcal{A}$'s output is not well-formed, **return** 0
Set $\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(\mathsf{uid}^*_{\mathsf{item}^*}, \Pi^*_{\mathsf{Trace}}, \mathsf{info}_{t^*}, \mathsf{item}^*, \mathsf{M}^*, \Sigma^*) \leftarrow \mathcal{A}^{\mathsf{CrptU},\mathsf{SndToU},\mathsf{RevealU},\mathsf{Sign},\mathsf{MReg}}(\mathsf{st}, \mathsf{gpk})$
$X \leftarrow \mathsf{RUser}(\mathsf{item}^*, \mathsf{uid}_{\mathsf{item}^*})$
if $X = \perp$ **return** 0, **else** $\mathsf{upk}^* := X$
if $\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_{t^*}, \mathsf{item}^*, \mathsf{M}^*, \Sigma^*) = 0$, **return** 0
if $\exists(\mathsf{upk}, \mathsf{uid}_{\mathsf{item}^*}, t, \mathsf{item}^*, \mathsf{M}, \Sigma) \in \mathsf{SL}$ s.t. $\mathsf{uid}_{\mathsf{item}^*} \in \mathsf{HUL}[\mathsf{upk}] \wedge \mathsf{upk} \notin \mathsf{BUL}$
$\wedge\ \mathsf{Link}(\mathsf{gpk}, \mathsf{item}^*, (\mathsf{M}^*, \Sigma^*), (\mathsf{M}, \Sigma)) = 1$, **return** 1
if $\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}^*_{\mathsf{item}^*}, \Pi^*_{\mathsf{Trace}}, \mathsf{info}_{t^*}, \mathsf{item}^*, \mathsf{M}^*, \Sigma^*) = 1 \wedge \mathsf{uid}^*_{\mathsf{item}^*} \in \mathsf{HUL}[\mathsf{upk}^*]$
$\wedge\ \mathsf{upk}^* \notin \mathsf{BUL} \wedge (\mathsf{upk}^*, \mathsf{uid}^*_{\mathsf{item}^*}, t^*, \mathsf{item}^*, \mathsf{M}^*, \Sigma^*) \notin \mathsf{SL}$, **return** 1
else return 0

Experiment: $\mathsf{Exp}^{\mathsf{Public\text{-}Link}}_{\mathsf{rep\text{-}sys},\mathcal{A}}(n)$

$\mathsf{pp} \leftarrow \mathsf{RepSetup}(1^n)$; $\mathsf{CUL} := \emptyset$
$(\mathsf{st}, \mathsf{info}, \mathsf{msk}, \mathsf{mpk}, \mathsf{tsk}, \mathsf{tpk}) \leftarrow \mathcal{A}(\mathsf{pp})$
if $\mathcal{A}$'s output is not well-formed, **return** 0
Set $\mathsf{gpk} := (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$
$(\mathsf{item}, \mathsf{uid}_{\mathsf{item}}, \mathsf{info}_t, \{(\mathsf{M}_b, \Sigma_b, \Pi_{\mathsf{Trace},b})\}_{b=0,1}) \leftarrow \mathcal{A}^{\mathsf{CrptU},\mathsf{MReg}}(\mathsf{st}, \mathsf{gpk})$
if $\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_t, \mathsf{item}, \mathsf{M}_b, \Sigma_b) = 0$ for $b = 0$ or 1, **return** 0
if $\mathsf{Link}(\mathsf{gpk}, \mathsf{item}, (\mathsf{M}_0, \Sigma_0), (\mathsf{M}_1, \Sigma_1)) = 1$, **return** 0
if $\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}_{\mathsf{item}}, \Pi_{\mathsf{Trace},b}, \mathsf{info}_t, \mathsf{item}, \mathsf{M}_b, \Sigma_b) = 0$ for $b = 0$ or 1, **return** 0
else return 1

Fig. 4. Security experiments for the reputation system-1

Public-Linkability. A reputation system is publicly linkable if for any (*possibly inefficient*) adversary it is unfeasible to output two reviews for the same item that trace to the same user but does not link. This should hold even if the adversary can chose the keys of GM and TM.

Tracing Soundness. A reputation system has tracing soundness if no (*possibly inefficient*) adversary can output a review that traces back to two different signers even if the adversary can corrupt all users and chose the keys of GM and TM.

4 Our Lattice-Based Reputation System

Intuition Behind Our Scheme. It is helpful to think of our reputation system as a *group of group signatures* managed by a global group manager (or call it a system manager), whom we refer to as a group manager GM for simplicity. This group manager shares the managerial role with the tracing manager TM who is only called for troubleshooting, *i.e.,* to trace users who misused the system. The group manager maintains a set of groups, each corresponding to a product/item owned by a certain service provider. Users who bought a certain item are eligible to become a member of the group that corresponds to that item, and can therefore write *one* anonymous review for that item. Every user in the system will have his own pair of public-secret key $(\mathsf{upk}, \mathsf{usk})$. When he wants to join the system for a particular item, he would engage in the $\mathsf{Join\text{-}Issue}$ protocol with GM, after which, he would be assigned a position $\mathsf{uid} = \mathsf{bin}(j) \in \{0,1\}^{\ell}$ in the Merkle-tree that corresponds to the item in question, and his public key will be accumulated in that tree. Here, j (informally) denotes the j-th unique user to have bought the corresponding item. The user can now get his witness w_j that attests to the fact that he is indeed a consumer of the item, on which he is then ready to write a review for that item. Technically speaking, he needs to provide a non-interactive zero-knowledge argument of knowledge for a witness to the following relation $\mathcal{R}_{\mathsf{Sign}}$:

$$\begin{aligned}\mathcal{R}_{\mathsf{Sign}} = \big\{&(\mathbf{A}, \mathbf{u}, \mathcal{H}_{\mathsf{Tag}}(\mathsf{item}), \tau, c_1, c_2, \mathbf{B}, \mathbf{P}_1, \mathbf{P}_2), (\mathbf{p}, w_j, \mathbf{x}, \mathbf{e}, \mathsf{uid}_{\mathsf{item}}, \mathbf{r}_1, \mathbf{r}_2) : \\ &\mathbf{p} \neq \mathbf{0}^{nk} \wedge \mathsf{TVerify}_{\mathbf{A}}(\mathbf{p}, w_j, \mathbf{u}) = 1 \wedge \mathbf{A} \cdot \mathbf{x} = \mathbf{G} \cdot \mathbf{p} \bmod q \\ &\wedge \big(\mathsf{Enc}_{\mathsf{Regev}}\big((\mathbf{B}, \mathbf{P}_1, \mathbf{P}_2), \mathsf{uid}_{\mathsf{item}}; (\mathbf{r}_1, \mathbf{r}_2)\big) = (c_1, c_2) \\ &\wedge \tau = \mathcal{H}_{\mathsf{Tag}}(\mathsf{item})^{\top}\mathbf{x} + \mathbf{e}\big\}.\end{aligned}$$

As can be seen, the signer encrypts his uid and computes a tag for the item in question. This tag ensures that he can only write one review for each item, otherwise his reviews will be publicly linkable and therefore detectable by GM. Regarding the verification, anyone can then check the validity of the signature by simply running the verify algorithm of the underlying NIZKAoK proof system. In any misuse/abuse situation, TM can simply decrypt the ciphertext attached to the signature to retrieve the identity of the signer. TM also needs to prove correctness of opening (to avoid framing scenarios) via the generation of a NIZKAoK for the following relation $\mathcal{R}_{\mathsf{Trace}}$:

$$\mathcal{R}_{\mathsf{Trace}} = \{(c_1, c_2, \mathsf{uid}_{\mathsf{item}}, \mathbf{B}, \mathbf{P}_1), (\mathbf{S}_1, \mathbf{E}_1) : \mathsf{Dec}_{\mathsf{Regev}}\big((\mathbf{S}_1, \mathbf{E}_1), (c_1, c_2)\big) = \mathsf{uid}_{\mathsf{item}}\}$$

Finally, for public linkability, we require that any two given signatures (Σ_0, Σ_1) for the same item can be publicly checked to see if they are linkable, *i.e.,* check that were produced by the same reviewer. This can be done simply by feeding the tags τ_0 and τ_1 of the two signatures, to the $\mathsf{Link}_{\mathsf{LIT}}$ algorithm of the underlying LIT scheme. If $\mathsf{Link}_{\mathsf{LIT}}$ returns 0, then Σ_0 and Σ_1 were not produced by the same user, and therefore are legitimate reviews from two different users. Otherwise, in the case it returns 0, we know that some user reviewed twice for the same item;

the GM asks TM to trace those signatures and find out who generated them and GM will then revoke the traced user from the system.

4.1 Our Construction

Underlying Tools. In our construction, we use the multi-bit variant of the encryption scheme of Regev [KTX07,PVW08], which we denote by $(\mathsf{KeyGen}_{\mathsf{Regev}}, \mathsf{Enc}_{\mathsf{Regev}}, \mathsf{Dec}_{\mathsf{Regev}})$. We also employ the lattice-based tag scheme $(\mathsf{KeyGen}_{\mathsf{LIT}}, \mathsf{TAG}_{\mathsf{LIT}}, \mathsf{Link}_{\mathsf{LIT}})$ provided in Sect. 2.2. We assume that both schemes share the same noise distribution χ (see below). We also use a lattice-based accumulator $(\mathsf{TSetup}, \mathsf{TAcc}_{\mathbf{A}}, \mathsf{TVerify}_{\mathbf{A}}, \mathsf{TUpdate}_{\mathbf{A}})$ [LNWX17]. Finally, we use a Stern-like zero-knowledge proof system where the commitment scheme of [KTX08] is used internally. More details on these building blocks can be found in the full version of the paper.

Construction. The proposed reputation system consists of the following PPT algorithms:

$\mathsf{RepSetup}(1^n)$: On input of the security parameter 1^n, it outputs the public parameters,

$$\mathsf{pp} = (N, n, q, k, m, m_E, \omega, \ell, \beta, \chi, \kappa, \mathcal{H}_{\mathsf{Tag}}, \mathcal{H}_{\mathsf{Sign}}, \mathcal{H}_{\mathsf{Trace}}, \mathbf{A}).$$

Where, $N = 2^\ell = \mathsf{poly}(n)$ is the number of potential users, $q = \mathcal{O}(n^{1.5}), k = \lceil \log_2 q \rceil, m = 2nk, m_E = 2(n+\ell)k, \omega = 3m, \beta = \sqrt{n} \cdot \omega(\log n)$, and a $\beta/\sqrt{2}$-bounded noise distribution χ. Moreover, $\mathcal{H}_{\mathsf{Tag}} : \{0,1\}^* \to \mathbb{Z}_q^{m \times \omega}$ is the hash function used for the tag scheme, and $\mathcal{H}_{\mathsf{Sign}}, \mathcal{H}_{\mathsf{Trace}} : \{0,1\}^* \to \{1,2,3\}^\kappa$ are two hash functions used for the NIZKAoK proof systems for $\mathcal{R}_{\mathsf{Sign}}$ and $\mathcal{R}_{\mathsf{Trace}}$, where $\kappa = \omega(\log n)$. Finally, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$.

$\mathsf{KeyGen}_{\mathsf{GM}}(\mathsf{pp}) \leftrightarrow \mathsf{KeyGen}_{\mathsf{TM}}(\mathsf{pp})$: This is for the group manager and tracing manager to set up their keys and publish the system's public information. The group manager samples $\mathsf{msk} \leftarrow \{0,1\}^m$, and sets $\mathsf{mpk} := \mathbf{A} \cdot \mathsf{msk} \bmod q$. On the other hand, TM runs $(\mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Regev}}(1^n)$ and sets $\mathsf{tpk} := \mathsf{pk}_{\mathsf{Enc}} = (\mathbf{B}, \mathbf{P}_1, \mathbf{P}_2)$ and $\mathsf{tsk} := \mathsf{sk}_{\mathsf{Enc}} = (\mathbf{S}_1, \mathbf{E}_1)$. GM receives tpk from TM and creates an empty reg table. Namely, $\mathsf{reg}[\mathsf{item}][\mathsf{bin}(j)][1] = \mathbf{0}^{nk}$ and $\mathsf{reg}[\mathsf{item}][\mathsf{bin}(j)][2] = 0$ for $j = 1, \cdots, N-1$ and all item in the system, *i.e.,* it is epoch 0 and no users have joined the system yet[7]. Here, GM maintains multiple local counters c_{item} to keep track of the registered users for each item, which are all set initially to 0. Finally, GM outputs $\mathsf{gpk} = (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$ and $\mathsf{info} = \emptyset$.

$\mathsf{UKgen}(1^n)$: This algorithm is run by the user. It samples $\mathbf{x} \leftarrow \mathsf{KeyGen}_{\mathsf{LIT}}(1^n)$ where $\mathbf{x} \in [-\beta, \beta]^m$ and sets $\mathsf{usk} := \mathbf{x}$. It then computes $\mathsf{upk} := \mathbf{p} = \mathsf{bin}(\mathbf{A}\mathbf{x} \bmod q) \in \{0,1\}^{nk}$. Hereafter, the user is identified by his public key upk.

[7] Recall that for simplicity of presentation, we assume the all items are provided to the GM. Our scheme is general enough so that the items can dynamically be added/removed from the system by GM.

$\mathsf{Join} \leftrightarrow \mathsf{Issue}$: A user $(\mathsf{upk}, \mathsf{usk}) = (\mathbf{p}, \mathbf{x})$ requests to join the group that corresponds to item at epoch t. He sends $\mathbf{p}$ to GM. If GM accepts the request, it issues an identifier for this user, *i.e.,* $\mathsf{uid}_{\mathsf{item}} = \mathsf{bin}(c_{\mathsf{item}}) \in \{0,1\}^{\ell}$. The user's signing key for item is then set to $\mathsf{gsk}[\mathsf{uid}_{\mathsf{item}}][\mathsf{item}] = (\mathsf{uid}_{\mathsf{item}}, \mathbf{p}, \mathbf{x})$. Now, GM updates the Merkle tree via $\mathsf{TUpdate}_{\mathsf{item},\mathbf{A}}(\mathsf{uid}_{\mathsf{item}}, \mathbf{p})$, and sets $\mathsf{reg}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}][1] := \mathbf{p}$, $\mathsf{reg}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}][2] := t$. Finally, it increments the counter $c_{\mathsf{item}} := c_{\mathsf{item}} + 1$.

$\mathsf{RepUpdate}(\mathsf{gpk}, \mathsf{msk}, R, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{reg})$: This algorithm is be run by GM. Given a set R of users to be revoked, it first retrieves all the $\mathsf{uid}_{\mathsf{item}}$ associated to each $\mathsf{upk} = \mathbf{p} \in R$. It then runs $\mathsf{TUpdate}_{\mathsf{item},\mathbf{A}}(\mathsf{reg}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}][1], \mathbf{0}^{nk})$ for all the retrieved $\mathsf{uid}_{\mathsf{item}}$. It finally recomputes $\mathbf{u}_{t_{\mathsf{new}},\mathsf{item}}$ and publishes

$$\mathsf{info}_{\mathsf{new}} = \left\{ \left(\mathbf{u}_{t_{\mathsf{new}},\mathsf{item}}, W_{\mathsf{item}} \right) \right\}_{\mathsf{item}},$$

where, $W_{\mathsf{item}} = \{w_{i,\mathsf{item}}\}_i$ and $w_{i,\mathsf{item}} \in \{0,1\}^{\ell} \times (\{0,1\}^{nk})^{\ell}$ is the witness that proves that $\mathsf{upk}_i = \mathbf{p}_i$ is accumulated in $\mathbf{u}_{t_{\mathsf{new}},\mathsf{item}}$. Here, the first ℓ-bit string term of the witness refers to the user identifier $\mathsf{uid}_{\mathsf{item}}$ associated to item.

$\mathsf{Sign}(\mathsf{gpk}, \mathsf{gsk}[\mathsf{item}][\mathsf{uid}_{\mathsf{item}}], \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M})$: If $\mathsf{info}_{t_{\mathsf{current}}}$ does not contain a witness $w_{i,\mathsf{item}}$ with the first entry being $\mathsf{uid}_{\mathsf{item}} \in \{0,1\}^{\ell}$, return $\perp$. Otherwise, the user downloads $\mathbf{u}_{t_{\mathsf{current}},\mathsf{item}}$ and his witness $w_{i,\mathsf{item}}$ from $\mathsf{info}_{t_{\mathsf{current}}}$. Then, it computes $(c_1, c_2) \leftarrow \mathsf{Enc}_{\mathsf{Regev}}(\mathsf{tpk}, \mathsf{uid}_{\mathsf{item}})$ and the tag $\tau \leftarrow \mathsf{TAG}_{\mathsf{LIT}}(\mathsf{item}, \mathbf{x})$, where recall $\mathsf{usk} = \mathbf{x}$. Finally, it generates a NIZKAoK $\Pi_{\mathsf{Sign}} = (\{\mathsf{CMT}_i\}_{i=1}^{\kappa}, \mathsf{CH}, \{\mathsf{RSP}\}_{i=1}^{\kappa})$ for the relation R_{Sign} , where

$$\mathsf{CH} = \mathcal{H}_{\mathsf{Sign}}\big(\mathsf{M}, \{\mathsf{CMT}_i\}_{i=1}^{\kappa}, \mathbf{A}, \mathbf{u}, \mathcal{H}_{\mathsf{Tag}}(\mathsf{item}), \tau, c_1, c_2, \mathbf{B}, \mathbf{P}_1, \mathbf{P}_2\big) \in \{1,2,3\}^{\kappa},$$

and outputs the signature $\Sigma = (\Pi_{\mathsf{Sign}}, \tau, c_1, c_2)$.

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M}, \Sigma)$: It verifies if Π_{Sign} is a valid proof. If so it outputs 1 and otherwise it outputs 0.

$\mathsf{Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{reg}, \mathsf{item}, \mathsf{M}, \Sigma)$: It first runs $\mathsf{uid}_{\mathsf{item}} \leftarrow \mathsf{Dec}_{\mathsf{Regev}}((\mathbf{S}_1, \mathbf{E}_1), (c_1, c_2))$. Then, it generates a NIZAoK proof Π_{Trace} for the relation R_{Trace}.

$\mathsf{Judge}(\mathsf{gpk}, \mathsf{uid}_{\mathsf{item}}, \Pi_{\mathsf{Trace}}, \mathsf{info}_{t_{\mathsf{current}}}, \mathsf{item}, \mathsf{M}, \Sigma)$: It verifies if Π_{Trace} is a valid proof. If so it outputs 1 and otherwise it outputs 0.

$\mathsf{Link}(\mathsf{gpk}, \mathsf{item}, (\mathsf{M}_0, \Sigma_0), (\mathsf{M}_1, \Sigma_1))$: It parses Σ_0 and Σ_1 and outputs $b \leftarrow \mathsf{Link}_{\mathsf{LIT}}(\tau_0, \tau_1)$, where $b = 1$ when it is linkable and 0 otherwise.

4.2 Security Analysis

We show that our reputation system is secure. Each of the following theorems correspond to the security definitions provided in Sect. 3.2, except for the correctness which can be easily checked to hold. Here, we only provide the high-level overview of some of the proofs that we believe to be of interest, and defer the formal proofs to the full version of the paper. The parameters that appear in the theorems are as provided in the above construction.

Theorem 3 (Anonymity). *Our reputation system is anonymous, assuming the hardness of the decision* $\mathsf{LWE}_{n,q,\chi}$ *problem.*

Proof Overview. We proceed in a sequence of hybrid experiments to show that $|\mathsf{Exp}^{\mathsf{Anon\text{-}0}}_{\mathsf{rep\text{-}sys},\mathcal{A}}(n) - \mathsf{Exp}^{\mathsf{Anon\text{-}1}}_{\mathsf{rep\text{-}sys},\mathcal{A}}(n)| \leq \mathsf{negl}$ for any PPT algorithm. The high level strategy is similar to the anonymity proof for the dynamic group signature scheme provided in [LNWX17], Lemma 2. Namely, for the challenge signature, we swap the user identifier $\mathsf{uid}_{\mathsf{item}}$ embedded in the ciphertexts (c_1, c_2) and the user's secret key usk embedded in the tag τ. The main difference between the proof of [LNWX17] is that for our reputation system we have to swap the tag in the challenge signature. For this, we use the tag indistinguishability property of the underlying tag scheme LWE-LIT presented in Theorem 1. This modification in the experiments are provided in Exp_5 of our proof.

Theorem 4 (Non-Frameability). *Our Reputation System is non-frameable, assuming the hardness of the* $\mathsf{SIS}_{n,m,q,1}$ *problem of the search* $\mathsf{faeLWE}_{m,n,q,\chi}$ *(or equivalently the search* $\mathsf{LWE}_{m-n,q,\chi}$*) problem.*

Proof Overview. For an adversary to win the experiment, he must output a tuple $(\mathsf{uid}^*_{\mathsf{item}^*}, \Pi^*_{\mathsf{Trace}}, \mathsf{info}_{t^*}, \mathsf{item}^*, \mathsf{M}^*, \Sigma^*)$ such that (informally): (i) the pair (M^*, Σ^*) links to some other message-signature pair (M, Σ) corresponding to item^* of an honest non-corrupt user or (ii) the proof Π^*_{Trace} traces the signature Σ^* back to some honest non-corrupt user. Since the latter case (ii) essentially captures the non-frameability of fully dynamic group signatures, the proof follows similarly to [LNWX17], Lemma 3. However, for case (i), we must use a new argument, since this is a security notion unique to reputation systems. In particular, we aim to embed a search LWE problem into the tag of the message-signature pair (M, Σ) of an honest non-corrupt user (where the simulator does not know the secret key usk) for which the adversary outputs a linking signature forgery (M^*, Σ^*). Due to the special nature of our LWE tag scheme, we can prove that if the signatures link, then the two secret keys $\mathsf{usk}, \mathsf{usk}^*$ embedded in the tags must be the same. Therefore, by extracting usk^* from the adversary's forgery, we can solve the search LWE problem. However, the problem with this approach is that since the simulator does not know usk, he will not be able to provide the adversary with this particular user's public key upk, which is defined as $\mathbf{A} \cdot \mathsf{usk} \mod q$. Our final idea to overcome this difficulty is by relying on the so called *first-are-error-less* LWE problem [BLP+13, ALS16], which is proven to be as difficult as the standard LWE problem. Namely, the simulator will be provided with $\mathbf{A} \cdot \mathsf{usk}$ as the error-less LWE samples and uses the remaining non-noise-less LWE samples to simulate the tags.

Theorem 5 (Public Linkability). *Our reputation system is unconditionally public-linkable.*

Proof Overview. We show that no such (possibly inefficient) adversary exists by assuming the linkability property of our underlying tag scheme LWE-LIT presented in Theorem 2, which holds unconditionally. Our strategy is to prove by contradiction. Assuming that an adversary winning the public-linkability experiment exists, we obtain two signatures Σ_0, Σ_1 on item such that the two tags τ_0, τ_1 associated with the signatures does not link, but the two tags embed the

same user secret key usk (which informally follows from the $\Pi_{\mathsf{Trace},b}$ provided by the adversary). Then, by extracting the usk from the signatures produced by the adversary, we can use $(\tau_0, \tau_1, I = \mathsf{item}, \mathsf{sk} = \mathsf{usk})$ to win the linkability experiment of the tag scheme. Thus a contradiction.

The following two theorems follow quite naturally from the proofs of the dynamic group signatures schemes of [LNWX17]. At a high level, this is because the following security notions captures threats that should hold regardless of the presence of tags.

Theorem 6 (Traceability). *Our reputation system is traceable assuming the hardness of the* $\mathsf{SIS}_{n,m,q,1}$ *problem.*

Theorem 7 (Tracing Soundness). *Our reputation system is unconditionally tracing sound.*

Acknowledgement. We would like to thank the anonymous reviewers of FC 2018 for their valuable comments. The first author was funded by a research grant from the UK Government. The second author was partially supported by JST CREST Grant Number JPMJCR1302 and JSPS KAKENHI Grant Number 17J05603.

References

[ACBM08] Androulaki, E., Choi, S.G., Bellovin, S.M., Malkin, T.: Reputation systems for anonymous networks. In: Borisov, N., Goldberg, I. (eds.) PETS 2008. LNCS, vol. 5134, pp. 202–218. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70630-4_13

[ACPS09] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_35

[ALS16] Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_12

[AT99] Ateniese, G., Tsudik, G.: Some open issues and new directions in group signatures. In: Franklin, M. (ed.) FC 1999. LNCS, vol. 1648, pp. 196–211. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48390-X_15

[BBS04] Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_3

[BCC+16] Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 117–136. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_7

[BJK15] Blömer, J., Juhnke, J., Kolb, C.: Anonymous and publicly linkable reputation systems. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 478–488. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_29

[BLP+13] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: STOC, pp. 575–584 (2013)

[BMW03] Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_38

[BS04] Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 168–177 (2004)

[BSS10] Bethencourt, J., Shi, E., Song, D.: Signatures of reputation. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 400–407. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_35

[BSZ05] Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: the case of dynamic groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30574-3_11

[BW06] Boyen, X., Waters, B.: Compact group signatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 427–444. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_26

[Cam97] Camenisch, J.: Efficient and generalized group signatures. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 465–479. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_32

[CG04] Camenisch, J., Groth, J.: Group signatures: better efficiency and new theoretical aspects. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 120–133. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30598-9_9

[CSK13] Clauß, S., Schiffner, S., Kerschbaum, F.: k-anonymous reputation. In: ACM SIGSAC, pp. 359–368 (2013)

[CVH91] Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_22

[Del00] Dellarocas, C.: Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In: ACM Conference on Electronic Commerce, pp. 150–157 (2000)

[DMS03] Dingledine, R., Mathewson, N., Syverson, P.: Reputation in p2p anonymity systems. In: Workshop on Economics of Peer-to-Peer Systems (2003)

[EE17] El Bansarkhani, R., El Kaafarani, A.: Direct anonymous attestation from lattices. IACR Cryptology ePrint Archive (2017)

[GK11] Goodrich, M.T., Kerschbaum, F.: Privacy-enhanced reputation-feedback methods to reduce feedback extortion in online auctions. In: ACM Conference on Data and Application Security and Privacy, pp. 273–282 (2011)

[GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206. ACM (2008)

[JI02] Josang, A., Ismail, R.: The beta reputation system. In: Proceedings of the 15th Bled Electronic Commerce Conference, pp. 2502–2511 (2002)

[Ker09] Kerschbaum, F.: A verifiable, centralized, coercion-free reputation system. In: ACM Workshop on Privacy in the Electronic Society, pp. 61–70 (2009)

[KSGM03] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th International Conference on World Wide Web, pp. 640–651 (2003)

[KTX07] Kawachi, A., Tanaka, K., Xagawa, K.: Multi-bit cryptosystems based on lattice problems. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 315–329. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_21

[KTX08] Kawachi, A., Tanaka, K., Xagawa, K.: Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 372–389. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_23

[LNWX17] Ling, S., Nguyen, K., Wang, H., Xu, Y.: Lattice-based group signatures: achieving full dynamicity with ease. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 2017. LNCS, vol. 10355, pp. 293–312. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_15

[MK14] Michalas, A., Komninos, N.: The lord of the sense: a privacy preserving reputation system for participatory sensing applications. In: IEEE SCC, pp. 1–6 (2014)

[MKKSM13] Molavi Kakhki, A., Kliman-Silver, C., Mislove, A.: Iolaus: securing online content rating systems. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 919–930 (2013)

[MP13] Micciancio, D., Peikert, C.: Hardness of SIS and LWE with small parameters. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 21–39. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_2

[NY90] Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: STOC, pp. 427–437 (1990)

[Pei09] Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: STOC, pp. 333–342 (2009)

[PVW08] Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31

[Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC, pp. 84–93 (2005)

[RKZF00] Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Commun. ACM **12**, 45–48 (2000)

[RZ02] Resnick, P., Zeckhauser, R.: Trust among strangers in internet transactions: empirical analysis of ebay's reputation system. In: The Economics of the Internet and E-commerce, pp. 127–157 (2002)

[SKCD16] Soska, K., Kwon, A., Christin, N., Devadas, S.: Beaver: a decentralized anonymous marketplace with secure reputation. Cryptology ePrint Archive, Report 2016/464 (2016)

[Ste06] Steinbrecher, S.: Design options for privacy-respecting reputation systems within centralised internet communities. In: Fischer-Hübner, S., Rannenberg, K., Yngström, L., Lindskog, S. (eds.) SEC 2006. IIFIP, vol. 201, pp. 123–134. Springer, Boston, MA (2006). https://doi.org/10.1007/0-387-33406-8_11

[VMG+12] Viswanath, B., Mondal, M., Gummadi, K.P., Mislove, A., Post, A.: Canal: scaling social network-based Sybil tolerance schemes. In: Proceedings of the 7th ACM European Conference on Computer Systems, pp. 309–322 (2012)

[YSK+09] Yu, H., Shi, C., Kaminsky, M., Gibbons, P.B., Xiao, F.: Dsybil: optimal sybil-resistance for recommendation systems. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 283–298 (2009)

[ZWC+16] Zhai, E., Wolinsky, D.I., Chen, R., Syta, E., Teng, C., Ford, B.: Anonrep: towards tracking-resistant anonymous reputation. In: NSDI, pp. 583–596 (2016)

Blockchain Measurements

Estimating Profitability of Alternative Cryptocurrencies (Short Paper)

Danny Yuxing Huang[1(✉)], Kirill Levchenko[2], and Alex C. Snoeren[2]

[1] Princeton University, Princeton, USA
dhuang@cs.ucsd.edu
[2] University of California, San Diego, San Diego, USA

Abstract. Digital currencies have flourished in recent years, buoyed by the tremendous success of Bitcoin. These blockchain-based currencies, called *altcoins*, are associated with a few thousand to millions of dollars of market capitalization. Altcoins have attracted enthusiasts who enter the market by mining or buying them, but the risks and rewards could potentially be significant, especially when the market is volatile. In this work, we estimate the potential profitability of mining and speculating 18 altcoins using real-world blockchain and trade data. Using opportunity cost as a metric, we estimate the mining cost for an altcoin with respect to a more popular but stable coin. For every dollar invested in mining or buying a coin, we compute the potential returns under various conditions, such as time of market entry and hold positions. While some coins offer the potential for spectacular returns, many follow a simple bubble-and-crash scenario, which highlights the extreme risks—and potential gains—in altcoin markets.

1 Introduction

In its nine years of existence, Bitcoin [1] (BTC) has been tremendously popular, reaching billions of dollars of market capitalization at the time of this writing. Its success has inspired the creation of many new digital currencies that borrow Bitcoin's key design principles—a blockchain-based public ledger and a means of acquiring a stake in the currency computationally—and, in many cases, Bitcoin's source code. Today, there are over 1,400 such currencies, collectively called *altcoins*. Unlike Bitcoin, which can be used as a medium of exchange, the vast majority of altcoins appear to serve largely as speculative investment vehicles. The market capitalization and trade volume for a given altcoin can range from thousands to millions of dollars. Whether one believes in a coin's merits or not, altcoins offer ample opportunities for speculation for altcoin investors, especially given the volatile prices. As such, the potential risks and rewards can be significant.

One way to analyze the altcoin market is to look at the profit-driven investors. Among these investors are *miners*, a logical role who expends energy in finding hash collisions to computationally produce the digital assets in a process known

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 409–419, 2018.
https://doi.org/10.1007/978-3-662-58387-6_22

as *mining*; and also *speculators*, another logical role who takes advantage of the price volatility of altcoins and profits from speculation.[1] An investor has the flexibility to choose to become a miner and/or a speculator at any point in time. To enter the altcoin market, an investor can mine an altcoin or buy it from the market. If she mines some units of an altcoin,[2] she can further hold onto the coin units and sell them when the price rises, thereby speculating in addition to mining.

The fluidity of these roles makes it difficult for us to retroactively analyze profitability. While existing techniques can potentially help us identify the transfer of altcoins to and from exchanges [6], blockchains frequently do not record when and how many altcoins are internally traded at exchanges; thus, it is difficult to track the profitability of individual investors. Instead, we develop a set of techniques to analyze the potential profitability of particular altcoin investment stategies, from the collective perspectives of miners and speculators. Specifically, our analysis asks these questions: (1) For every $1 of investment, what is the potential profitability of mining versus speculating an altcoin? (2) How does the potential profitability of mining/speculating vary across multiple coins?

In this work, we use historical data to retroactively examine the costs and potential revenue of altcoin mining and speculation. We use *opportunity cost* to compare the relative cost of mining across different altcoins as viewed by a profit-driven miner. Furthermore, we artificially construct simple investment strategies that miners and speculators could have followed—for example, holding units of an altcoin for a fixed period of time before liquidating, without considering the market. We retroactively simulate these strategies to estimate the potentially profitability of miners and speculators. Using this entirely descriptive rather than prescriptive method, we make the following observations based on the altcoins we study. Miners who mine an altcoin immediately after it is listed on exchanges tend to enjoy higher potential returns than miners who mine on subsequent days. In contrast, speculators who buy an altcoin shortly after it is listed on exchanges are likely to generate lower returns than speculators who buy at a later point, and they also generate lower returns than miners in the same period. A more detailed description of our study, including additional findings, can be found in the technical report [7].

2 Methodology

To estimate the potential profitability of mining and speculating across different altcoins, we use historical blockchain and trade data for 18 altcoins, along with

[1] We are aware that there are many ways to profit from altcoins, including gaming the mining protocol [2,3] or trading altcoins as if they were penny stocks [4,5]. It is beyond the scope of this paper to discuss all these ways. Furthermore, there may be other participants in the altcoin ecosystem that are not necessarily profit-driven; again, these participants are beyond the our scope.

[2] Throughout this paper, we use "altcoin" or "coin" interchangeably to refer to the cryptocurrency. In contrast, we use "units" to refer to individual units of reward as a result of mining an altcoin.

Bitcoin (BTC) and Litecoin (LTC). This data contains detailed block metadata, such as the reward and difficulty of each block, along with daily aggregated trade statistics, such as the mean price and trade volume every day. We show these 18 altcoins in the first column of Table 1. A full description of our data collection and clean-up process is available in our technical report [7]. In this section, we describe how we use this data to estimate the cost of mining and profitability.

Estimating the Cost of Mining. In order for profit-driven miners to decide which coin to mine through proof-of-work, they first need to estimate the cost of mining each coin. However, the precise value is difficult to calculate, as the capital investment and energy costs differ across individual miners. As an alternative to direct costs, we instead consider opportunity cost. Economists consider the opportunity cost of an activity to be the revenue lost by engaging that activity *rather than its best alternative*. We apply the same idea to altcoin mining. In particular, for two coins based on the same hash function, the real costs of mining one versus the other depends entirely on their respective difficulty, because the underlying unit of work, computing a hash, is the same. For example, XJO is a SHA-256-based altcoin. The opportunity cost of mining XJO is the revenue a miner can expect from mining another SHA-256 currency instead of XJO over the same time period. To be a meaningful concept for the miner, this alternative revenue should be something a miner can reasonably expect to receive *a priori*. In other words, to say that a miner chose to mine A units of XJO rather than receive D US Dollars for mining another currency, the miner must be certain that he could get D US Dollars by choosing the alternative *before* choosing one or the other. In our comparisons, we use the least volatile alternative currencies with the highest trade volumes. For SHA-256-based coins, this is Bitcoin; for Scrypt-based coins, this is Litecoin. We call the currency whose opportunity cost we are computing (e.g. XJO) the *target currency*, and Bitcoin or Litecoin the *base currency*.

Formally, we define the opportunity cost of mining a unit of a currency on a given day as follows. First, we determine the expected number of hashes, H, necessary to mine a unit of the target currency based on the difficulty of mining the currency that day. Next, we determine the expected number of units of Bitcoin (for SHA-256-based altcoins) or Litecoin (for Scrypt-based altcoins) that could be mined on that day with H hashes. Finally, we convert this expected number of bitcoins or litecoins to US Dollars at that day's exchange rate. Thus, the opportunity cost of mining a unit of currency X is $\text{OppCost}_X = D_X \cdot R_B / D_B$, where D_X is the expected number of hashes required to mine a unit of X based on the day's difficulty, D_B is the expected number of hashes required to mine a unit of the base currency (Bitcoin or Litecoin) based on the day's difficulty, and R_B is the exchange rate of the base currency, in US Dollars per unit of the currency, on that day. We compute D_X and D_B based on the blockchain data of the target and base currencies, while we obtain R_B from our trade data.

Some altcoins start as simple SHA-256/Scrypt proof-of-work cryptocurrencies, but later change the hash functions or types of proof (such as proof-of-stake). In such circumstances, we consider the history of the currency up to the

day of change, so that we can use BTC or LTC as the base currency for calculating opportunity costs of mining from the start of the blockchain to the day of the change. For example, DOGE blocks are based on Scrypt in the first 276 days of the blockchain; afterwards, DOGE allows auxiliary proof-of-work mining (i.e., "merged mining"). As a result, we only consider the first 276 days of the 919-day blockchain for DOGE.

Validation. Focusing on the relevant analysis periods, we compute the opportunity cost, c, of mining a unit of a given coin on a day. From the trade data that we have collected, we can compare it to the market price, p, of the unit on the same day and compute the Pearson correlation coefficient for the daily c and p values. Across the 18 altcoins we study, 12 altcoins are associated with a correlated coefficient > 0.80. (As a comparison, the Dow Jones Industrial Average and the S&P 500 Index between May 2012 and 2017 are correlated with a coefficient of 0.99.) The high correlation between c and p suggests not only is opportunity cost is an effective estimate of the actual mining cost, but that the markets are reasonably efficient. One possible explanation for market efficiency is that as more hype is created around a coin, the market price increases, which in turn attracts more miners. This increases the difficulty of mining and also the opportunity cost, so the opportunity cost goes up along with the price. Conversely, a coin that has attracted a significant amount of hashing capacity has a high difficulty and thus opportunity cost. Thus, miners expect to sell the mined coin units at a higher price. For a detailed analysis of the opportunity cost of mining each altcoin, refer to our technical report [7].

Estimating Profitability. Computing the profit obtained by individual actors is difficult, especially when we only have a daily aggregates of the trade data. Instead, we focus on the potential profitability of a dollar invested, either through mining (i.e., \$1 of opportunity cost) or speculating (e.g., literally spending \$1 to buy a given altcoin).

We estimate the profitability of miners through simulation. Using historical blockchain and trade data, we retroactively simulate the investment of \$1 worth of opportunity cost in mining across different altcoins, start dates, and durations—conditions that we have artificially constructed in the hope of covering a diverse range of investment strategies. For speculators, we use a similar simulation, varying the time when an investor enters the market by buying \$1 worth of an altcoin's units, as well as the holding position of the investor. In this way, we can compute the profitability of mining/speculating depending on the participant's strategy. An altcoin market typically has a trade volume much higher than \$1, such that \$1 of mining or speculating is unlikely to change the price significantly. This profitability analysis, while retrospective, assesses the relative risks and rewards for each coin, across multiple coins.

Table 1. Mining continuously for 7 and 30 days. All units are in percentages unless otherwise stated.

Coin	(a) 7 Days of mining					(b) 30 Days of mining				
	1st day r	E(r)	$\sigma(r)$	P	$T_{r\geq 0}$ (Days)	1st day r	E(r)	$\sigma(r)$	P	$T_{r\geq 0}$ (Days)
ARG	4.41	2.61	7.22	76.28	8	−0.51	0.71	1.50	80.71	0
AUR	10.17	0.63	2.25	61.35	19	1.37	0.13	0.35	62.31	18
BTA	6.67	2.01	5.18	69.33	14	0.99	0.45	1.07	68.97	13
CURE	6.23	−6.68	5.92	14.54	18	0.72	−1.54	1.01	8.79	8
DGC	3.24	1.16	2.79	61.66	111	0.77	0.27	0.57	64.89	207
DOGE	70.63	4.22	10.51	100.00	N/A	8.88	0.76	1.31	100.00	N/A
DOT	10.48	18.29	12.12	99.21	14	2.94	4.92	1.15	100.00	N/A
EFL	16.48	2.00	2.07	89.29	30	1.61	0.47	0.34	94.89	18
HAM	13.49	−2.19	6.43	18.82	46	3.10	−0.51	1.30	14.89	65
PPC	0.09	−1.13	1.37	16.50	1	0.02	−0.26	0.17	3.40	4
RPC	10.52	0.74	3.08	59.54	6	0.82	0.17	0.46	59.85	32
SWING	2.74	−2.46	3.77	19.09	3	−0.04	−0.53	0.55	21.83	0
TROLL	−0.01	4.37	1.31	98.08	0	0.87	0.99	0.06	100.00	N/A
UNO	8.44	−5.43	5.52	20.99	79	1.26	−1.34	1.23	15.98	72
VCN	56.98	−2.22	11.77	27.18	34	7.40	−0.75	1.87	30.23	25
VIA	−1.68	2.76	2.09	95.07	0	0.71	0.67	0.33	100.00	N/A
WBB	6.21	6.58	4.45	97.87	83	0.75	1.54	0.78	100.00	N/A
XJO	3.03	−5.51	4.00	4.48	15	0.18	−1.28	0.75	0.83	4

3 Estimating Profitability

In this section, we compute the profitability of miners and speculators for every dollar invested, either by expending $1 worth of opportunity cost in mining or buying $1 worth of an altcoin.

Mining. We start by considering the profitability of miners. Using the unit opportunity cost that we computed in the previous section, we construct a simulator in which a miner continuously mines over a duration of d days, starting on Day i. Every day, he invests $1 worth of opportunity cost in mining. At the end of each day, he sells all the coin units mined on that day. At the end of the d days, his total revenue would be v dollars. We vary i between 1 to $N - d$, where N is the length of the trading period; for instance, $i = 1$ means that our simulated miner starts mining on the day when an altcoin is first listed on an exchange. We also vary d for $d = 1, 7, 30$ days. For all these i and d combinations, we compute the daily average returns, r, by solving for r in this equation: $d(1 + r)^d = v$.

Table 1(a) shows the result of our first simulation, in which a miner continuously mines for $d = 7$ days. The "1st Day r" column shows the value of r if the miner starts mining a coin on Day 1 and continues until Day 7, selling

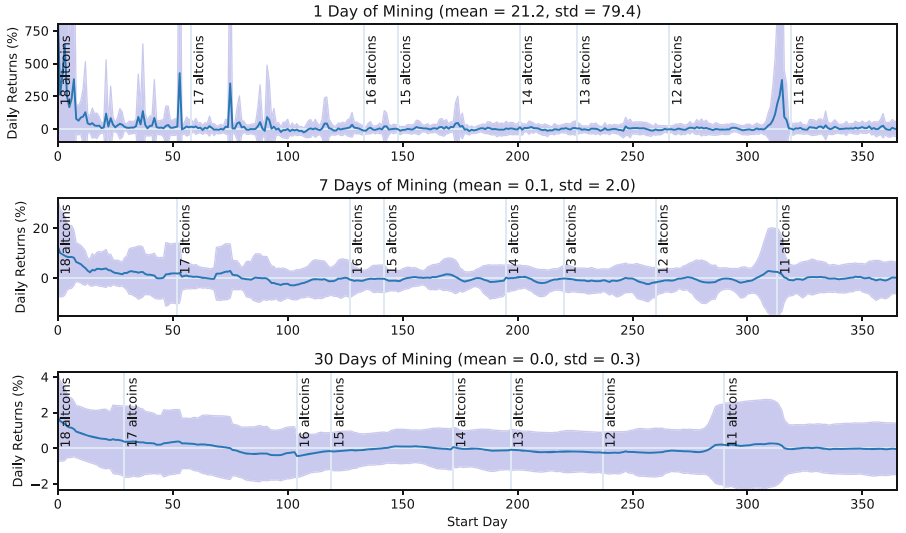


Fig. 1. Potential profitability of mining a random altcoin.

any mined units on the same day. In contrast, the "$E(r)$" column computes the expected/mean r if the miner starts mining on a random day. The standard deviation is shown in the "$\sigma(r)$" column. The larger the standard deviation, the higher the risk of investment if a miner starts on a random day. For instance, mining DOT on a random day results in an expected daily return of $18.29 \pm 12.12\%$, which potentially presents a higher reward and risk than mining AUR, with an expected daily return of $0.63 \pm 2.25\%$. We stress that the r value is computed based on a simulated investment of \$1 worth of opportunity cost. In total, DOT is associated with \$917 of trade volume and \$3,198 of total opportunity cost. Even though its r value is high, the amount of actual profit extracted is likely to be limited. In contrast, a PPC miner can generate an expected return of $-1.13 \pm 1.37\%$. Given that PPC is associated with more than \$8 million worth of trade volume and \$8 million total opportunity cost, an actual miner has the potential to suffer significant losses.

Another way to measure risk is to compute the probability, P, of achieving a positive r if a miner starts mining an altcoin for 7 days on a random start day. As shown in the "P" column, miners of altcoins like DOGE and WBB will earn positive returns on (almost, in the case of WBB) any random start day. XJO miners, by contrast, will earn positive returns on a random start day with a probability of only 4.48%.

We observe that of the 18 altcoins in the table, the 1st Day r values are higher than the corresponding $E(r)$ values in 14 altcoins, which suggests that miners of these 14 coins obtain higher returns if they start mining as soon as an altcoin is listed on an exchange, relative to the average case (i.e. $E(r)$). One possible reason is that when an altcoin is first listed, the amount of mining capacity it

Table 2. A speculator that holds for 7 and 30 days. All units are in percentages unless otherwise specified.

Coin	(a) 7 Days of speculating					(b) 30 Days of speculating				
	1st day r	E(r)	$\sigma(r)$	P	$T_{r<0}$ (Days)	1st day r	E(r)	$\sigma(r)$	P	$T_{r<0}$ (Days)
ARG	10.72	−0.32	5.56	39.02	0	−0.48	−0.41	2.90	31.48	26
AUR	6.69	−0.22	5.87	41.38	0	3.39	−0.52	2.46	40.22	0
BTA	−26.24	0.35	6.87	47.31	11	−11.71	0.56	3.06	54.35	31
BTC	−1.97	0.44	2.51	55.17	6	−0.72	0.42	1.43	57.59	18
CURE	−3.74	−0.45	3.76	41.90	22	−6.30	−0.40	1.77	40.19	47
DGC	−5.33	0.00	4.44	41.26	10	−1.90	−0.06	2.21	37.04	10
DOGE	86.27	−0.06	4.53	39.06	0	5.24	−0.06	1.48	37.84	0
DOT	−17.47	−0.37	20.13	41.09	2	−18.67	−1.68	7.79	45.28	23
EFL	17.07	0.02	4.60	47.95	0	0.21	−0.18	2.06	47.83	0
HAM	−19.35	0.46	7.15	51.05	8	−1.12	0.35	2.35	64.24	1
LTC	−4.12	0.08	3.54	45.45	6	−0.34	0.05	1.63	39.55	6
PPC	−1.66	0.14	3.08	44.25	6	−0.16	0.11	1.60	46.99	3
RPC	−1.99	−1.13	4.89	39.08	1	−1.46	−1.25	2.24	28.50	88
SWING	1.61	−0.52	5.20	42.28	0	−1.04	−0.66	2.05	48.59	51
TROLL	−1.93	−0.69	4.38	51.92	1	−2.25	−0.16	1.22	51.72	13
UNO	3.34	−0.09	3.74	47.45	0	−2.08	−0.10	1.39	44.65	20
VCN	−43.26	0.80	7.99	58.97	5	−13.06	1.18	2.44	73.26	6
VIA	2.72	−0.21	3.53	44.74	0	1.49	−0.39	1.43	33.19	0
WBB	−20.82	0.17	5.77	41.97	22	−7.41	0.25	2.41	53.05	17
XJO	7.36	−0.82	3.37	35.68	0	−2.37	−0.86	1.58	29.90	25

has attracted is still on the rise, as there is friction for miners to reconfigure their equipment to mine a new-to-market altcoin; thus the opportunity cost of mining tends to be low in the beginning. Furthermore, the market price is often high when an altcoin is listed—a period typically associated with hype. The gap between high price and low opportunity cost creates a potential for miners to profit during this period.

In fact, miners can potentially profit during the first few consecutive days after an altcoin is listed on exchanges. Column "$T_{r\geq 0}$" shows the number of consecutive days since Day 1, such that a miner who starts mining on one of these days and continues for 7 days will not encounter a negative r. For example, an ARG miner who starts mining on any day of the first 8 days will receive $r \geq 0$; if she starts mining on Day 9, she would receive $r < 0$ for the first time (although she could still obtain positive returns subsequently). For TROLL and VIA, the 1st Day r is already negative, so $T_{r\geq 0} = 0$. For DOGE, all r values are positive regardless of the start day; thus $T_{r\geq 0} = \text{N/A}$.

Finally, we repeat the simulation above, changing $d = 30$ days. We show the results in Table 1(b). Again, for 14 out of the 18 altcoins, mining on Day 1

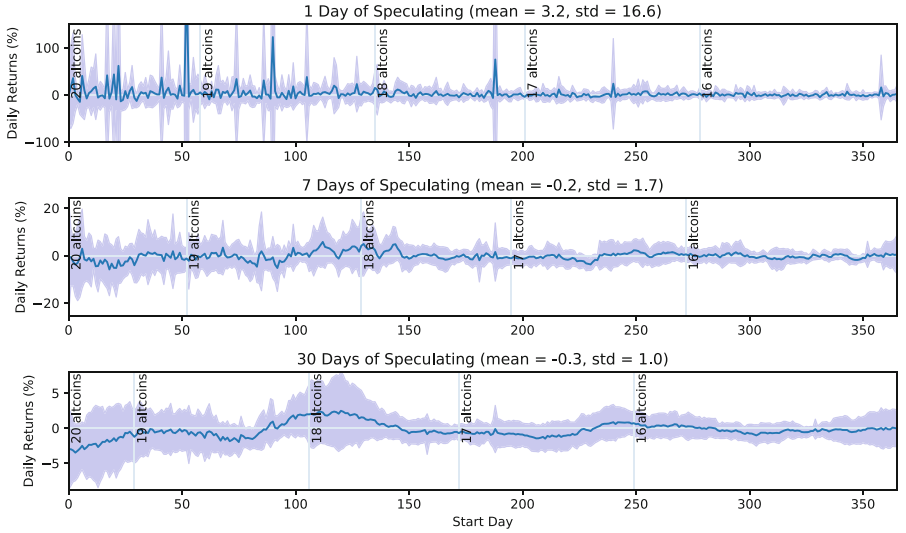


Fig. 2. Potential profitability of speculating a random coin.

will yield a higher r than the expected case. The expected returns are lower for $d = 30$ than $d = 7$ in 11 of the altcoins, while all the $\sigma(r)$ values are smaller.

So far, we have examined the daily average returns for individual altcoins. This approach assumes that a miner already knows which altcoin to mine. Our next simulation departs from such an assumption, and it instead looks at a case where a miner randomly picks one of the 18 altcoins in Table 1 and start mining on Day i. Again, $i = 1$ means the miner starts on the same day when a coin is listed on an exchange. The miner will stick to mining this altcoin for d days, devoting $1 of opportunity cost of mining every day, and selling all mined units at the end of each day. Since the miner picks a coin at random for each i, our goal is to compute the distribution of daily returns for these 18 coins for given i and d values.

First, suppose we set $d = 1$ day. The top chart in Fig. 1 shows the result of our new simulation. The x-axis shows the start day of mining, i (relative to the first day of listing at an exchange for each coin). The y-axis shows the distribution of daily average returns, r. The solid line represents the expected (i.e. mean) r for picking a random coin and starting to mine on Day i for $d = 1$ day. The band above and below the solid line indicates the standard deviation of r. For i between 0 and 57, a miner can randomly mine one of the 18 altcoins on Day i. At its peak, the mean r is $885.1 \pm 2,781.0\%$ on Day $i = 0$. The expected r value decreases over time. Between $i = 58$ and $i = 132$, the miner can only pick one of 17 altcoins, as we do not have the trading data for one of the altcoins beyond 57 days. In general, the mean of the expected r values between $i = 0$ and $i = 365$ is $21.2 \pm 79.4\%$ (i.e. the expected returns for a miner who picks a random coin on a random start day).

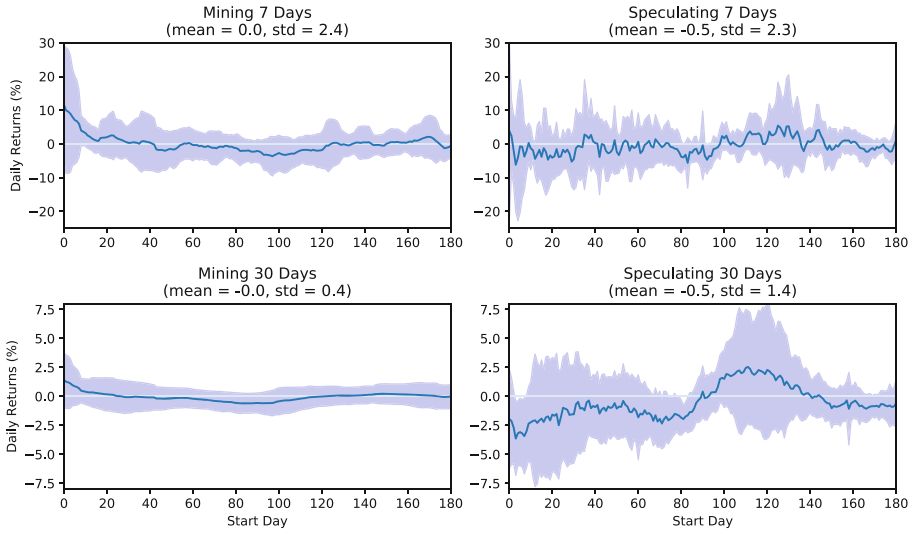


Fig. 3. Comparing the potential profitability of mining and speculating for the same set of 14 altcoins.

The middle and bottom charts in Fig. 1 show the results for $d = 7$ and 30 days. As d increases, the expected r and its standard deviation both decrease. Effectively, with lower expected returns, the risks are potentially lower, too. Furthermore, as d increases, a persistent pattern is that the returns tend to be higher when i is low, regardless of the d values. This implies that a miner who picks a random altcoin and mines it shortly after the altcoin is listed is likely to receive higher returns than later.

Speculating. In contrast to miners who acquire altcoins through mining, speculators acquire altcoins by buying from exchanges. Also, while miners mine and sell on the same day, speculators buy coins on one day and sell them later. We design similar simulations to measure the potential profitability for speculators. Specifically, we require that a speculator enter the market on a random day i, buy \$1 worth of coins, hold them the next $d - 1$ days, sell all the coins at the end of the d-day period for a total of v dollars. Again, we compute the average daily return by solving for r in this equation: $1 \cdot (1 + r)^d = v$.

Table 2 shows the results of our simulation. Similar to Tables 1 and 2 shows the returns on the 1st Day, as well as the expected r values for $d = 7$ and 30 days. However, the trend is the opposite. Across the 18 altcoins, plus LTC and BTC, a speculator who enters the market on Day 1 receives *lower* returns than the average case for 12 of the coins for $d = 7$; for 30 days, we observe the same trend across 16 of the coins. In contrast to the $T_{r \geq 0}$ metric in Table 1, we compute $T_{r<0}$ here in Table 2, which counts the number of consecutive days since Day 1, such that if a speculator enters the market on one of these days for a given d value, she will receive $r < 0$. For instance, for $d = 7$ days, if a CURE speculator

enters the market on any day between 1 and 22, she will receive negative returns; on Day 23, she will receive positive returns for the first time.

In addition to analyzing the returns for individual altcoins, we examine the case where a speculator picks an altcoin at random. Figure 2 shows the result. Similar to Fig. 1 and 2 shows that as d increases from 1 Day, 7 Days, to 30 Days, the mean of the expected r values decreases, and so does the standard deviation; in other words, as the holding time increases, the potential returns and risks decrease. Contrary to Fig. 1 and 2 shows that a speculator who picks a random altcoin and enters the market soon after the altcoin is listed on exchanges is likely to receive *lower* returns than if she enters the market later.

To compare the returns between mining and speculating, we identify 14 out of the 18 coins, such that all of them can be involved in mining and speculation for $d = 7, 30$ days and $i = 0, \ldots, 180$ days. Again, we assume that an investor randomly picks one of the 14 coins on Day i, enters the market either by mining or buying, and exits d days later. We show the results in Fig. 3. For both $d = 7$ and 30 days, if an investor who picks a random altcoin decides to enter the market early, her expected returns will be higher if she becomes a miner. For $d = 30$, if an investor decides to become a speculator, she can potentially receive higher returns in the best case (e.g. more than 6% around $i = 120$) than the best-case returns in mining (i.e. less than 5% at $i = 0$). However, the risk of speculation is also higher, as indicated by the larger standard deviation for the expected r value.

For more details on our findings, discussions of the results, and a description of the related work, refer to our technical report [7].

4 Conclusion

In this work, we compare the profitability of mining versus speculation for 18 altcoins. By comparing against BTC and LTC, we use opportunity cost to estimate the miners' effort in the 18 coins, and we design simulations to estimate the daily returns per $1 of investment, either through mining or speculating, under various conditions. These simulations show that a miner who starts mining shortly after an altcoin is listed can potentially earn higher returns than the average case, whereas a speculator who enters the market shortly after an altcoin is listed on exchanges might potentially earn lower returns. We also show that returns from mining a random altcoin tend to be lower with smaller standard deviations—less risky—than from speculation.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
2. Kiayias, A., Koutsoupias, E., Kyropoulou, M., Tselekounis, Y.: Blockchain mining games. In: Proceedings of the 2016 ACM Conference on Economics and Computation, pp. 365–382. ACM (2016)

3. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
4. Bouraoui, T.: Stock spams: an empirical study on penny stock market. Int. Rev. Bus. Res. Pap. **5**, 292–305 (2009)
5. Fraedrich, J.P.: Signs and signals of unethical behavior. In: Business Forum, vol. 17, p. 13. California State University, Los Angeles, School of Business and Economics (1992)
6. Meiklejohn, S., et al.: A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the ACM Internet Measurement Conference (2013)
7. Huang, D.Y., Levchenko, K., Snoeren, A.C.: Measuring profitability of alternative cryptocurrencies. Technical report CS2017-1019, University of California, San Diego (2017). https://www.sysnet.ucsd.edu/~dhuang/altcoin-report/

A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin

Roman Matzutt[1(✉)], Jens Hiller[1], Martin Henze[1], Jan Henrik Ziegeldorf[1], Dirk Müllmann[2], Oliver Hohlfeld[1], and Klaus Wehrle[1]

[1] Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany
{matzutt,hiller,henze,ziegeldorf,hohlfeld,wehrle}@comsys.rwth-aachen.de
[2] Data Protection Research Institute, Goethe University Frankfurt, Frankfurt, Germany
muellmann@jur.uni-frankfurt.de

Abstract. Blockchains primarily enable credible accounting of digital events, e.g., money transfers in cryptocurrencies. However, beyond this original purpose, blockchains also irrevocably record *arbitrary* data, ranging from short messages to pictures. This does not come without risk for users as each participant has to locally replicate the complete blockchain, particularly including potentially harmful content. We provide the first systematic analysis of the benefits and threats of arbitrary blockchain content. Our analysis shows that certain content, e.g., illegal pornography, can render the mere possession of a blockchain illegal. Based on these insights, we conduct a thorough quantitative and qualitative analysis of unintended content on Bitcoin's blockchain. Although most data originates from benign extensions to Bitcoin's protocol, our analysis reveals more than 1600 files on the blockchain, over 99% of which are texts or images. Among these files there is clearly objectionable content such as links to child pornography, which is distributed to all Bitcoin participants. With our analysis, we thus highlight the importance for future blockchain designs to address the possibility of unintended data insertion and protect blockchain users accordingly.

1 Introduction

Bitcoin [45] was the first completely distributed digital currency and remains the most popular and widely accepted of its kind with a market price of ∼4750USD per bitcoin as of August 31st, 2017 [14]. The enabler and key innovation of Bitcoin is the blockchain, a public append-only and tamper-proof log of all transactions ever issued. These properties establish trust in an otherwise trustless, completely distributed environment, enabling a wide range of new applications, up to distributed general-purpose data management systems [69] and purely digital data-sharing markets [41]. In this work, we focus on arbitrary, i.e., non-financial,

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 420–438, 2018.
https://doi.org/10.1007/978-3-662-58387-6_23

data on Bitcoin's famous blockchain, which primarily stores financial transactions. This non-financial data fuels, e.g., digital notary services [50], secure releases of cryptographic commitments [16], or non-equivocation schemes [62].

However, since all Bitcoin participants maintain a *complete local copy* of the blockchain (e.g., to ensure correctness of blockchain updates and to bootstrap new users), these desired and vital features put all users at risk when *objectionable content* is irrevocably stored on the blockchain. This risk potential is exemplified by the (mis-)use of Bitcoin's blockchain as an anonymous and irrevocable content store [35,40,56]. In this paper, we systematically analyze non-financial content on Bitcoin's blockchain. While most of this content is harmless, there is also content that is considered objectionable in many jurisdictions, e.g., the depiction of a nude, young woman or hundreds of links to child pornography. As a result, it could become (or already is today) illegal to possess a copy of the blockchain, which is required to participate in Bitcoin. Hence, illegal content can jeopardize the currently popular multi-billion dollar blockchain systems.

These observations raise the question whether or not such initially unintended content is ultimately beneficial or destructive for blockchain-based systems. To address this question, we provide the first comprehensive and *systematic* study of non-financial content on Bitcoin's blockchain. We first *survey* and explain methods to store non-financial content on Bitcoin's blockchain and discuss potential benefits as well as threats, most notably w.r.t. content deemed illegal in different jurisdictions. Subsequently and in contrast to related work [12,40,56], we *quantify* and discuss unintended blockchain content w.r.t. the wide range of insertion methods. We believe that objectionable blockchain content is a pressing issue despite potential benefits and hope to stimulate research to mitigate the resulting risks for novel as well as existing systems such as Bitcoin.

This paper is organized as follows. We survey methods to insert arbitrary data into Bitcoin's blockchain in Sect. 2 and discuss their benefits and risks in Sect. 3. In Sect. 4, we systematically analyze non-financial content on Bitcoin's blockchain and assess resulting consequences. We discuss related work in Sect. 5 and conclude this paper in Sect. 6.

2 Data Insertion Methods for Bitcoin

Beyond intended recording of financial transactions, Bitcoin's blockchain also allows for injection of *non-financial* data, either short messages via special transaction types or even complete files by encoding arbitrary data as standard transactions. We give a brief introduction to Bitcoin transactions and then survey methods available to store arbitrary content on the blockchain via transactions.

Bitcoin *transactions* transfer funds between a payer (sender) and a payee (receiver), who are identified by public-private key pairs. Payers announce their transactions to the *Bitcoin network*, whose *miners* then publish these transactions in new *blocks* using their computational power in exchange for a *fee*. These fees vary, with an average fee of 215 sat (satoshi, $1\,\text{sat} = 10^{-8}$ bitcoin) per byte during August 2017 [4]. Each transaction consists of several *input scripts*, which

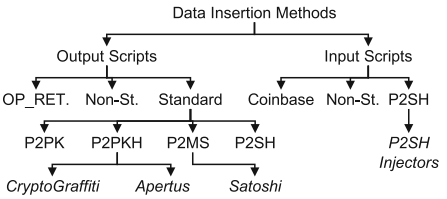


Fig. 1. Bitcoin data insertion methods (italics show content insertion services)

Table 1. Payload, costs, and efficiency of low-level data insertion methods

Method	Payload	Costs/B	Eff.
OP_RET.	80B	3.18–173.55ct	Poor
Coinbase	96B	—	Poor
Non-St. Out.	99 044B	1.03–198.05ct	Poor
Non-St. In.	99 044B	1.03–198.05ct	Med.
P2PK	85 345B	1.24–207.79ct	High
P2PKH	58 720B	1.87–197.58ct	High
P2MS	92 625B	1.11–234.33ct	High
P2SH Out.	62 400B	1.77–195.54ct	High
P2SH In.	99 018B	1.03–225.61ct	High

unlock funds of previous transactions, and of several *output scripts*, which specify receivers of funds. To unlock funds, input scripts contain a signature for the previous transaction generated by the owner of the funds. To prevent malicious scripts from causing excessive transaction verification overheads, Bitcoin uses transaction script *templates* and expects peers to discard non-compliant scripts.

Figure 1 shows the insertion methods for non-financial data we identified for Bitcoin. We distinguish *low-level data insertion methods* inserting small data chunks and *content insertion services*, which systematically utilize these low-level methods to insert arbitrary content. From now on, we refer to non-financial blockchain data as *content* if it has a self-contained structure, e.g., a file or readable text, or as *data* otherwise, e.g., fragments inserted via a low-level method.

2.1 Low-Level Data Insertion Methods

We first survey the efficiency of the low-level data insertion methods w.r.t. to *insertable payload* and *costs* per transaction (Table 1). To this end, we first explain our comparison methodology, before we detail (i) intended data insertion methods (OP_RETURN and coinbase), (ii) utilization of non-standard transactions, and (iii) manipulation of standard transactions to insert arbitrary data.

Comparison Methodology. We measure the *payload per transaction (PpT)*, i.e., the number of non-financial bytes that can be added to a single standard-sized transaction ($\leq$ 100 000B). *Costs* are given as the minimum and maximum *costs per byte (CpB)* for the longest data chunk a transaction can hold, and for inserting 1B. Costs are inflicted by paying transaction fees and possibly *burning* currency (at least 546 sat per output script), i.e., making it unspendable. For our cost analysis we assume Bitcoin's market price of 4748.25USD as of August 31st, 2017 [14] and the average fees of 215 sat per byte as of August 2017 [4]. Note that high variation of market price and fees results in frequent changes of presented absolute costs per byte. Finally, we rate the overall *efficiency* of an approach w.r.t. insertion of arbitrary-length content. Intuitively, a method is efficient if it allows for easy insertion of large payloads at low costs.

OP_RETURN. This special transaction template allows attaching one small data chunk to a transaction and thus provides a *controlled channel* to annotate transactions without negative side effects. E.g., in typical implementations peers increase performance by caching spendable transaction outputs and OP_RETURN outputs can safely be excluded from this cache. However, data chunk sizes are limited to 80B per transaction.

Coinbase. In Bitcoin, each block contains exactly one coinbase transaction, which introduces new currency to incentivize miners to dedicate their computational power to maintain the blockchain. The input script of coinbase transactions is up to 100B long and consists of a variable-length field encoding the new block's position on the blockchain [9]. Stating a larger size than the overall script length allows placing arbitrary data in the resulting gap. This method is cumbersome as only active miners can insert only small data chunks.

Non-standard Transactions. Transactions can deviate from the approved transaction templates [48] via their output scripts as well as input scripts. In theory, such transactions can carry arbitrarily encoded data chunks. Transactions using non-standard *output* scripts can carry up to 96.72 KiB at comparably low costs. They are cumbersome as miners ignore them with high probability. Yet, non-standard output scripts occasionally enter the blockchain if miners insufficiently check them (cf. Sect. 4.2). Contrarily, non-standard *input* scripts are only required to match their respective output script. Hence, input scripts can be altered to carry arbitrary data if their semantics are not changed, e.g., by using dead conditional branches. This makes non-standard input scripts slightly better suited for large-scale content insertion than non-standard output scripts.

Standard Financial Transactions. Even *standard financial transactions* can be (mis-)used to insert data using mutable values of output scripts. There are four approved templates for standard financial transactions: Pay to public key (P2PK) and pay to public-key hash (P2PKH) transactions send currency to a dedicated receiver, identified by an address derived from her private key, which is required to spend any funds received [48]. Similarly, multi-signature (P2MS) transactions require m out of n private keys to authorize payments. Pay to script hash (P2SH) transactions refer to a *script* instead of keys to enable complex spending conditions [48], e.g., to replace P2MS [10]. The respective public keys (P2PK, P2MS) and hash values (P2PKH, P2SH) can be replaced with arbitrary data as Bitcoin peers cannot verify their correctness before they are referenced by a subsequent input script. While this method can store large amounts of content, it also incurs significant costs: In addition to transaction fees, the user must burn bitcoins as she replaces valid receiver identifiers with arbitrary data (i.e., invalid receiver identities), making the output unspendable. Using multiple outputs enables PpTs ranging from 57.34 KiB (P2PKH) to 96.70 KiB (P2SH inputs) at CpBs from 1.03ct to 1.87ct. As they behave similarly w.r.t. data insertion, we collectively refer to all standard financial transactions as P2X in the following. P2SH scripts also allow for efficient data insertion into input scripts as P2SH input scripts are published with their redeem script. Due to miners'

verification of P2SH transactions, transactions are not discarded if the redeem script is not template-compliant (but the overall P2SH transaction is).

We now survey different services that systematically leverage the discussed data insertion methods to add larger amounts of content to the blockchain.

2.2 Content Insertion Services

Content insertion services rely on the low-level data insertion methods to add content, i.e., files such as documents or images, to the blockchain. We identify four conceptually different content insertion services and present their protocols.

CryptoGraffiti. This web-based service [30] reads and writes messages and files from and to Bitcoin's blockchain. It adds content via multiple P2PKH output scripts within a single transaction, storing up to 60 KiB of content. To retrieve previously added content, CryptoGraffiti scans for transactions that either consist of at least 90% printable characters or contain an image file.

Satoshi Uploader. The Satoshi Uploader [56] inserts content using a single transaction with multiple P2X outputs. The inserted data is stored together with a length field and a CRC32 checksum to ease decoding of the content.

P2SH Injectors. Several services [35] insert content via slightly varying P2SH input scripts. They store chunks of a file in P2SH input scripts. To ensure file integrity, the P2SH redeem scripts contain and verify hash values of each chunk.

Apertus. This service [29] allows *fragmenting* content over multiple transactions using an arbitrary number of P2PKH output scripts. Subsequently, these fragments are referenced in an *archive* stored on the blockchain, which is used to retrieve and reassemble the fragments. The chosen encoding optionally allows augmenting content with a comment, file name, or digital signature.

To conclude, Bitcoin offers various options to insert arbitrary, non-financial data. These options range from small-scale data insertion methods exclusive to active miners to services that allow any user to store files of arbitrary length. This wide spectrum of options for data insertion raises the question which benefits and risks arise from storing content on Bitcoin's blockchain.

3 Benefits and Risks of Arbitrary Blockchain Content

In Bitcoin, there are several methods to insert arbitrary, non-financial data into its blockchain in both intended and unintended ways. In this section, we discuss the potential benefits of storing arbitrary data on Bitcoin's blockchain as well as the risks of (mis-)using these channels for content insertion.

3.1 Benefits of Arbitrary Blockchain Content

Besides the manipulation of standard financial transactions, Bitcoin offers coinbase and OP_RETURN transactions as explicit channels to irrevocably insert

small chunks of non-financial data into its blockchain (cf. Sect. 2). As we discuss in the following, each insertion method has distinguishing benefits:

OP_RETURN. Augmenting transactions with short pieces of arbitrary data is beneficial for a wide area of applications [12,40,62]. Different services use OP_RETURN to link non-financial assets, e.g., vouchers, to Bitcoin's blockchain [12,40], to attest the existence of digital documents at a certain point of time as a digital notary service [12,50,58], to realize distributed digital rights management [12,70], or to create non-equivocation logs [8,62].

Coinbase. Coinbase transactions differ from OP_RETURN as only miners, who dedicate significant computational resources to maintain the blockchain, can use them to add extra chunks of data to their newly mined blocks. Beyond advertisements or short text messages [40], coinbase transactions can aid the mining process. Adding random bytes to the coinbase transactions allows miners to increase entropy when repeatedly testing random nonces to solve the proof-of-work puzzle [48]. Furthermore, adding identifiable *voting flags* to transactions enables miners to vote on proposed features, e.g., the adoption of P2SH [10].

Large-Scale Data Insertion. Storing large amounts of data on the blockchain creates a long-term non-manipulable file storage. This enables, e.g., the archiving of historical data or censorship-resistant publication, which helps protecting whistleblowers or critical journalists [66]. However, their content is replicated to all users, who do not have a choice but to persistently store it.

Hence, non-financial data on the blockchain enables new applications that leverage Bitcoin's security guarantees. In the following, we discuss threats of forcing oblivious, honest users to download copies of all blockchain content.

3.2 Risks of Arbitrary Blockchain Content

Despite potential benefits of data on the blockchain, insertion of objectionable content can put all participants of the Bitcoin network at risk [11,40,43], as such unwanted content is unchangeable and locally replicated just like benign data by each peer of the Bitcoin network. To fortify this threat, we first develop an extensive catalog of content that poses high risks if possessed by individuals and then argue that objectionable blockchain content is able to harm oblivious, honest users. We identify five categories of objectionable content:

Copyright Violations. With the advent of file-sharing networks, pirated data has become a huge challenge for copyright holders. To tackle this problem, copyright holders predominantly target users that actively distribute pirated data. E.g., German law firms sue users who distribute copyright-protected content via file-sharing networks for fines on behalf of the copyright holders [28]. In recent years, prosecutors also convicted downloaders of pirated data. For instance, France temporarily suspended users' Internet access and subsequently switched to issuing high fines [36]. As users distribute their copy of the blockchain to new peers, copyright-protected material on the blockchain can thus provoke legal disputes about copyright infringement.

Malware. Another threat is malware [20,42], which could potentially be spread via blockchains [31]. Malware has serious consequences as it can destroy sensitive documents, make devices inoperable, or cause financial loss [34]. Furthermore, blockchain malware can irritate users as it causes antivirus software to deny access to important blockchain files. E.g., Microsoft's antivirus software detected a non-functional virus signature from 1987 on the blockchain, which had to be fixed manually [68].

Privacy Violations. By disclosing sensitive personal data, individuals can harm their own privacy and that of others. This threat peaks when individuals deliberately violate the privacy of others, e.g., by blackmailing victims under the threat of disclosing sensitive data about them on the blockchain. Real-world manifestations of these threats are well-known, e.g., non-consensually releasing private nude photos or videos [54] or fully disclosing an individual's identity to the public with malicious intents [21]. Jurisdictions such as the whole European Union begin to actively prosecute the unauthorized disclosure *and forwarding* of private information in social networks to counter this novel threat [5].

Politically Sensitive Content. Governments have concerns regarding the leakage of classified information such as state secrets or information that otherwise harms national security, e.g., propaganda. Although whistleblowers unveil problems such as corruption, they force all blockchain users to keep a copy of leaked material. Depending on the jurisdiction, the intentional disclosure or the mere possession of such content may be illegal. While, e.g., the US government usually tends to prosecute *intentional* theft or disclosure of state secrets [63], in China the mere possession of state secrets can result in longtime prison sentences [49]. Furthermore, China's definition of state secrets is vague [49] and covers, e.g., "activities for safeguarding state security" [60]. Such vague allegations w.r.t. state secrets have been applied to critical news in the past [18,24].

Illegal and Condemned Content. Some categories of content are virtually universally condemned and prosecuted. Most notably, possession of child pornography is illegal at least in the 112 countries [64] that ratified an optional protocol to the Convention on the Rights of the Child [65]. Religious content such as certain symbols, prayers, or sacred texts can be objectionable in extremely religious countries that forbid other religions and under oppressive regimes that forbid religion in general. As an example, possession of items associated with an opposed religion, e.g., bibles in radical Islamist countries, or blasphemy have proven risky and were sometimes even punished by death [13,38].

In conclusion, a wide range of objectionable content can ultimately harm users if possessed by them. In contrast to systems such as social media platforms, file-sharing networks, or online storage systems, such content can be stored on blockchains anonymously and irrevocably. Since all blockchain data is downloaded and persistently stored by users, they are liable for any objectionable content added to the blockchain by others. Consequently, it would be illegal to participate in a blockchain system as soon as illegal content is introduced to it.

While this risk has previously been acknowledged [43], definitive answers require court rulings yet to come. However, considering legal texts we anticipate a high potential for illegal blockchain content to jeopardize blockchain-based systems such as Bitcoin in the future. Our belief stems from the fact that, w.r.t. child pornography as an extreme case of illegal content, legal texts from countries such as the USA [47], England [3], and Ireland [32] deem all data illegal that can be converted into a visual representation of illegal content. As we stated in Sect. 2, it is easily possible to locate and reassemble such content on the blockchain. Hence, even though convertibility usually covers creating a visual representation by, e.g., decoding an image file, we expect that the term can be interpreted to include blockchain data in the future. For instance, this is already covered implicitly by German law, as a person is culpable for possession of illegal content if she *knowingly possesses* an *accessible document* holding said content [2]. It is critical here that German law perceives the hard disk holding the blockchain as a document [1] and that users can easily reassemble any illegal content on the blockchain. Furthermore, users can be assumed to *knowingly* maintain control over such illegal content w.r.t. German law if sufficient media coverage causes the content's existence to become public knowledge among Bitcoin users [61], as has been attempted by Interpol [31]. We thus believe that legislators will deal with non-financial blockchain content and that this has the potential to jeopardize systems such as Bitcoin if they hold illegal content.

4 Blockchain Content Landscape

To understand the landscape of non-financial blockchain data and assess its potentials and risks, we thoroughly analyze Bitcoin's blockchain as it is the most widely used blockchain today. Especially, we are interested in (i) the *degree of utilization* of data and content insertion methods, (ii) the *temporal evolution* of data insertion, and (iii) the *types* of content on Bitcoin's blockchain, especially w.r.t. *objectionable* content. In the following, we first outline our measurement methodology before we present an overview and the evolution of non-financial data on Bitcoin's blockchain. Finally, we analyze files stored on the blockchain to determine if any objectionable content is already present on the blockchain.

4.1 Methodology

We detect data-holding transactions recorded on Bitcoin's blockchain based on our study of data insertion methods and content insertion services (cf. Sect. 2). We distinguish detectors for data insertion methods and detectors for content insertion services. To reduce false positives, e.g., due to public-key hash values that resemble text, we exclude all standard transaction outputs that include already-spent funds from analysis. This is sensible as data-holding transactions replace public keys or hashes such that spending requires computing corresponding private keys or pre-images, which is assumed to be infeasible. Contrarily, even though we thoroughly analyzed possible insertion methods, there is still a chance

that we do not exhaustively detect all non-financial data. Nevertheless, our content type analysis establishes a solid lower bound as we only consider readable files retrieved from Bitcoin's blockchain. In the following, we explain the key characteristics of the two classes of our blockchain content detectors.

Low-Level Insertion Method Detectors. The first class of detectors is tailored to match individual transactions that are likely to contain non-financial data (cf. Sect. 2.1). These detectors detect manipulated financial transactions as well as OP_RETURN, non-standard, and coinbase transactions.

Our text detector scans P2X output scripts for mutable values containing $\geq$ 90% printable ASCII characters (to avoid false positives). The detector returns the concatenation of all output scripts of the same transaction that contain text.

Finally, we consider all coinbase and OP_RETURN transactions as well as non-standard output scripts. We detect coinbase transactions based on the length field mismatch described in Sect. 2.1. OP_RETURN scripts are detectable as they always begin with an `OP_RETURN` operation. Non-standard output scripts comprise all output scripts which are not template-conform.

Service Detectors. We implemented detectors specific to the content insertion services we identified in Sect. 2.2. These service-specific detectors enable us to *detect and extract* files based on the services' protocols. These detectors also track the data insertion method used in service-created transactions.

The CryptoGraffiti detector matches transactions with an output that sends a tip to a public-key hash controlled by its provider. For such a transaction, we concatenate all mutable values of output scripts that spend fewer than 10 000 sat and store them in a file. This threshold is used to ignore non-manipulated output scripts, e.g., the service provider spending their earnings.

To detect a Satoshi Uploader transaction, we concatenate all of its mutable values that spend the same small amount of bitcoins. If we find the first eight bytes to contain a valid combination of length and CRC32 checksum for the transaction's payload, we store the payload as an individual file.

We detect P2SH Injector content based on redeem scripts containing more than one hash operation (standard transactions use at most one). We then extract the concatenation of the second inputs of all redeem scripts (the first one contains a signature) of a transaction as one file.

Finally, the Apertus detector recursively scans the blockchain for Apertus archives, i.e., Apertus-encoded lists of previous transaction identifiers. Once a referenced Apertus payload does not constitute another archive, we retrieve its payload file and optional comment by parsing the Apertus protocol.

Suspicious Transaction Detector. To account for less wide-spread insertion services, we finally analyze standard transactions that likely carry non-financial data but are not detected otherwise. We only consider transactions with at least 50 *suspicious outputs*, i.e., roughly 1 KiB of content. We consider a set of outputs suspicious if all outputs (i) spend the same small amount ($<$ 10 000 sat) and (ii) are unspent. This detector trades off detection rate against false-positive rate.

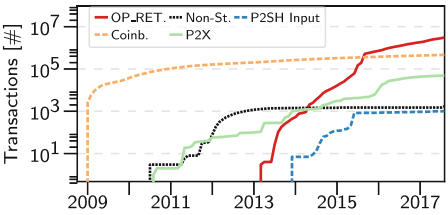


Fig. 2. Cumulative numbers of detected transactions per data insertion method

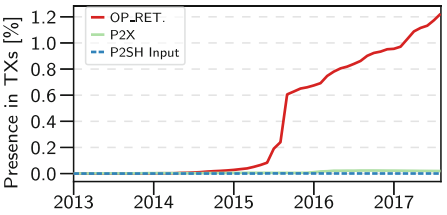


Fig. 3. Ratio of transactions that utilize data insertion methods

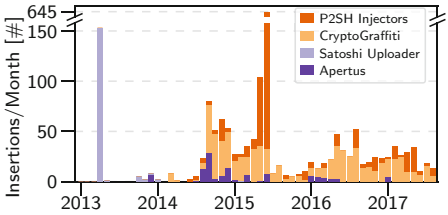


Fig. 4. Number of files inserted via content insertion services per month

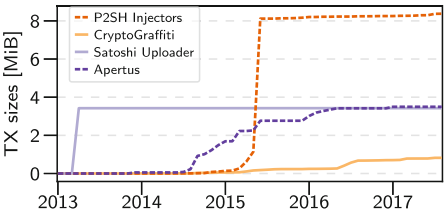


Fig. 5. Cumulative sizes of transactions from content insertion services

Due to overlaps with service detectors, we exclude matches of this detector from our quantitative analysis, but discuss individual findings in Sect. 4.3.

4.2 Utilization of Data Insertion Methods

Data and content insertion in Bitcoin has evolved over time, transitioning from single miners exploiting coinbase transactions to sophisticated services that enable the insertion of whole files into the blockchain. We study this evolution in terms of used data insertion methods as well as content insertion services and quantify the amount of blockchain data using our developed detectors. Our key insights are that OP_RETURN constitutes a widely accepted success story while content insertion services are currently only infrequently utilized. However, the introduction of OP_RETURN did not shut down other insertion methods, e.g., P2X manipulation, which enable single users to insert objectionable content.

Our measurements are based on Bitcoin's complete blockchain as of August 31st, 2017, containing 482 870 blocks and 250 845 217 transactions with a total disk size of 122.64 GiB. We first analyze the popularity of different data insertion methods and subsequently turn towards the utilization of content insertion services to assess how non-financial data enters the blockchain.

Data Insertion Methods. As described in Sect. 2.1, OP_RETURN and coinbase transactions constitute *intended* data insertion methods, whereas P2X and

non-standard P2SH inputs manipulate legitimate transaction templates to contain arbitrary data. Figure 2 shows the cumulative number of transactions containing non-financial data on a logarithmic scale. In total, our detectors found 3 535 855 transactions carrying a total payload of 118.53 MiB, i.e., 1.4% of Bitcoin transactions contain non-financial data. We strive to further understand the characteristics of this non-financial blockchain content as even a single instance of objectionable content can potentially jeopardize the overall system.

The vast majority of extracted transactions are OP_RETURN (86.8% of all matches) and coinbase (13.13%) transactions. Combined, they constitute 95.90 MiB (80.91% of all extracted data). Out of all blocks, 96.15% have content-holding coinbase transactions. While only 0.26% of these contain $\geq 90\%$ printable text, 33.49% of them contain ≥ 15 consecutive printable ASCII characters (mostly surrounded by data without obvious structure). Of these short messages, 14.39% contain voting flags for new features (cf. Sect. 3.1). Apart from this, miners often advertise themselves or leave short messages, e.g., prayer verses.

OP_RETURN transactions were introduced in 2013 to offer a benign way to augment single transactions with non-financial data. This feature is widely used, as shown by Fig. 3. Among all methods, OP_RETURN is the only one to be present with a rising tendency, with currently 1.2% of all transactions containing OP_RETURN outputs. These transactions predominantly manage off-blockchain assets or originate from notary services [12]. While P2X transactions are continuously being manipulated, they make up only 0.02% of all transactions; P2SH inputs are virtually irrelevant. Hence, short non-financial data chunks are well-accepted, viable extensions to the Bitcoin system (cf. Sect. 3.1).

P2X transactions are asymmetric w.r.t. the number and sizes of data-carrying transactions. Although constituting only 1.6% of all detector hits, they make up 9.08% of non-financial data (10.76 MiB). This again highlights the high content-insertion efficiency of P2X transactions (cf. Sect. 2.1).

Finally, we discuss non-standard transactions and non-standard P2SH input scripts. In total, we found 1 703 transactions containing non-standard outputs. The three first non-standard transactions (July 2010) repeatedly used the `OP_CHECKSIG` operation. We dedicate this to an attempted DoS attack that aimed to cause high verification times. Furthermore, we found 23 P2PKH transactions from October 2011 that contained `OP_0` instead of a hash value. The steady increase of non-standard transactions in 2012 is due to scripts that consist of 32 seemingly random bytes. Contrarily, P2SH input scripts sporadically carry non-standard redeem scripts and are then often used to insert larger data chunks (as they are used by P2SH Injectors). This is due to P2SH scripts not being checked for template conformity. We found 888 such transactions holding 8.37 MiB of data. Although peers should reject such transactions [48], they still often manage to enter the blockchain. Non-standard P2SH scripts even carry a substantial amount of data (7.07% of the total data originate from P2SH Injectors).

Content Insertion Services. We now investigate to which extent content insertion services are used to store content on Bitcoin's blockchain. Figure 4 shows utilization patterns for each service and Fig. 5 shows the cumulative size

of non-financial data inserted via the respective service. Notably, only few users are likely responsible for the majority of service-inserted content.

In total, content insertion services account for 16.12 MiB of non-financial data. More than a half of this content (8.37 MiB) originates from P2SH Injectors. The remainder was mostly inserted using Apertus (21.70% of service-inserted data) and Satoshi Uploader (21.24%). Finally, CryptoGraffiti accounts for 0.82 MiB (5.10%) of content related to content insertion services. In the following, we study how the individual services have been used over time.

Our key observation is that both CryptoGraffiti and P2SH Injectors are infrequently but steadily used; since 2016 we recognize on average 23.65 data items being added per month using these services. Contrarily, Apertus has been used only 26 times since 2016, while the Satoshi Uploader has not been used at all. In fact, the Satoshi Uploader was effectively used only during a brief period: 92.73% of all transactions emerged in April 2013. During this time, the service was used to upload four archives, six backup text files, and a PDF file.

Although Apertus and the Satoshi Uploader have been used only infrequently, together they constitute 64.32% of all P2X data we detected. This stems from the utilization of those services to store files on the blockchain, e.g., archives or documents (Satoshi Uploader), or images (Apertus). Similarly, P2SH Injectors are used to backup conversations regarding the development of the Bitcoin client, especially online chat logs, forum threads, and emails, with a significant peak utilization between May and June 2015 (76.46% of P2SH Injector matches). Especially Apertus is well-suited for this task as files are spread over multiple transactions. Based on the median, the average Apertus file has a size of 17.15 KiB and is spread over 10 transactions, including all overheads. The largest Apertus file is 310.72 KiB large (including overheads), i.e., three times the size of a standard transaction, and is spread over 96 transactions. The most heavily fragmented Apertus file even spans 664 transactions. Contrarily, 95.7% of CryptoGraffiti matches are short text messages with a median length of 80B.

In conclusion, content insertion services are only infrequently used with varying intentions and large portions of content were uploaded in bursts, indicating that only few users are likely responsible for the majority of service-inserted blockchain content. While CryptoGraffiti is mostly used to insert short text messages that also fit into one OP_RETURN transaction, other services are predominantly used to store larger files, e.g., images or documents. As such files can constitute objectionable content, we further investigate them in the following.

4.3 Investigating Blockchain Files

After quantifying basic content insertion in Bitcoin, we now focus on readable files that are extractable from the blockchain. We refer to *files* as findings of our content-insertion-service or suspicious-transaction detectors that are viewable using appropriate standard software. We reassemble fragmented files only if this is unambiguously possible, e.g., via an Apertus archive. Out of the 22.63 MiB of blockchain data not originating from coinbase or OP_RETURN transactions, we can extract and analyze 1 557 files with meaningful content. In addition to

Table 2. Distribution of blockchain file types according to our content-insertion-service and suspicious-transactions detectors.

File type	Via service?		Overall portion	File type	Via service?		Overall portion
	Yes	No			Yes	No	
Text	1 353	54	87.07%	Archive	4	0	0.25%
Images	144	2	9.03%	Audio	2	0	0.12%
HTML	45	0	2.78%	PDF	2	0	0.12%
Source code	7	3	0.62%	**Total**	1 557	59	100.0%

these, we extracted 59 files using our suspicious-transaction detector (92.25% text). Table 2 summarizes the different file types of the analyzed files. The vast majority are text-based files and images (99.34%).

In the following, we discuss our findings with respect to objectionable content. We manually evaluated all readable files with respect to the problematic categories we identified in Sect. 3.2. This analysis reveals that content from all those categories already exists on Bitcoin's blockchain today. For each of these categories, we discuss the most severe examples. To protect the safety and privacy of individuals, we omit personal identifiable information and refrain from providing exact information on the location of critical content on the blockchain.

Copyright Violations. We found seven files that publish (intellectual) property and showcase Bitcoin's potential to aid copyright violations. Stored are the text of a book, a copy of the original Bitcoin paper [45,56], and two short textual white papers. Furthermore, we found two leaked cryptographic keys: one firmware secret key [56] and an RSA private key. Finally, the blockchain contains a so-called illegal prime, encoding software to break the copy protection of DVDs [56].

Malware. We could not find actual malware on Bitcoin's blockchain. However, an individual non-standard transaction contains a non-malicious cross-site scripting detector. A security researcher inserted this small piece of code, which, if interpreted by an online blockchain parser, notifies the author about the vulnerability. Such malicious code could become a threat for users as most websites offering an online blockchain parser also offer online Bitcoin accounts.

Privacy Violations. Users store memorable private moments on the blockchain. We extracted six wedding-related images and one image showing a group of people, labeled with their online pseudonyms. Furthermore, 609 transactions contain online public chat logs, emails, and forum posts discussing Bitcoin, including topics such as money laundering. Storing *private* chat logs on the blockchain can, e.g., leak single user's private information irrevocably. Moreover, third parties can release information without knowledge nor consent of affected users. Most notably, we found at least two instances of *doxing*, i.e., the complete disclosure of another individual's personal information. This data includes phone numbers, addresses, bank accounts, passwords, and multiple online identities.

Recently, jurisdictions such as the European Union began to punish such serious privacy violations, including the distribution of doxing data [5]. Again, the blockchain's immutability aggravates the impact of such assaults.

Politically Sensitive Content. The blockchain has been used by whistleblowers as a censorship-resistant permanent storage for leaked information. We found backups of the WikiLeaks Cablegate data [37] as well as an online news article concerning pro-democracy demonstrations in Hong Kong in 2014 [25]. As stated in Sect. 3.2, restrictive governments are known to prosecute the possession of such content. For example, state-critical media coverage has already put individuals in China [18] or Turkey [24] at the risk of prosecution.

Illegal and Condemned Content. Bitcoin's blockchain contains at least eight files with sexual content. While five files only show, describe, or link to mildly pornographic content, we consider the remaining three instances objectionable in almost all jurisdictions: Two of them are backups of link lists to child pornography, containing 274 links to websites, 142 of which refer to Tor hidden services. The remaining instance is an image depicting mild nudity of a young woman. In an online forum this image is claimed to show child pornography, albeit this claim cannot be verified (due to ethical concerns we refrain from providing a citation). Notably, two of the explicit images were only detected by our suspicious-transaction detector, i.e., they were not inserted via known services.

While largely harmless, potentially objectionable blockchain content is infrequently being inserted, e.g., links to alleged child pornography or privacy violations. We thus believe that future blockchain designs must proactively cope with objectionable content. Peers can, e.g., filter incoming transactions or revert content-holding transactions [11,51], but this must be scalable and transparent.

5 Related Work

Previous work related to ours comprises i) mitigating the distribution of objectionable content in file-sharing peer-to-peer networks, ii) studies on Bitcoin's blockchain, iii) reports on Bitcoin's susceptibility for content insertion, and iv) approaches to retrospectively remove blockchain content.

The trade-off between enabling open systems for data distribution and risking that unwanted or even illegal content is being shared is already known from peer-to-peer networks. Peer-to-peer-based file-sharing protocols typically limit the spreading of objectionable *public* content by tracking the reputation of users offering files [6,26,55,73] or assigning a reputation to files themselves [19,67]. This way, users can reject objectionable content or content from untrustworthy sources. Contrarily, distributed content stores usually resort to encrypt *private* files before outsourcing them to other peers [7,17]. By storing only encrypted files, users can plausibly deny possessing any content of others and can thus obliviously store it on their hard disk. Unfortunately, these protection mechanisms are not applicable to blockchains, as content cannot be deleted once it has been added to the blockchain and the utilization of encryption cannot be enforced reliably.

Bitcoin's blockchain was analyzed w.r.t. different aspects by numerous studies. In a first step, multiple research groups [33,39,53,71,72] studied the currency flows in Bitcoin, e.g., to perform wealth analyses. From a different line of research, several approaches focused on user privacy and investigated the identities used in Bitcoin [23,44,46,52,59]. These works analyzed to which extent users can be de-anonymized by clustering identities [23,44,46,52,59] and augmenting these clusters with side-channel information [23,44,52,59]. Finally, the blockchain was analyzed w.r.t. the use cases of OP_RETURN transactions [12]. While this work is very close to ours, we provide a first comprehensive study of the complete landscape of non-financial data on Bitcoin's blockchain.

The seriousness of objectionable content stored on public blockchains has been motivated by multiple works [11,40,43,51,56,57]. These works, however, focus on reporting individual incidents or consist of preliminary analyses of the distribution and general utilization of content insertion. To the best of our knowledge, this paper gives the first comprehensive analysis of this problem space, including a categorization of objectionable content and a survey of potential risks for users if such content enters the blockchain. In contrast to previously considered attacks on Bitcoin's ecosystem [22,27], illegal content can be inserted instantly at comparably low costs and can put all participants at risk.

The utilization of chameleon hash functions [15] to chain blocks recently opened up a potential approach to mitigate unwanted or illegal blockchain content [11]. Here, a single blockchain maintainer or a small group of maintainers can retrospectively revert single transactions, e.g., due to illegal content. To overcome arising trust issues, μchain [51] leverages the consensus approach of traditional blockchains to vote on alterations of the blockchain history. These approaches tackle unwanted content for newly designed blockchains, and we seek to motivate a discussion on countermeasures also for *existing* systems, e.g., Bitcoin.

6 Conclusion

The possibility to store non-financial data on cryptocurrency blockchains is both beneficial and a threat for its users. Although controlled channels to insert non-financial data at small rates open up a field of new applications such as digital notary services, rights management, or non-equivocation systems, objectionable or even illegal content has the potential to jeopardize a whole cryptocurrency. Although court rulings do not yet exist, legislative texts from countries such as Germany, the UK, or the USA suggest that illegal content such as child pornography can make the blockchain illegal to possess for all users.

As we have shown in this paper, a plethora of fundamentally different methods to store non-financial and potentially objectionable content on the blockchain exists in Bitcoin. As of now, this can affect at least the 112 countries in which possessing content such as child pornography is illegal. This especially endangers the multi-billion dollar markets powering cryptocurrencies such as Bitcoin.

To assess this problem's severity, we comprehensively analyzed the *quantity* and *quality* of non-financial blockchain data in Bitcoin today. Our quantitative

analysis shows that 1.4% of the roughly 251 million transactions on Bitcoin's blockchain carry arbitrary data. We could retrieve over 1 600 files, with new content infrequently being added. Despite a majority of arguably harmless content, we also identify different categories of objectionable content. The harmful potential of *single instances* of objectionable blockchain content is already showcased by findings such as links to illegal pornography or serious privacy violations.

Acknowledgements. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference number 16KIS0443. The responsibility for the content of this publication lies with the authors.

References

1. German Criminal Code, Section 11 (2013)
2. German Criminal Code, Sections 184b and 184c (2013)
3. Protection of Children Act, Chapter 37, Section 7 (2015)
4. Bitcoin transaction fees (2016). https://bitcoinfees.info. Accessed 23 Sept 2017
5. General Data Protection Regulation, Section 24 (2016)
6. Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: ACM CIKM, pp. 310–317 (2001)
7. Adya, A., et al.: FARSITE: federated, available, and reliable storage for an incompletely trusted environment. SIGOPS Oper. Syst. Rev. **36**(SI), 1–14 (2002)
8. Ali, M., Shea, R., Nelson, J., Freedman, M.J.: Blockstack: a new decentralized internet (2017). Accessed 23 Sept 2017
9. Andresen, G.: Block v2 (Height in Coinbase) (2012). https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki. Accessed 23 Sept 2017
10. Andresen, G.: Pay to script hash (2012). https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki. Accessed 23 Sept 2017
11. Ateniese, G., Magri, B., Venturi, D., Andrade, E.: Redactable blockchain - or - rewriting history in bitcoin and friends. In: IEEE EuroS&P, pp. 111–126 (2017)
12. Bartoletti, M., Pompianu, L.: An analysis of bitcoin OP_RETURN metadata. In: FC Bitcoin Workshop (2017)
13. Bellinger, J., Hussain, M.: Freedom of speech: the great divide and the common ground between the united states and the rest of the world. In: Islamic Law and International Human Rights Law: Searching for Common Ground? pp. 168–180 (2012)
14. Blockchain.info: Bitcoin charts (2011). https://blockchain.info/charts. Accessed 23 Sept 2017
15. Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-hashes with ephemeral trapdoors. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 152–182. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_6
16. Clark, J., Essex, A.: CommitCoin: carbon dating commitments with bitcoin. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 390–398. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_28
17. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_4

18. Committee to Protect Journalists: Chinese journalist accused of illegally acquiring state secrets (2015). https://cpj.org/x/660d. Accessed 23 Sept 2017
19. Damiani, E., di Vimercati, D.C., Paraboschi, S., Samarati, P., Violante, F.: A reputation-based approach for choosing reliable resources in peer-to-peer networks. In: ACM CCS, pp. 207–216 (2002)
20. Dell Security: Annual threat report (2016). Accessed 23 Sept 2017
21. Douglas, D.M.: Doxing: a conceptual analysis. Eth. Inf. Technol. **18**(3), 199–210 (2016)
22. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
23. Fleder, M., Kester, M., Sudeep, P.: Bitcoin transaction graph analysis (2015)
24. Freedom House: Turkey freedom of the press report (2016). https://freedomhouse.org/report/freedom-press/2016/turkey. Accessed 23 Sept 2017
25. Gracie, C.: Hong Kong stages huge National Day democracy protests (2014). http://www.bbc.com/news/world-asia-china-29430229. Accessed 23 Sept 2017
26. Gupta, M., Judge, P., Ammar, M.: A reputation system for peer-to-peer networks. In: ACM NOSSDAV, pp. 144–152 (2003)
27. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security, pp. 129–144 (2015)
28. Herald Union: Copyright infringement by illegal file sharing in Germany (2015). http://www.herald-union.com/copyright-infringement-by-illegal-file-sharing-in-germany. Accessed 23 Sept 2017
29. HugPuddle: Apertus - Archive data on your favorite blockchains (2013). http://apertus.io. Accessed 23 Sept 2017
30. Hyena: Cryptograffiti.info. http://cryptograffiti.info. Accessed 23 Sept 2017
31. Interpol: INTERPOL cyber research identifies malware threat to virtual currencies (2015). https://www.interpol.int/News-and-media/News/2015/N2015-033. Accessed 23 Sept 2017
32. Irish Office of the Attorney General: Child Trafficking and Pornography Act, Section 2. Irish Statue Book, pp. 44–61 (1998)
33. Kondor, D., Pósfai, M., Csabai, I., Vattay, G.: Do the rich get richer? An empirical analysis of the Bitcoin transaction network. PLOS ONE **9**(2), 1–10 (2014)
34. F-Secure Labs: Ransomware: how to predict, prevent, detect & resond. Threat Response (2016). Accessed 23 Sept 2017
35. Le Calvez, A.: Non-standard P2SH scripts (2015). https://medium.com/@alcio/non-standard-p2sh-scripts-508fa6292df5. Accessed 23 Sept 2017
36. Lee, D.: France ends three-strikes internet piracy ban policy (2013). http://www.bbc.com/news/technology-23252515. Accessed 12 Dec 2017
37. Lynch, L.: The Leak heard round the world? Cablegate in the evolving global mediascape. In: Brevini, B., Hintz, A., McCurdy, P. (eds.) Beyond WikiLeaks: Implications for the Future of Communications, Journalism and Society, pp. 56–77. Palgrave Macmillan, London (2013). https://doi.org/10.1057/9781137275745_4
38. Lyons, K., Blight, G.: Where in the world is the worst place to be a Christian? (2015). Accessed 23 Sept 2017
39. Maesa, D.D.F., Marino, A., Ricci, L.: Uncovering the Bitcoin blockchain: an analysis of the full users graph. In: IEEE DSAA, pp. 537–546 (2016)
40. Matzutt, R., Hohlfeld, O., Henze, M., Rawiel, R., Ziegeldorf, J.H., Wehrle, K.: POSTER: I don't want that content! On the risks of exploiting Bitcoin's blockchain as a content store. In: ACM CCS (2016)

41. Matzutt, R., et al.: myneData: towards a trusted and user-controlled ecosystem for sharing personal data. In: Eibl, M., Gaedke, M. (eds.) INFORMATIK, pp. 1073–1084. Gesellschaft für Informatik, Bonn (2017)
42. McAfee Labs: Threats report (December 2016). (2016) Accessed 23 Sept 2017
43. McReynolds, E., Lerner, A., Scott, W., Roesner, F., Kohno, T.: Cryptographic currencies from a tech-policy perspective: policy issues and technical directions. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) FC 2015. LNCS, vol. 8976, pp. 94–111. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_8
44. Meiklejohn, S., et al.: A fistful of Bitcoins: characterizing payments among men with no names. In: IMC, pp. 127–140 (2013)
45. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
46. Ober, M., Katzenbeisser, S., Hamacher, K.: Structure and anonymity of the Bitcoin transaction graph. Future Internet **5**(2), 237–250 (2013)
47. Office of the Law Revision Counsel of the United States House of Representatives: U.S. Code, Title 18, Chapter 110, §2256 (2017)
48. Okupski, K.: Bitcoin developer reference. Technical report (2014)
49. Peerenboom, R.P.: Assessing human rights in China: why the double standard (2005). Accessed 23 Sept 2017
50. PoEx Co., Ltd: Proof of existence (2015). https://proofofexistence.com. Accessed 23 Sept 2017
51. Puddu, I., Dmitrienko, A., Capkun, S.: μchain: how to forget without hard forks. IACR Cryptology ePrint Archive 2017/106 (2017). Accessed 23 Sept 2017
52. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Altshuler, Y., Elovici, Y., Cremers, A., Aharony, N., Pentland, A. (eds.) Security and Privacy in Social Networks, pp. 197–223. Springer, New York (2013)
53. Ron, D., Shamir, A.: Quantitative analysis of the full Bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_2
54. Scheller, S.H.: A picture is worth a thousand words: the legal implications of revenge porn. North Carolina Law Rev. **93**(2), 551–595 (2015)
55. Selcuk, A.A., Uzun, E., Pariente, M.R.: A reputation-based trust management system for P2P networks. In: IEEE CCGrid, pp. 251–258 (2004)
56. Shirriff, K.: Hidden surprises in the Bitcoin blockchain and how they are stored: Nelson Mandela, Wikileaks, photos, and Python software (2014). http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html. Accessed 23 Sept 2017
57. Sleiman, M.D., Lauf, A.P., Yampolskiy, R.: Bitcoin message: data insertion on a proof-of-work cryptocurrency system. In: ACM CW, pp. 332–336 (2015)
58. Snow, P., Deery, B., Lu, J., Johnston, D., Kirby, P.: Factom: business processes secured by immutable audit trails on the blockchain (2014). https://www.factom.com/devs/docs/guide/factom-white-paper-1-0. Accessed 23 Sept 2017
59. Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: extracting intelligence from the Bitcoin network. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 457–468. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_29
60. Standing Committee of the National People's Congress: Law of the People's Republic of China on guarding state secrets (1989). Accessed 23 Sept 2017
61. Taylor, G.: Concepts of intention in german criminal law. Oxf. J. Legal Stud. **24**(1), 99–127 (2004)

62. Tomescu, A., Devadas, S.: Catena: efficient non-equivocation via Bitcoin. In: IEEE S&P, pp. 393–409 (2017)
63. Tucker, E.: A look at federal cases on handling classified information (2016). http://www.military.com/daily-news/2016/01/30/a-look-at-federal-cases-on-handling-classified-information.html. Accessed 23 Sept 2017
64. United Nations: Appendix to the optional protocols to the convention on the rights of the child on the involvement of children in armed conflict and on the sale of children, child prostitution and child pornography (2000)
65. United Nations: Optional protocols to the convention on the Rights of the Child on the involvement of children in armed conflict and on the sale of children, child prostitution and child pornography, vol. 2171, pp. 247–254 (2000)
66. Waldman, M., Rubin, A.D., Cranor, L.: Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In: USENIX Security, pp. 59–72 (2000)
67. Walsh, K., Sirer, E.G.: Experience with an object reputation system for peer-to-peer filesharing. In: NSDI (2006)
68. Wei, W.: Ancient 'STONED' virus signatures found in Bitcoin blockchain (2014). https://thehackernews.com/2014/05/microsoft-security-essential-found.html. Accessed 23 Sept 2017
69. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2016). Accessed 23 Sept 2017
70. Zeilinger, M.: Digital art as 'monetised graphics': enforcing intellectual property on the blockchain. Philosop. Technol. **31**, 15–41 (2016)
71. Ziegeldorf, J.H., Grossmann, F., Henze, M., Inden, N., Wehrle, K.: CoinParty: secure multi-party mixing of Bitcoins. In: ACM CODASPY, pp. 75–86 (2015)
72. Ziegeldorf, J.H., Matzutt, R., Henze, M., Grossmann, F., Wehrle, K.: Secure and anonymous decentralized Bitcoin mixing. FGCS **80**, 448–466 (2018)
73. Zimmermann, T., Rüth, J., Wirtz, H., Wehrle, K.: Maintaining integrity and reputation in content offloading. In: IEEE/IFIP WONS, pp. 1–8 (2016)

Decentralization in Bitcoin and Ethereum Networks

Adem Efe Gencer[1,2], Soumya Basu[1,2(✉)], Ittay Eyal[1,3], Robbert van Renesse[1,2], and Emin Gün Sirer[1,2]

[1] Initiative for Cryptocurrencies and Contracts (IC3), Ithaca, USA
[2] Computer Science Department, Cornell University, Ithaca, USA
[3] Electrical Engineering Department, Technion, Haifa, Israel
soumya@cs.cornell.edu

Abstract. Blockchain-based cryptocurrencies have demonstrated how to securely implement traditionally centralized systems, such as currencies, in a decentralized fashion. However, there have been few measurement studies on the level of decentralization they achieve in practice. We present a measurement study on various decentralization metrics of two of the leading cryptocurrencies with the largest market capitalization and user base, Bitcoin and Ethereum. We investigate the extent of decentralization by measuring the network resources of nodes and the interconnection among them, the protocol requirements affecting the operation of nodes, and the robustness of the two systems against attacks. In particular, we adapted existing internet measurement techniques and used the Falcon Relay Network as a novel measurement tool to obtain our data. We discovered that neither Bitcoin nor Ethereum has strictly better properties than the other. We also provide concrete suggestions for improving both systems.

1 Introduction

Cryptocurrencies are emerging as a new asset class, with a market capitalization of about $150B as of Sept 2017 [15], a growing ecosystem, and a diverse community. The most prominent platforms that account for over 70% of this market are Bitcoin [57] and Ethereum [28,70]. The underlying technology, the *blockchain*, achieves consensus in a decentralized, open system and enables innovation in industries that conventionally relied upon trusted authorities. Some examples of such services include land record management [3], domain name registration [51], and voting [55]. The key feature that empowers such services and makes these platforms interesting is *decentralization*. Without it, such services are technologically easy to construct but require trust in a centralized administrator.

Decentralization is a property regarding the fragmentation of control over the protocol. In the Bitcoin and Ethereum protocols, users submit transactions for miners to sequence into blocks. Better decentralization of miners means higher resistance against censorship of individual transactions. For communication, Bitcoin and Ethereum also have a peer-to-peer network for disseminating block and

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 439–457, 2018.
https://doi.org/10.1007/978-3-662-58387-6_24

transaction information. Both Bitcoin and Ethereum also contain *full nodes*, which serve two critical roles: (1) to relay blocks and transactions to miners (2) and to answer queries for end users about the state of the blockchain. Understanding the network properties of full nodes is crucial for protocol design and analysis of each network's resilience to attacks. Ongoing research explores ways to make the Bitcoin and Ethereum networks more decentralized without measurements on the underlying network. Hence, debates and decisions about the underlying networks are often based on assumptions rather than measurement.

In this paper, we present a comprehensive measurement study on decentralization metrics in these operational systems and shed light on whether or not existing assumptions are satisfied in practice. We adapt prior Internet measurement techniques for Bitcoin and Ethereum and use novel approaches to obtain application layer data. Our main data sources are (1) direct measurements of these networks from multiple vantage points, (2) a Bitcoin relay network called *Falcon* that we deployed and operated for a year, and (3) blockchain histories of Bitcoin and Ethereum. Our study presents findings regarding the network properties, impact of protocol requirements, security, and client interactions.

This paper makes three contributions. First, it provides new tools and techniques for measuring blockchain-based cryptocurrency networks. The key tool introduced here is the Falcon relay network that we built to serve as a backbone for ferrying blocks. This network was deployed for Bitcoin across five continents, providing a unique vantage point on pruned blocks. Second, we perform a comparative study of decentralization metrics in Bitcoin and Ethereum. Our key findings are: (1) the Bitcoin network can increase the bandwidth requirements for nodes by a factor of 1.7 and keep the same level of decentralization as 2016, (2) the Bitcoin network is geographically more clustered than Ethereum, with many nodes likely residing in datacenters. (3) Ethereum has lower mining power utilization than Bitcoin and would benefit from a relay network, and (4) small miners experience more volatility in block rewards in Bitcoin than Ethereum.

2 Bitcoin and Ethereum

Bitcoin and Ethereum use Nakamoto consensus [5–7,38,57] to regulate transaction serialization in their blockchains. While architecturally very similar, these systems differ significantly in terms of their API, abstractions, and wire protocol.

2.1 The Bitcoin Protocol

Bitcoin is a protocol that sequences transactions into groups called blocks. The protocol targets a block production interval of 10 min with a maximum size of 1 MB. At the time of our measurements, the last 100 blocks had a 0.99 MB median block size and a 9.8 min mean interval. The wire protocol implements a peer-to-peer network based on flooding block and transaction announcements.

The peer to peer network is formed through point to point links. To form a link, clients establish a TCP connection and perform a protocol-level three-way

handshake. The protocol-level handshake exchanges the state of each client, such as the height of the blockchain and a version string associated with the software being run. When a client discovers or receives a new block, it floods the network with the hash of the block. If a neighboring client needs the block, it requests the block based on the hash value. There are many different block formats, such as compact [17] and Merkle [44] blocks, but we focus on retrieval of full blocks.

2.2 The Ethereum Protocol

The Ethereum protocol [28] focuses on providing a platform to facilitate building decentralized applications on its blockchain. To sequence transactions, Ethereum adopts a design inspired by Nakamoto consensus and the GHOST protocol [64].

Ethereum adopts a chain selection rule to harness the residual mining power in pruned blocks for improved security. The protocol includes such blocks, called *uncles*, in its blockchain and rewards the corresponding miners [70]. Ethereum targets a block interval between 10 to 20 s [41]. The block size is indirectly determined by an execution fee, called *gas*, that fluctuates over time. At the time of our measurements, the last 100 blocks were generated with a 2.9 KB median block size and a 16.3 s average interval.

Ethereum employs a UDP-based node discovery mechanism inspired by Kademlia [54], but the rest of the P2P communication is over TCP. Unlike Bitcoin, messages between nodes are encrypted and authenticated. Ethereum's wire protocol is poorly documented, so we rely on the client implementations [19,42,60,61] and Ethereum wiki pages [25–27,29,30] for information.

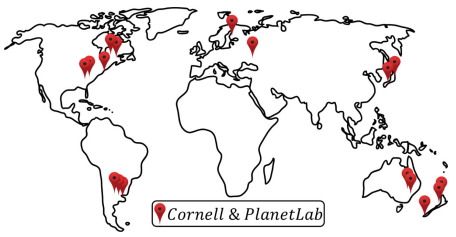


Fig. 1. The measurement infrastructure is built on 18 globally distributed nodes.

In Ethereum, clients request blocks by the corresponding block hash. Older clients request blocks, which consist of a body and header, while newer clients request each piece separately. The measurement system in this paper focuses on retrieval of full blocks and block bodies.

3 Measurement Infrastructure

Blockchain-based cryptocurrencies operate on global peer-to-peer networks that span multiple administrative domains. Measurement of such networks concerns the exploration of the relationship between peers, the capabilities of individual peers, and the properties of the system as a whole–e.g. its security and fairness. To characterize Bitcoin and Ethereum, we deployed *Blockchain Measurement System* (*BMS*), a measurement system than ran experiments of varying duration–from a few days up to 12 months.

Table 1. Timeline of measurements. All dates are in 2017 unless otherwise noted.

Measurement	Network		Num. Nodes	Dates
Bandwidth (All) Latency (BTC IPv4) *(Single beacon)*	BTC	IPv4	3441	Jan 11–16;Jan 30–Mar 16
		IPv6	515	Jan 13–14; Apr 20–25
		Tor	127	Jan 13; Apr 23–25
	ETH	IPv4	285	Mar 27–Apr 25
Peer-to-Peer Latency *(Mult. Vantage Pts.)*	BTC	IPv4	3390 (5.7M edges)	Jan 10–15; Jan 30–Mar 01
	ETH	IPv4	4302 (9.3M edges)	Mar 01–Apr 11
Latency *(Single Beacon)*	BTC	IPv6	845	Jan 13–14; Feb 03–Apr 25
Pruned Blocks	BTC	IPv4	5977	May 5 2016–Apr 29

Network Properties. BMS uses multiple vantage points in order to gain a comprehensive view of the cryptocurrency networks. To capture the evolution of these networks, BMS has been continuously collecting data regarding the provisioned bandwidth of peers and peer-to-peer latency. BMS first connects to a peer, collects measurements, and then disconnects before proceeding to the next peer. These measurements target (1) Bitcoin nodes connected over IPv4, IPv6, and Tor [23] and (2) Ethereum nodes connected over IPv4. As of May 2017, Ethereum does not have any Tor nodes mainly because Tor is exclusively TCP, whereas Ethereum node discovery is UDP-based. Moreover, this study excludes Ethereum's IPv6 network because BMS was unable to discover enough nodes to reach generalized conclusions. Table 1 shows the timeline of the data collection for each network and the number of nodes measured in each measurement.

To estimate the peer-to-peer latency, BMS uses multiple vantage points geographically distributed across the world. Figure 1 shows the geographic distribution of the measurement infrastructure. 15 out of 18 nodes reside in PlanetLab's global research network [14] and the remaining three nodes are part of Cornell's academic network, located in Ithaca, NY.

To measure the provisioned bandwidth of nodes in Bitcoin and Ethereum, BMS used nodes with extensive resources. In particular, measuring the maximum bandwidth that Bitcoin and Ethereum nodes have access to requires nodes with (1) high download capacities to ensure that the bottlenecks are not in the measurement apparatus, and (2) sufficient disk capacities to store detailed results. Since these machines need access to orders of magnitude higher bandwidth capacity than what is achievable on shared infrastructure, such as PlanetLab nodes, some BMS data was collected using dedicated, well-provisioned beacon nodes located at Cornell University.

Finally, BMS needs to pick a sample of nodes from the Bitcoin and Ethereum networks. As a sample, BMS uses a list containing nodes from Bitcoin and Ethereum node crawling sites [1,31], and a locally deployed Ethereum supernode configured with a high peer limit. Interpretations in this paper assume that inferences made from the reachable public nodes are representative of their entire networks. In reality, these networks contains nodes that are not visible to the

public, e.g. they are behind a NAT or a firewall. One such class of nodes are part of *mining*. While much of the mining infrastructure is private, prior measurement work shows that mining operations often have gateway nodes to communicate with the peer-to-peer network [56]. The properties of internal mining pool nodes are orthogonal to the focus of this paper.

Blockchain Information. A naive approach to obtaining information about the blockchain would be to simply run a Bitcoin and Ethereum node. However, this precludes information that cannot be obtained through the respective wire protocols. Many important decentralization metrics center around the analysis of blocks that are not part of the main blockchain. In Ethereum, many of these blocks become uncles which can simply be requested through the wire protocol.

In Bitcoin, however, a block that is not part of the main blockchain simply becomes *pruned*. Pruned blocks in Bitcoin have no effect on the state of the system, they are deleted by clients without impacting correctness. Thus, it is crucial to connect directly to miners to capture pruned blocks.

A critical component of BMS to observe pruned blocks is the Falcon Relay Network, which relays blocks between Bitcoin miners. The Falcon Relay Network uses cut-through routing to quickly disseminate blocks worldwide, which incentivizes miners to connect to Falcon. Indeed, Falcon is directly connected to at least 36.4% of the entire hashpower in Bitcoin. Since there is just one other operational relay network for Bitcoin [16,18], Falcon has observed blocks that have not been seen on other well-connected nodes [8].

4 Measurements

In this section, we present the measurements taken by BMS. In each measurement, we describe the methodology, followed by the results of our analysis. As with any measurement study of a large-scale, uninstrumentable artifact, measurements are not perfect; we conclude each section by addressing some potential sources of error and their mitigation.

4.1 Provisioned Bandwidth

Provisioned bandwidth is an estimate on a node's transmission capacity characterizing how much bandwidth the node has to communicate with the rest of the cryptocurrency network. Greater provisioned bandwidth helps miners to propagate/collect blocks to/from the network faster. Thus, it becomes more difficult for a malicious miner to situate themselves in the network to achieve the rushing property [35] and attack the blockchain. Knowledge of provisioned bandwidth also aids in setting protocol parameters, such as the block size and frequency.

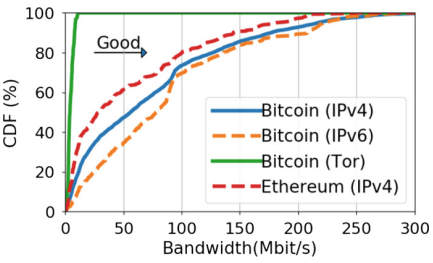
Methodology. BMS measures the provisioned bandwidth of each peer by requesting a large amount of data from each peer and seeing how fast the peers can stream the data to BMS's measurement nodes. BMS does this by asking for blocks that were first seen over a year ago – similar to how a stale node asks

for blocks to sync state. Each request asks for the same set of blocks in Bitcoin and blocks or the corresponding bodies in Ethereum. Next, BMS divides the time into epochs and records the number of bytes received during each epoch. This process continues until either BMS receives all data or a predefined timeout of 30 s is reached. A long timeout helps BMS eliminate effects from TCP slow start and other initialization noise as well as identify and eliminate spurious spikes in throughput caused by buffering in the kernel by BMS. Finally, BMS processes the collected data to determine the provisioned bandwidth. To do so, first, it identifies the independent data streams by combining successive epochs containing active data transfers. Then, it eliminates streams that are shorter than 500 ms to mitigate initialization artifacts such as TCP slow start. BMS then outputs the maximum observed throughput among the remaining distinct continuous streams as the provisioned bandwidth of the remote peer.

The experiments in this paper are run on servers with 1 Gbps links at Cornell University. This has not changed from 2016 to 2017, which allows us to make comparisons to a previous study in 2016 [20].

	Bitcoin			**Eth.**
	IPv4 [Mbps]	IPv6 [Mbps]	Tor [Mbps]	IPv4 [Mbps]
10%	5.7	11.0	2.1	3.4
33%	23.3	45.2	3.1	11.2
50%	56.1	78.2	4.1	29.4
67%	91.1	94.3	5.6	68.3
90%	177.0	207.9	8.1	144.4
Avg.	73.1	86.5	4.7	55.0
Std. Dev.	68.4	66.9	2.4	58.8

(a) Provisioned bandwidth statistics.



(b) CDF

Fig. 2. Statistics on distribution of provisioned bandwidth and the CDF.

Results. Table 2a summarizes per-node bandwidth results that BMS has collected. We see that Bitcoin nodes in both IPv4 and IPv6 networks have consistently higher bandwidth compared to Ethereum IPv4 nodes. In particular, the median Bitcoin IPv4 and IPv6 nodes have about 1.9× and 2.7× the bandwidth of the typical Ethereum IPv4 node. In contrast, Bitcoin Tor nodes have an order of magnitude lower bandwidth compared to directly connected nodes, though they are not unusable – e.g. 90% has more than 2 Mbit/s. Ongoing research explores alternatives to the Tor network that also provide efficient communication [50].

Figure 2b shows the cumulative distribution of the bandwidth measurements. Steep increases in the Bitcoin IPv4/IPv6 curves at around 10 Mbps and 100 Mbps regions represent typical bandwidth capacities of a home user, and a typical Amazon EC2 Bitcoin instance. For Ethereum, we observe a similar accumulation around 10 Mbps region, but the bandwidth is more evenly distributed over the

remaining nodes. As the long tailed distribution and higher standard deviation indicates, bandwidth of Bitcoin IPv4/IPv6 nodes are spread out over a wider range of values compared to Ethereum nodes. While the most well provisioned Bitcoin nodes have around 300 Mbps of spare bandwidth, the highest Ethereum node capacity that BMS has observed is limited to 250 Mbps.

One of the most interesting discoveries of this study is that the Bitcoin network has improved tremendously in terms of its provisioned bandwidth. The results show that Bitcoin IPv4 nodes, which used to be connected to the network with a median bandwidth of 33 Mbit/s in 2016 [20], have a median bandwidth of 56 Mbit/s, as of February 2017. In other words, the provisioned bandwidth of a typical full node is now 1.7× of what it was in 2016.

Critical system parameters, such as the maximum block size and block frequency, can be increased when the provisioned bandwidth increases. The increase in provisioned bandwidth suggests that the block size can be increased by a factor of 1.7 without increasing centralization beyond its de facto level in 2016.

Caveats. As with every measurement technique in the real world, our results above are subject to experimental limitations and expected errors. The accuracy of the measurements may drop under certain circumstances, including the cases where: (1) the network bottleneck lies on the side of the measurement beacon rather than the remote peer, (2) network traffic on the side of BMS interferes with the collected results, (3) the remote peer intentionally shapes the traffic to selectively limit the bandwidth available to BMS, for instance via bandwidth throttling, and (4) different steady state bandwidth between Bitcoin and Ethereum, skewing the numbers for one system over another The setup of our bandwidth infrastructure helps minimize potential inaccuracies due to the first two issues. Moreover, analysis of popular Bitcoin [5] and Ethereum client implementations [19,42,60,61] shows that the third case is not supported by this software and would require additional, potentially non-trivial, work to set up. To verify the impact of the last issue, we ran an Ethereum and Bitcoin client and saw that their bandwidth consumption differed by 0.2 Mbps, which introduces about a 1% error on our measurements above.

In addition to our analysis above, we also expect to see certain artifacts in our data. As noted above, we see clusters of nodes around 10 Mbps and 100 Mbps, which are typical bandwidth capacities of home and EC2 users, respectively.

4.2 Network Structure

The structure of the peer-to-peer network impacts the security and performance for cryptocurrencies. A geographically clustered network can quickly propagate a new block to many other nodes. This makes it more difficult for a malicious miner to propagate conflicting blocks/transactions quicker than honest nodes. However, a less clustered network may mean that full nodes are being run by a wider variety of users which is also good for decentralization.

Methodology. Since it is not possible to obtain direct measurements between peers we do not control, we use the state of the art estimation techniques to establish bounds and gain insights into network structure.

Single Beacon Latency. We first collect direct ICMP ping measurements from BMS nodes to all peers in the network. We report the minimum observed ping latency, as it provides a physical bound on the distance to the BMS beacon.

Peer-to-Peer Latency. Measuring the peer-to-peer latency requires access to the end points. In both Bitcoin and Ethereum, peers do not reveal their neighbors. Hiding the network structure boosts privacy and security [45,56], but also makes it harder to infer properties about the network. BMS provides latency estimates for a superset of the actual links between known peers. We do not normalize for the slightly different network sizes, 3390 for Bitcoin and 4302 for Ethereum, as our samples from both networks were very similar. Since measuring peer-to-peer latencies directly is not feasible, we establish bounds from observed latencies from multiple beacons, using techniques from prior literature [37]. BMS starts with the measurements taken from a single beacon. Then, it uses the triangle inequality to estimate the upper and lower bounds for the latency between peers. Repeating this process from other vantage points yields a set of bounds for each pair of peers. Finally, BMS determines a range for latency estimates between each peer by picking the maximum lower bound and the minimum upper bound. The paper also presents the average of the lower bound and upper bound latency between peers. In this study, BMS includes nodes that do not support the DAO fork [10] in its measurements for Ethereum.

Geographical Distance. BMS takes the minimum of repeated latency measurements to eliminate transient network effects and capture the geographic distance between two nodes [13,43,69]. BMS also uses IP geolocation data to calculate distances between peer nodes as an additional validation on our results. To calculate distances, BMS applies the Haversine formula [63] using the coordinate values gathered from an IP-based geolocation service [46].

Results. Our measurements indicate considerable differences between P2P latencies of Bitcoin and Ethereum IPv4 networks, summarized in Table 2 and PDF graphed in Fig. 3.

Table 2. Min single beacon latencies observed and P2P latency estimates.

	Single beacon		Peer-to-Peer	
	Bitcoin		Bitcoin	Eth.
	IPv4 [ms]	IPv6 [ms]	IPv4 [ms]	IPv4 [ms]
10%	29	40	48	92
33%	78	80	79	125
50%	89	95	109	152
67%	98	95	152	200
90%	201	165	286	276
Avg.	97	103	135	171
Std. Dev.	59	62	88	76

We find that Bitcoin has many more nodes that are closer geographically than Ethereum. Figure 3 shows that Ethereum's most likely latencies are centered around 120 ms, while Bitcoin nodes tend to be clustered around 50 ms. Only 13% of Ethereum latencies are under 100 ms, while Bitcoin has a surprisingly high 46%.

Additionally, the estimated peer-to-peer latency between Ethereum nodes is 26.7% higher than Bitcoin on average. This geographic proximity between nodes, along with the observation that Bitcoin has many nodes with 100 Mbps of provisioned bandwidth (see Sect. 4.1), seems to indicate that many Bitcoin nodes are run in datacenters. 56% of Bitcoin's nodes and 28% of Ethereum's nodes belong to an autonomous system that provides dedicated hosting services, a difference significant at the 1% significance level.

Indeed Ethereum nodes are not accumulated in a single geographical region, but are more evenly distributed around the world. Figure 3c shows the CDF of distances between peer to peer nodes based on IP geolocation information. The results corroborate our findings based on network latency measurements and show that Ethereum nodes are geographically further apart than Bitcoin. As additional evidence, when we use geolocation on the P2P distances and plot the CDF in Fig. 3c, we see that Ethereum nodes are further apart than Bitcoin.

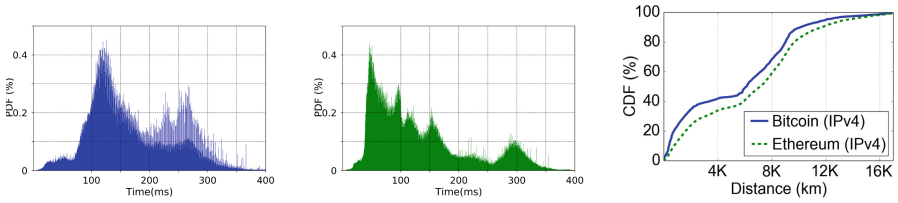


Fig. 3. The histogram of P2P latencies in Ethereum (a) and Bitcoin (b), as well as the CDF of geographical distances (c).

Sanity Checks. The first two columns of Table 2 present single beacon latency in Bitcoin IPv4/IPv6 networks. The results indicate that both the median and the average latency to IPv4 nodes are smaller than IPv6 nodes. As there are fewer IPv6 nodes than IPv4 nodes, we expect this result since IPv4 nodes are more likely to be closer to our beacons.

While there has been a large body of work showing the prevalence of triangle inequality violations in the Internet [12,52,67], there are several reasons BMS's measurements are not affected significantly. First, such violations were shown to occur less than 5% of network snapshots [52]. Since we take the minimum latency observed from a beacon, triangle inequality violations will only occur in our dataset less than 1% of the time [52]. TIVs are also significantly less prevalent when dealing with latencies less than 300 ms, which includes almost our entire dataset [67]. To ensure that the above results hold for our dataset as well, we used a geolocation service as ground truth to verify our results.

One other limitation in our study is that it is impossible to collect measurements using ICMP pings from nodes that block ICMP traffic and from Tor nodes that only communicate over TCP.

4.3 Distribution of Mining Power

Mining on cryptocurrency networks is a complex process that typically requires large computation power. With the current mining difficulty of Bitcoin and Ethereum, using commodity hardware to generate blocks is not feasible [21] which centralizes the mining process somewhat. However, as long as there are many different entities mining, the system is still decentralized. We compare the decentralization of the mining process between Bitcoin and Ethereum.

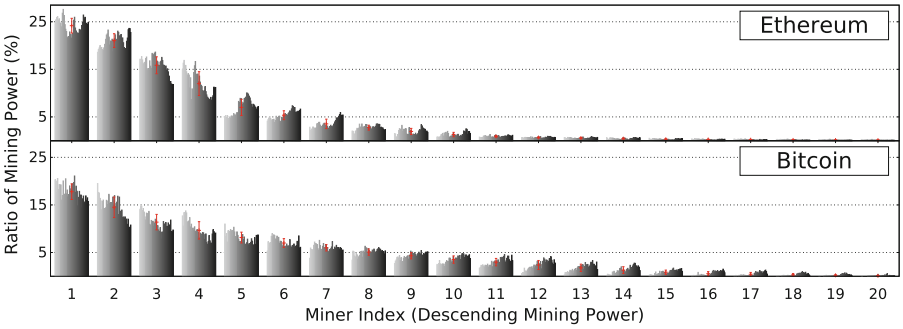


Fig. 4. Distribution of mining power in Bitcoin and Ethereum networks. Bars indicate observed standard deviation from the average.

Methodology. To identify the power of miners in Bitcoin and Ethereum, we examined their weekly distribution over the last 10 months starting on July 15, 2016. Our mining power estimations are based on the ratio of main chain blocks generated by distinct entities. Hence, pruned blocks in Bitcoin and uncles in Ethereum do not affect these estimations. In both networks, miners voluntarily disclose their identity as part of each block they mine. We gathered this data from a public API for Bitcoin [9] and a blockchain explorer for Ethereum [32]. In Bitcoin, 1.8% of the blocks were unidentified, which we treated as if they were generated by distinct individual miners. Finally, we manually processed identities to detect and merge duplicates. This includes pools operated by the same administrator [47] and multiple identities representing the same pool, which we identified by looking for the same pool name with a corresponding tag, e.g. 'DwarfPool1' and 'DwarfPool2'. While it is important to note that miners can be either solo miners or mining pools, this distinction is immaterial for the purposes of this analysis. The argument that mining pools provide a degree of decentralization due to mining pool participants having a check on pool operator behavior has no empirical support. For instance, censorship attacks by pool operators are difficult, if not impossible, to detect by pool participants. Additionally, when miners exceeded the 51% threshold on three separate occasions in Bitcoin's history, the pool participants did not disband the pool despite clear evidence of a behavior widely understood to be unacceptable. Most crucially,

whether mining pools provide a degree of decentralization is inconsequential for the purposes of this paper, which provides an accurate historical account. We report what happened at the time the blocks were mined, as recorded on the blockchain. As such, it is immaterial whether the miners were part of a pool or whether they were solo miners. At the time a block was committed to the chain, pool participants were plaintively cooperating as part of the same mining entity.

Results. For each week of the analysis period, we calculated the corresponding mining power of entities and ranked each miner accordingly. Figure 4 shows the top 20 weekly mining power distribution in the Ethereum and Bitcoin networks. Each group of bars represents a chronologically ordered collection of weekly *mining power ratios*, defined as the fraction of blocks contributed by a miner.

Figure 4 illustrates that, in Bitcoin, the weekly mining power of a single entity has never exceeded 21% of the overall power. In contrast, the top Ethereum miner has never had less than 21% of the mining power. Moreover, the top four Bitcoin miners have more than 53% of the average mining power. On average, 61% of the weekly power was shared by only three Ethereum miners. These observations suggest a slightly more centralized mining process in Ethereum.

Although miners do change ranks over the observation period, each spot is only contested by a few miners. In particular, only two Bitcoin and three Ethereum miners ever held the top rank. The same mining pool has been at the top rank for 29% of the time in Bitcoin and 14% of the time in Ethereum. Over 50% of the mining power has exclusively been shared by eight miners in Bitcoin and five miners in Ethereum throughout the observed period. Even 90% of the mining power seems to be controlled by only 16 miners in Bitcoin and only 11 miners in Ethereum. Hence, both platforms rely heavily on very few distinct mining entities to maintain the blockchain. Indeed, we see in Fig. 5 that the mining power trends can be fit as exponential distributions with curves $0.21e^{-0.19x}$ and $0.35e^{-0.30x}$ in Bitcoin and Ethereum, respectively. These curves yield a coefficient of determination value of 0.99.

These results show that a Byzantine quorum system [53] of size 20 could achieve better decentralization than proof-of-work mining at a much lower resource cost. This shows that further research is necessary to create a permissionless consensus protocol without such a high degree of centralization.

Sanity Checks. Similar to other works in the literature [58,68], we assume that miners accurately self-identify themselves. A miner that contributes a significant portion of the hash power to the cryptocurrency can exert some amount of influence over protocol changes. Thus, it is likely that miners will want to claim blocks that they generated as their own. While strong miners gain political clout and attract more members, getting too large raises alarms among the community about centralization. Thus, such miners may

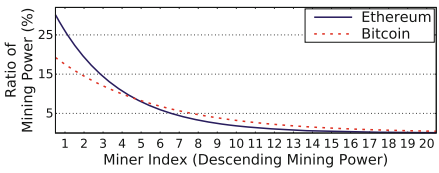


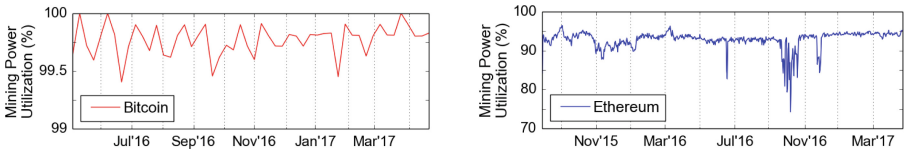
Fig. 5. Exponential trendlines for the average distribution of mining power.

conceal or obfuscate this information to appear less powerful – e.g. by generating multiple identities. For instance, two major mining pools, Ethpool and Ethermine, publicly reveal that they share the same admin [47]. Thus, any analysis based on the voluntary miner data skews toward a more decentralized network than the reality.

4.4 Mining Power Utilization

Mining power utilization [34], which measures the fraction of mined blocks that remain in the main chain, is a metric for evaluating the efficiency of a protocol, as well as a second order metric for robustness against rollbacks. As mining power utilization increases, the protocol is able to convert more of the energy spent to useful work, and therefore the cost to launch an attack is higher.

Methodology. To study the mining power utilization, we analyzed weekly and daily distribution of pruned blocks in Bitcoin and uncles in Ethereum, compared to the main chain blocks. We retrieved this data from (1) the Falcon network, (2) a local Bitcoin client, and (3) public blockchain explorers for Bitcoin [9] and Ethereum [32]. In particular, the Bitcoin blockchain explorer and Falcon exclusively provided 12% and 20% of the total 124 pruned blocks, respectively. Both sources commonly discovered the remaining 68%.



(a) Bitcoin- weekly MPU from 99%

(b) Ethereum- daily MPU from 70%

Fig. 6. Mining power utilization (MPU) for Bitcoin and Ethereum

Results. Figure 6a and b show weekly and daily distributions of mining power utilization in Bitcoin and Ethereum networks, respectively. The results show that Bitcoin utilization is always above 99%, which means that a pruned block in Bitcoin is a relatively rare event. In contrast, daily utilization in Ethereum is typically between 90% to 94% range and never goes above the 97% threshold. During 2016, Ethereum faces occasional drops in its utilization down from 74% to 88%, including (1) the days following the exploitation of the DAO vulnerability [10] from June 17 to 18, (2) attacks on Ethereum network [11,66] between September 22 to October 19, and (3) the days following the Spurious Dragon hard fork [48] between November 23 to 29. These results indicate a strong relationship between mining power utilization and real life events in Ethereum. This may be the result of preventive measures that spam the network to slow down the DAO attacker, bad actors generating blocks with excessive resource demands,

and miners with outdated clients. These results indicate that a relay network, like Falcon, would be greatly beneficial to the Ethereum network.

Sanity Checks. The design of the Ethereum protocol requires peers to store and propagate uncle blocks, which are not on the main chain. In contrast, Bitcoin's blockchain only stores the main chain so peers do not propagate pruned blocks. Hence, capturing such blocks in Bitcoin requires actively watching the network. While the Falcon relay network provides a strong incentive to miners to relay blocks through it, some miners may choose not to do so. Consequently, we may be missing some pruned blocks that were generated by the Bitcoin network.

4.5 Fairness

Section 4.3 presented the mining power distribution, which looks at the main chain presence of miners. The impact of this distribution on a miner's pruned block rate is unclear. To study this relationship, we examine *fairness* defined as the ratio of a miner's share of pruned blocks to her mining power. In a fair protocol, miners generate pruned blocks proportional to their mining power; hence, the fairness is close to 1. A fairness greater than 1 implies that the miner is at a disadvantage, while a fairness less than 1 implies that the miner has an advantage.

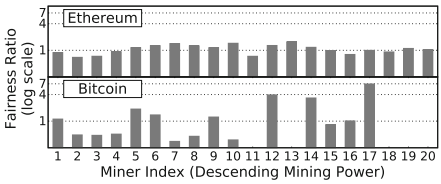


Fig. 7. Fairness distribution. Zero fairness means no pruned block from miner.

Methodology. We used the Falcon network, and a Bitcoin blockchain explorer [8] to retrieve pruned Bitcoin blocks. These sources have, respectively, provided 109 and 99 blocks, yielding a total of 124 distinct pruned blocks. We collected uncles from an Ethereum blockchain explorer [32].

Similar to Sect. 4.3, our results here also assume that miners voluntarily identify themselves in uncles/pruned blocks. Another caveat here lies in gathering pruned blocks. While we incentivize miners to relay blocks through Falcon, there is no guarantee that they necessarily will do so. We suspect that explicit storage of uncles in Ethereum captures a larger proportion of pruned blocks.

Results. Figure 7 shows the distribution of fairness of 20 miners with the highest mining power. The results indicate that, in both networks, the top four miners generally are more successful at appending blocks to the main chain. We run the Kolmogorov-Smirnov goodness of fit test with a p-value of 0.01 to compare the fairness distributions of Bitcoin and Ethereum. Perhaps surprisingly, we see that the fairness of Ethereum and Bitcoin differ significantly from each other keeping a constant time period. The reason for this difference is a much larger standard deviation in Bitcoin's miner fairness compared to Ethereum (1.72 versus 0.25). The mean of both fairness distributions, however, are very similar, with Ethereum at 1.08 and Bitcoin at 1.22.

A high variance results in centralization pressure since smaller miners will have a more difficult time affording the loss of revenue due to a transiently high fairness score. This high variance is a direct result of a significantly smaller number of blocks being generated in Bitcoin. Since Ethereum has a higher block frequency than Bitcoin, smaller miners have a more predictable payoff than larger miners. This makes Ethereum more predictable to mine for smaller miners due to the lower variance in block rewards. Thus, it is important for blockchain protocols to take variance of the block rewards in addition to the mean.

Simply increasing the block frequency may not be the solution to decrease the variance of block rewards since the mining power distribution may be affected as well. The increased block frequency in Ethereum may be part of the cause of the slightly more centralized mining power distribution (see Sect. 4.3).

Sanity Checks. Similar to Sect. 4.4, our results here also assume that miners voluntarily identify themselves in uncles/pruned blocks. As before, if the miners are lying, they are likely to present a more fair system than reality. Another caveat here lies in gathering pruned blocks. While we incentivize miners to relay blocks through Falcon, there is no guarantee that they will. We suspect that explicit storage of uncles in Ethereum allows for more accurate analysis.

Finally, Bitcoin has a significantly lower block generation frequency than Ethereum. On top of that, Bitcoin also has a lower pruned block rate than Ethereum does, which means it has significantly fewer pruned blocks. Thus, this fairness metric is much noisier in Bitcoin compared to Ethereum.

5 Related Work

Network measurements in blockchain-based systems have mainly focused on Bitcoin. One such study [22] demonstrated that the latency is the dominating factor in propagation of blocks smaller than 20 KB. Following work [20] has shown that (1) this limit has increased to 80 KB and (2) nodes are provisioned with substantially higher bandwidth capacity than what the protocol demands. Feld et al. [36] pointed out a strong AS-level centralization that may impact Bitcoin network's connectivity – i.e. 10 ASes contain over 30% of peers. Recent work [2] presented the level of vulnerability, showing that 13 ASes cover the same fraction of peers, but only 39 IP prefixes host half of the overall mining power. Ours is the first work that does a similar type of study on Ethereum as well.

Other work studied various aspects of the Bitcoin overlay network. Miller et al. [56] found that a small fraction of the network, containing around 100 nodes, represents more than 75% of the mining power. The study conjectured that these nodes are well-connected to major mining pools; hence, provide higher efficiency in broadcasting blocks. Biryukov et al. [4] examined how peer neighbors discover IP addresses that correspond to pseudonymous identities. Another study [49] deanonymized peers by observing anomalous relaying behavior in network. Pappalardo et al. [59] observed that low value transactions may experience waiting times of over a month. Other work measured churn and geolocated peers [24]. Gervais et al. [40] discussed centralization concerns regarding the

client development process, distribution of mining power, and spendable coins. Most of these works focus on attacks and the structure of the overlay network, while this work focuses on the resource capabilities of the nodes used in the overlay network.

Recent work presented ways to reduce resource requirements to participate in blockchain systems. Such solutions enhance decentralization by increasing the diversity of participants. Aspen [39] achieves this through sharding the blockchain. In this system, users store, process, and propagate only the data that is relevant to them, hence need fewer resources to join the network. Another approach [62] relies on authenticated data structures to reduce load on nodes. Relay networks increase network efficiency through faster block propagation. The first such system [16] achieved this by avoiding full block verification and retransmitting known transactions. Falcon, the source of pruned block data in the Bitcoin network in this paper, relies on cut-through routing for faster block propagation. Finally, FIBRE incorporates cut-through routing with compact blocks [17] and forward error correction over UDP. The novelty in our work was utilizing Falcon data in order to gain insights into transient application layer information.

Blockchain explorers [8,32,33,65] provide a variety of data on cryptocurrency networks, including online blockchain history; statistics on blockchain components, transaction fees, and market value; and node information. While these sources of information are useful to the community, this work scientifically tests whether the intuitions provided by these sources of information indeed hold.

6 Conclusion

Decentralization in blockchain-based platforms is a component of the value proposition these systems offer. This work presents a comparative assessment of decentralization in two most popular cryptocurrencies, Bitcoin and Ethereum. To do so, it relies on novel measurement techniques to obtain application layer information using the Falcon Network and the application of well-established internet measurement techniques.

Our observations show that Bitcoin has a higher capacity network than Ethereum, but with more clustered nodes likely in datacenters. We also observe that Bitcoin and Ethereum have fairly centralized mining processes and that further research is needed to decentralize permissionless consensus protocols further. In Ethereum, the block rewards have less variance than Bitcoin's. Finally, Ethereum has a lower mining power utilization than Bitcoin, likely due to the high block frequency.

Further, we see that Bitcoin has undergone tremendous growth and can increase the block size by a factor of 1.7x without any decrease in decentralization compared to 2016. Additionally, our study uncovers that the volatility of mining rewards is an important, but often ignored, metric. Finally, we see that Ethereum would likely benefit from a relay network to increase its mining power utilization.

Acknowledgements. The authors thank Vitalik Buterin and the anonymous reviewers for their feedback on earlier drafts of this manuscript. Ittay Eyal is supported by the Viterbi Fellowship in the Center for Computer Engineering at the Technion. This work was partially funded and supported by AFOSR grant F9550-16-0250, NSF CSR-1422544, NSF CNS-1601879, NSF CNS-1544613, NSF CCF-1522054, NSF CNS-1518779, NSF CNS-1704615, ONR N00014-16-1-2726, NIST Information Technology Laboratory (60NANB15D327, 70NANB17H181), Facebook, Infosys, and IC3, the Initiative for Cryptocurrencies and Smart Contracts. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650441. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. 21.CO. Bitnodes. https://bitnodes.21.co/. Accessed June 2017
2. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: routing attacks on cryptocurrencies. arXiv preprint arXiv:1605.07524 (2016)
3. Benben Team. Benben. http://benben.com.gh/. Accessed Oct 2016
4. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in Bitcoin P2P network. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS), pp. 15–29 (2014)
5. Bitcoin Community. Bitcoin source. https://github.com/bitcoin/bitcoin. Accessed June 2017
6. Bitcoin Community. Protocol rules. https://en.bitcoin.it/wiki/Protocol_rules. Accessed June 2017
7. Bitcoin Community. Protocol specification. https://en.bitcoin.it/wiki/Protocol_specification. Accessed June 2017
8. Blockchain Info Team. Blockchain Info. https://blockchain.info/. Accessed May 2017
9. BlockTrail Team. Blocktrail API. https://www.blocktrail.com/api/docs#api_data. Accessed Apr 2017
10. Buterin, V.: Critical update re: DAO vulnerability. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/. Accessed Apr 2017
11. Buterin, V.: Transaction spam attack: next steps. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/. Accessed Apr 2017
12. Cangialosi, F., Levin, D., Spring, N.: Ting: measuring and exploiting latencies between all tor nodes. In Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC), pp. 289–302 (2015)
13. Chandrasekaran, B., et al.: Alidade: IP geolocation without active probing. Technical report, Duke University (2015)
14. Chun, B., et al.: PlanetLab: an overlay testbed for broad-coverage services. ACM SIGCOMM CCR **33**(3), 3–12 (2003)
15. CoinMarketCap. Cryptocurrency market capitalizations. https://coinmarketcap.com/. Accessed May 2017
16. Corallo, M.: The Bitcoin relay network. BIP 152. http://bitcoinrelaynetwork.org/. Accessed May 2017
17. Corallo, M.: Compact block relay. BIP 152. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki. Accessed June 2017

18. Corallo, M.: FIBRE: Fast internet Bitcoin relay engine. https://github.com/bitcoinfibre/bitcoinfibre. Accessed Apr 2017
19. Cpp-ethereum Authors: Ethereum C++ client. https://github.com/ethereum/cpp-ethereum. Accessed Apr 2017
20. Croman, K., et al.: On scaling decentralized blockchains (a position paper). In: Proceedings of the Workshop on Bitcoin and Blockchain Research (BITCOIN), Barbados (2016)
21. Cryptocompare Team. Cryptocurrency mining calculator. https://www.cryptocompare.com/mining/calculator. Accessed June 2017
22. Decker, C., Wattenhofer, R.: Information propagation in the Bitcoin network. In: Proceedings of the IEEE International Conference on Peer-to-Peer Computing, pp. 1–10, Trento, Italy (2013)
23. Dingledine, R., Mathewson, N., Syverson, P.:Tor: the second-generation onion router. In: Proceedings of the USENIX Security Symposium (2004)
24. Donet Donet, J.A., Pérez-Solà, C., Herrera-Joancomartí, J.: The Bitcoin P2P network. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014. LNCS, vol. 8438, pp. 87–102. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44774-1_7
25. Ethereum Community. devp2p forward compatibility requirements for homestead. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-8.md. Accessed Apr 2017
26. Ethereum Community. DΞVp2p wire protocol. https://github.com/ethereum/wiki/wiki/%C3%90%CE%9EVp2p-Wire-Protocol. Accessed Apr 2017
27. Ethereum Community. Ethereum wire protocol. https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol. Accessed Apr 2017
28. Ethereum Community. A next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed Apr 2017
29. Ethereum Community. RLPx: Cryptographic network & transport protocol. https://github.com/ethereum/devp2p/blob/master/rlpx.md. Accessed Apr 2017
30. Ethereum Community. RLPx encryption. https://github.com/ethereum/go-ethereum/wiki/RLPx-Encryption. Accessed Apr 2017
31. EthereumJ. The Ethereum nodes explorer. https://www.ethernodes.org/. Accessed June 2017
32. Etherscan Team. Etherscan: The Ethereum block explorer. https://etherscan.io/. Accessed June 2017
33. Ethstats Team. Ethstats. https://ethstats.net/. Accessed June 2017
34. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-NG: a scalable blockchain protocol. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp. 45–59, Santa Clara, CA, USA (2016)
35. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In Proceedings of the International Financial Cryptography and Data Security Conference, Barbados (2014)
36. Feld, S., Schönfeld, M., Werner, M.: Analyzing the deployment of Bitcoin's P2P network under an AS-level perspective. In: Proceedings of the International Workshop on Secure Peer-to-Peer Intelligent Networks and Systems, vol. 32, pp. 1121–1126 (2014)
37. Francis, P., et al.: IDmaps: a global internet host distance estimation service. IEEE/ACM Trans. Netw. (TON) **9**, 525–540 (2001)

38. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
39. Gencer, A.E., van Renesse, R., Sirer, E.G.: Short paper: service-oriented sharding for blockchains. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 393–401. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_22
40. Gervais, A., Karame, G.O., Capkun, V., Capkun, S.: Is Bitcoin a decentralized currency? In: Proceedings of the IEEE Symposium on Security and Privacy, vol. 3, no. 12, pp. 54–60 (2014)
41. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS), pp. 3–16, Vienna, Austria (2016)
42. Go-ethereum Authors: Official Go implementation of the Ethereum protocol. https://github.com/ethereum/go-ethereum. Accessed Apr 2017
43. Guha, S., Murty, R., Sirer, E.G.: Sextant: a unified framework for node and event localization in sensor networks. In: Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 205–216. ACM (2005)
44. Hearn, M., Corallo, M.: Connection bloom filtering. BIP 37. https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki. Accessed Sept 2017
45. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on Bitcoin's peer-to-peer network. In: Proceedings of the USENIX Security Symposium, pp. 129–144, Washington, D.C., USA (2015)
46. IP Info Team. IP Info. http://ipinfo.io/. Accessed Apr 2017
47. jackwinters. Ethpool & Ethermine voting on the soft fork. https://forum.daohub.org/t/ethpool-ethermine-voting-on-the-soft-fork/5364. Accessed Apr 2017
48. Jameson, H.: Hard fork no. 4: Spurious Dragon. https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/. Accessed Apr 2017
49. Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in Bitcoin using P2P network traffic. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 469–485. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_30
50. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: an efficient communication system with strong anonymity. In: Proceedings of the Privacy Enhancing Technologies Symposium (PETS) (2016)
51. Loibl, A.: Namecoin. namecoin.info (2014)
52. Lumezanu, C., Baden, R., Spring, N., Bhattacharjee, B.: Triangle inequality variations in the internet. In: Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC), pp. 177–183 (2009)
53. Malkhi, D., Reiter, M.: Byzantine quorum systems. J. Distrib. Comput. **11**(4), 203–213 (1998)
54. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_5
55. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 357–375. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_20
56. Miller, A., et al.: Discovering Bitcoin's public topology and influential nodes (2015)

57. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
58. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: generalizing selfish mining and combining with an eclipse attack. IACR Cryptology ePrint Archive **2015**, 796 (2015)
59. Pappalardo, G., Di Matteo, T., Caldarelli, G., Aste, T.: Blockchain inefficiency in the Bitcoin peers network. arXiv preprint arXiv:1704.01414 (2017)
60. Parity Authors: Ethereum Rust client. https://github.com/paritytech/parity. Accessed Apr 2017
61. Pyethapp Authors: Python based client implementing the Ethereum protocol. https://github.com/ethereum/pyethapp/. Accessed Apr 2017
62. Reyzin, L., Meshkov, D., Chepurnoy, A., Ivanov, S.: Improving authenticated dynamic dictionaries, with applications to cryptocurrencies. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 376–392. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_21
63. Sinnott, R.W.: Virtues of the haversine. Sky and Telescope (1984)
64. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 507–527. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_32
65. Statoshi Team: Statoshi info. http://statoshi.info/. Accessed May 2017
66. Swende, M.: Announcement of imminent hard fork for EIP150 gas cost changes. https://blog.ethereum.org/2016/10/13/announcement-imminent-hard-fork-eip150-gas-cost-changes/. Accessed Apr 2017
67. Wang, G., Zhang, B., Ng, T.: Towards network triangle inequality violation aware distributed systems. In: Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC), pp. 175–188. ACM (2007)
68. Wang, L., Liu, Y.: Exploring miner evolution in Bitcoin network. In: Mirkovic, J., Liu, Y. (eds.) PAM 2015. LNCS, vol. 8995, pp. 290–302. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15509-8_22
69. Wang, Y., Burgener, D., Flores, M., Kuzmanovic, A., Huang, C.: Towards street-level client-independent IP geolocation. In: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp. 365–379. USENIX Association (2011)
70. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)

Blockchain Protocols

Anonymous Post-Quantum Cryptocash

Huang Zhang[1,2], Fangguo Zhang[1,2(✉)], Haibo Tian[1,2], and Man Ho Au[3]

[1] School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
isszhfg@mail.sysu.edu.cn
[2] Guangdong Key Laboratory of Information Security, Guangzhou 510006, China
[3] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China

Abstract. In this paper, we construct an anonymous and decentralized cryptocash system which is potentially secure against quantum computers. In order to achieve that, a linkable ring signature based on ideal lattices is proposed. The size of a signature in our scheme is $O(\log N)$, where N is the cardinality of the ring. The framework of our cryptocash system follows that of CryptoNote with some modifications. By adopting the short quantum-resistant linkable ring signature scheme, our system is anonymous and efficient. We also introduce how to generate the verifying and signing key pairs of the linkable ring signature temporarily. With these techniques, the privacy of users is protected, even though their transactions are recorded in the public ledger.

1 Introduction

Electronic currencies or cryptocash systems have been proposed for many years. But none of them is prevalent before the Bitcoin system appears. Bitcoin was first described by Nakamoto in 2008 [25]. Its success is partially due to its properties of decentralization and anonymity. To prevent "double spending", the system maintains the history of transactions among most nodes in a peer-to-peer network. A consensus mechanism called proof-of-work is used to maintain the history.

Later, researchers find that the public history of Bitcoin causes weaknesses which violate its original designing goals. The latest result states that Bitcoin only addresses the anonymity and unlinkability issues partially [3]. For example, multiple public keys of the same user can potentially be linked when a user sends change back to his wallet. In this case, two or more of a single user's public keys will appear in the same transaction [28]. Recently, there are more discussions about the weak anonymity of Bitcoin [27,30]. Although this weakness can be overcome by adopting mixing and distributed methods, the solutions have to include a trusted third party which is a violation to the decentralization property.

There are some creative works to design a strong anonymous cryptocash system. Miers *et al.* [23] proposed "Zerocoin" that allows users to spend their coins using anonymous proof of ownership instead of explicit public-key based digital

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 461–479, 2018.
https://doi.org/10.1007/978-3-662-58387-6_25

signatures. Saberhagen presented two properties, namely, "untraceability" and "unlinkability", that must be possessed in a fully anonymous cryptocash model. Then, they designed CryptoNote system with these properties [31]. Monero is a system based on CryptoNote. In CryptoNote, to provide anonymity, there are two ways for all transactions on the network: (1) hiding the sender's address using ring signatures, (2) hiding the receiver's identity using stealth addresses. Both sending and receiving addresses are verifying keys of a ring signature scheme. A ring signature can also be used in the Zerocoin system [12].

The notion of ring signatures, introduced by Rivest *et al.* [29], permits a user to sign a message on behalf of a group. A verifier is convinced that the real signer is a member of the group, but cannot explicitly identify the real signer. Considering the anonymity of a cryptocash system, a ring signature is obviously more suitable than a standard signature. But there is a cost: the size of the signature and the computational complexity are inherently larger than those of a standard signature. A traditional ring signature scheme usually features a signature size of $O(N)$, where the ring has N participants. To construct a ring signature of $O(\log N)$ or $O(1)$ size was an open problem in this field. Recently, Groth and Kohlweiss proposed a commitment-based scheme with logarithmic signature size [12].

However, a cryptocash system which replaces a standard signature with a ring signature naively suffers from the double spending attack. To fix this problem, it is necessary for the public to determine ring signatures generated by the same key pair. The traceable ring signature [9] is a candidate that enables users to trace the verifying and signing key pair which have been used for signing different messages. But the traceability is redundant for an efficiency-sensitive cryptocash system. CryptoNote and Monero chose to modify the traceable ring signature into a "one-time signature" to reduce the computational cost. Generally speaking, a linkable ring signature [17], which is a variant of the linkable spontaneous anonymous group signature [16], is sufficient enough for cryptocash systems to determine double spending. Even though signatures of these schemes are of size $O(N)$, CryptoNote and Monero do provide better privacy than Bitcoin.

Most cryptocash systems are based on traditional cryptographic schemes. The security of these schemes is based on hard computational problems, such as the factorization and discrete logarithm problem (DLP). However, researchers have proved that a quantum computer is able to solve these problems efficiently so that schemes based on them are not secure under the quantum computing model. One solution is to build schemes on computational problems that remain even hard for quantum computers. Lattice problems have been widely believed as suitable choices to build quantum resistant cryptographic schemes since Ajtai proposed his seminal work [2]. Some post-quantum signature schemes have been proposed recently [7,10,18]. Relying on these schemes, it is easy to obtain a post-quantum cryptocash system by replacing the ECDSA signature scheme in Bitcoin. However, the resulting cryptocash system is simply like Bitcoin in which the transactions are still linkable. Even though there are several lattice-based ring

signatures [6,36,37] including the one with logarithmic size [15], none of them has the linkable or traceable property which is vital to prevent double spending.

In this paper, we aim at designing an anonymous post-quantum cryptocash (APQC) system. In order to achieve this goal, we propose a linkable ring signature based on ideal lattices. The size of a signature in this scheme is $O(\log N)$, where N is the cardinality of the ring. The framework of our cryptocash system follows that of CryptoNote [31], and the ideal-lattice-based signature scheme is inspired by the work of Groth and Kohlweiss [12] with some modifications.

The paper is organized as follows: in Sect. 2, we introduce notations and concepts applied in our work. The model of the ring signature based cryptocash is described in Sect. 3. Section 4 involves the concrete construction of the ideal-lattice-based linkable ring signature. We design the standard transaction of our cryptocash system in Sect. 5. Section 6 is a brief conclusion for this paper.

2 Preliminaries

2.1 Notations

We use $\mathbb{Z}$, $\mathbb{R}$ to denote the set of all integers and the set of all reals, respectively. For any $x \in \mathbb{R}$, $\lceil x \rceil$ denotes the smallest integer that is not smaller than x. A set $\{x_1, \ldots, x_n\}$ is denoted by $\{x_i\}_{i=1}^n$. We use $|S|$ to indicate the cardinality of a set S. Vectors are named by lower-case bold letters (e.g., $\mathbf{x}$) and matrices by upper-case bold letters (e.g., $\mathbf{X}$). For a vector $\mathbf{x}$, $\|\mathbf{x}\|_p$ represents its ℓ_p norm, and p is omitted if $p = \infty$. The norm of a polynomial is defined similarly by regarding it as a vector. The ith entry of a vector $\mathbf{x}$ is denoted by x_i. If $\mathbf{x}$ is a vector of polynomials, then $\|\mathbf{x}\| = \max_i \|x_i\|$. A matrix $\mathbf{X}$ is identified with the ordered set $\{\mathbf{x}_i\}_i$ of its column vectors, and its ℓ_p norm is defined as $\|\mathbf{X}\|_p = \max_i \|\mathbf{x}_i\|_p$. If $a \in R$ and $\mathbf{X}$ is a matrix with entries in ring R, $a\mathbf{X}$ denotes the scalar multiplication. $\mathbf{I}$ is the identity matrix whose dimension is known from the context. For an integer i, i_j symbolizes the jth bit of i. $\delta_{i\ell}$ is Kronecker's delta, *i.e.*, $\delta_{\ell\ell} = 1$ and $\delta_{i\ell} = 0$ for $i \neq \ell$. For two strings x_1 and x_2, $x_1 \| x_2$ denotes the concatenation of them.

2.2 Lattices and Hard Problems

A lattice $\Lambda = \mathcal{L}(\mathbf{B})$ with dimension m and rank n is a subgroup of the linear space $\mathbb{R}^m$. Every element in Λ can be represented as an integral combination of its basis $\mathbf{B} \in \mathbb{R}^{m \times n}$. In our work, we will focus on a specific class of lattices, called ideal lattices, which can be described as ideals of certain polynomial rings.

Definition 1 ([19]). *An ideal lattice is an integer lattice $\mathcal{L}(\mathbf{B}) \subseteq \mathbb{Z}^n$ such that $\mathcal{L}(\mathbf{B}) = \{g \bmod f : g \in \mathcal{I}\}$ for some monic polynomial f of degree n and ideal $\mathcal{I} \in \mathbb{Z}[x]/\langle f \rangle$.*

The quotient ring $\mathbb{Z}[x]/\langle f \rangle$ is additively isomorphic to the integer lattice $\mathbb{Z}^n$.

To extend the hash function family in previous works [2,5,22], Micciancio defined the generalized knapsack function family [20,21].

Definition 2 ([21]). *For any ring R, subset $D \subset R$ and integer $m \geq 1$, the generalized knapsack function family $\mathcal{K}(R, D, m) = \{f_{\mathbf{a}} : D^m \to R\}_{\mathbf{a} \in R^m}$ is defined by*

$$f_{\mathbf{a}}(\mathbf{x}) = \sum_{i=1}^{m} x_i \cdot a_i,$$

for all $\mathbf{a} \in R^m$ and $\mathbf{x} \in D^m$, where $\sum_i x_i \cdot a_i$ is computed using the ring addition and multiplication operations.

Besides one-wayness, Micciancio showed that for a special case of the above function family, the distribution of $f_{\mathbf{a}}(\mathbf{x})$ is uniform and independent from $\mathbf{a}$.

Theorem 1 ([21]). *For any finite field $\mathbb{F}$, subset $S \subset \mathbb{F}$, and integers n, m, the hash function family $\mathcal{K}(\mathbb{F}^n, S^n, m)$ is ϵ-regular for*

$$\epsilon = \frac{1}{2}\sqrt{(1 + |\mathbb{F}|/|S|^m)^n - 1}.$$

In particular, for any $q = n^{O(1)}$, $|S| = n^{\Omega(1)}$ and $m = \omega(1)$, the function ensemble $\mathcal{K}(\mathbb{F}_q^n, S^n, m)$ is almost regular (i.e., ϵ is negligible).

Here, "ϵ-regular" means that the statistical distance between uniform distribution $U((\mathbb{F}^n)^m, \mathbb{F}^n)$ and $\{(\mathbf{a}, f_{\mathbf{a}}(\mathbf{x})) : \mathbf{a} \leftarrow U((\mathbb{F}^n)^m), \mathbf{x} \leftarrow U((S^n)^m)\}$ is at most ϵ. $\mathbb{F}^n$ is a ring of vectors with convolution product.

Sometimes, one-wayness is not sufficient enough for the design of a cryptographic protocol. Lyubashevsky and Micciancio proved that finding a collision in some instance of the generalized knapsack function family is as hard as solving the worst-case problem in a certain lattice [19].

Definition 3 (Collision Problem). *For any generalized knapsack function family $\mathcal{K}(R, D, m)$, define the collision problem $Col_{\mathcal{K}}(h_{\mathbf{a}})$ as follows: given a function $h_{\mathbf{a}} \in \mathcal{K}$, find $\mathbf{b}, \mathbf{c} \in D^m$ such that $\mathbf{b} \neq \mathbf{c}$ and $h_{\mathbf{a}}(\mathbf{b}) = h_{\mathbf{a}}(\mathbf{c})$.*

If there is no polynomial time algorithm that can solve $\mathrm{Col}_{\mathcal{K}}$ with non-negligible probability when given a function $h_{\mathbf{a}}$ which is distributed uniformly at random in $\mathcal{K}$, then $\mathcal{K}$ is collision resistant family of hash functions.

The expansion factor is a parameter proposed to quantify the quality of modulus f in the ideal lattice [19]. The expansion factor of f is defined as

$$\mathrm{EF}(f, k) = \max_{g \in \mathbb{Z}[x], \deg(g) \leq k(\deg(f)-1)} \|g\|_f / \|g\|_\infty$$

where $\|g\|_f$ is short for $\|g \bmod f\|_\infty$. Moreover, $\mathrm{EF}(x^n + 1, k) \leq k$.

The generalized knapsack function family $\mathcal{K}(R, D, m)$ considered in [19] is instantiated as follows. Let $R = \mathbb{Z}_q[x]/\langle f \rangle$ be a ring for some integer q, where $f \in \mathbb{Z}[x]$ is a monic, irreducible polynomial of degree n with expansion factor $\mathrm{EF}(f, 3) \leq \varepsilon$. Let $D = \{g \in R : \|g\| \leq \beta\}$ for some positive integer β.

Theorem 2 ([19]). *Let $\mathcal{K}(R, D, m)$ be a generalized knapsack function family as above with $m \geq \frac{\log q}{\log 2\beta}$ and $q > 2\varepsilon\beta mn^{1.5} \log n$. Then, for $\gamma = 8\varepsilon^2\beta mn \log^2 n$, there is a polynomial time reduction from f-$SPP_\gamma(\mathcal{I})$ for any ideal $\mathcal{I} \in R$ to $Col_{\mathcal{K}}(h)$ where h is chosen uniformly at random from $\mathcal{K}$.*

If we denote by $\mathcal{I}(f)$ the set of lattices that are isomorphic (as additive groups) to ideals of $\mathbb{Z}[x]/\langle f\rangle$, then there is a straightforward reduction from $\mathcal{I}(f)$-SVP$_\gamma$ to f-SPP$_\gamma$, and the vise versa. It is conjectured that approximating $\mathcal{I}(f)$-SVP$_\gamma$ to within a polynomial factor is a hard problem, although it is not NP-hard [1,11].

2.3 The Public-Key Encryption on Ideal Lattices

The encryption scheme we described here was proposed by Stehlé *et al.* [34]. The ideal-lattice-based encryption scheme is formalized as a collection of efficient procedures $\mathcal{ES} = (\mathbf{Setup}, \mathbf{KGen}, \mathbf{Enc}, \mathbf{Dec})$.

Setup(1^n): n is the security parameter. Fix $f(X) = X^n + 1 \in \mathbb{Z}[X]$ and $q = \text{poly}(n)$ a prime satisfying $q \equiv 3 \bmod 8$. Set $\sigma = 1$, $r = 1 + \log_3 q$, and $m = (\lceil \log q \rceil + 1)\sigma + r$. Let $R = \mathbb{Z}_q[X]/\langle f\rangle$. All the parameters generated in this procedure are published as the global parameter pp.

KGen(pp): On input global parameter pp, it runs the trapdoor generation algorithm **Id-Trap** to get a trapdoor function $h_{\mathbf{g}} : \mathbb{Z}_q^n \times \mathbb{Z}_q^{mn} \to \mathbb{Z}_q^{mn}$ and a trapdoor S, where $\mathbf{g}$ is the function index. The first component of the domain of $h_{\mathbf{g}}$ can be viewed as a subset of $\mathbb{Z}_2^{\ell_I}$ for $\ell_I = O(n \log q)$. Generate $\mathbf{r} \in \mathbb{Z}_2^{\ell_I + \ell_\mu}$ uniformly and define the Toeplitz matrix $M_{\text{GL}} \in \mathbb{Z}_2^{\ell_\mu \times \ell_i}$ whose ith row is $(r_i, \ldots, r_{\ell_I + i - 1})$. It outputs the public key $epk = (\mathbf{g}, \mathbf{r})$ and the secret key $esk = S$.

Enc(pp, epk, μ): Given ℓ_μ bit message μ with $\ell_\mu = n/\log n$ and public key $epk = (\mathbf{g}, \mathbf{r})$, sample $(\mathbf{s}, \mathbf{e})$ with $\mathbf{s} \in \mathbb{Z}_q^n$ uniform and $\mathbf{e}$ sampled from $\bar{\psi}_{\alpha q}$, where $\bar{\psi}_{\alpha q}$ is the reduction modulo q of the standard Gaussian distribution with parameter αq. It then evaluates $C_1 = h_{\mathbf{g}}(\mathbf{s}, \mathbf{e})$ and computes $C_2 = \mu \oplus (M_{\text{GL}} \cdot \mathbf{s})$, where the product $M_{\text{GL}} \cdot \mathbf{s}$ is computed over $\mathbb{Z}_2$, and $\mathbf{s}$ is viewed as a string over $\mathbb{Z}_2^{\ell_I}$. Return the ciphertext $C = (C_1, C_2)$.

Dec(pp, esk, C): Given cyphertext $C = (C_1, C_2)$ and secret key $esk = (S, \mathbf{r})$, invert C_1 to compute $(\mathbf{s}, \mathbf{e})$ such that $h_{\mathbf{g}}(\mathbf{s}, \mathbf{e}) = C_1$, and return message $\mu = C_2 \oplus (M_{\text{GL}} \cdot \mathbf{s})$.

To see the details of the trapdoor generation algorithm **Id-Trap** and the one-way trapdoor function family $\{h_{\mathbf{g}} : \mathbb{Z}_q^n \times \mathbb{Z}_q^{mn} \to \mathbb{Z}_q^{mn}\}_{g \in (\mathbb{Z}_q[x]/\langle f\rangle)^m}$, we refer to the literature [34] in which Stehlé *et al.* also proved that the above encryption scheme is IND-CPA secure if the Ideal-LWE$^f_{m,q;\Psi_{\alpha q}}$ problem is hard.

The notion of key privacy is formally defined by Bellare *et al.* [4]. It requires that the receiver of a ciphertext is anonymous from the point of view of the adversary. Fortunately, we can deduce from the observation 1 of [13] that the aforementioned encryption scheme $\mathcal{ES}$ is of key privacy.

3 Anonymous Cryptographic Currency Model Based on Linkable Ring Signatures

Cryptocash system based on linkable ring signatures emerged after researchers found that Bitcoin was not fully anonymous and untraceable. CryptoNote and

Monero are two typical instances. We describe here the properties of an anonymous cryptocash system and state the techniques [31] to construct such a system.

In a cryptocash system, there are three parties: a sender, who owns a coin and decides to spend it, a receiver, who is the destination that a coin is delivered to, and a public ledger where all transactions are recorded. An anonymous cryptocash system should satisfy the following properties:

- **Untraceability:** If Tx is a transaction from sender A to receiver B, and Tx has been recorded in the public ledger, no one else can determine the sender with probability significantly larger than $1/N$ by accessing the transcript of Tx, where N is the number of possible senders in a related input of the Tx. Moreover, even receiver B cannot prove that A is the true sender of Tx.
- **Unlinkability:** If Tx_1 is a transaction from sender A to receiver C, Tx_2 is another transaction from sender B to receiver C, and Tx_1, Tx_2 have been recorded in the public ledger, then for any subsequent transactions in the public ledger, no one else can use them to link the outputs of the two transactions to a single user, even for senders A and B.
- **Detecting Double Spending:** If Tx_1 is a transaction which describes that coin c has been sent from sender A to receiver B, and Tx_1 has been recorded in the public ledger, every user of the system could detect another transaction Tx_2 that describes the same coin c. Furthermore, Tx_2 will never be accepted and recorded in the public ledger.

To design a cryptocash protocol which provides all the above properties, the CryptoNote and Monero suggested to adopt the modification of the traceable ring signature [9], which generates a one-time signature on behalf of a temporal group. Since it is a one-time signature with an explicit identification tag about the signing key, it could prevent a coin being double-spent. Besides, since it is a ring signature where the identity of the real signer is hidden within a set of possible signers, it guarantees untraceability. In addition, ring signature supports unlinkability since the inputs in a transaction may be brought from outputs of transactions belonging to other users.

To employ a linkable ring signature in a cryptocash system, the receiver should produce a one-time key pair for each transaction. A sender could obtain the public key of the receiver for the transaction and build a transaction with an output script containing that key's information. The drawback of this trivial method is that a receiver has to maintain a lot of one-time keys. Furthermore, a sender has to contact each receiver for their fresh one-time public key when the sender builds a transaction. Alternatively, CryptoNote suggests another method which enables a receiver to store only a single key pair. A sender could produce a random value to generate a one-time public key for the receiver based on this single public key. The one-time public key is referred to as the destination address (destination key). This is a convenient design at the cost of a slightly weakened unlinkability. Specifically, if a user has a single key, a sender could always identify a receiver from the sender's transaction by its random value of the transaction. If two senders collude, and they have sent coins to the same receiver, they could identify the same receiver while the trivial method avoids

this. And if a later transaction includes the two senders' outputs at the same time, with a higher probability, the later transaction is made by the receiver. Note that a receiver could still produce another key pair at will as in the Bitcoin system to avoid the small problem.

Finally, let us observe a standard transaction in a linkable ring signature based cryptocash system. In such a system, the value of a coin is bound with a destination address. Suppose A and B are two users in the system. B has a single key pair (pk_B, sk_B). A has the private key sk_1 of a destination address vk_1, which represents a coin, say c, which has been sent to A previously. If A decides to send c to B, he generates a destination address vk_2 and an auxiliary input aux for B; he then chooses a number of transactions from the public ledger such that the delivered value of coin is equivalent to c; he extracts the destination addresses of those transactions and assembles them with vk_1 to form a ring L; he runs a ring signature algorithm to sign transaction Tx, which involves information about (c, aux, vk_2, L), with signing key sk_1 and broadcasts the transaction; If the signature generated by sk_1 is not linkable to any transaction on the ledger, the public ledger will accept this transaction and record it; B uses its private key sk_B to check every passing transaction to determine if transaction Tx is for B and recovers the signing key sk_2 corresponding to vk_2. With sk_2, user B can spend c by signing another transaction. However, even A does not know when and where B spends it due to the functionality of the linkable ring signature.

It is obvious that linkable ring signature is vital for an anonymous cryptocash system. We next detail the ideal-lattice-based version of a linkable ring signature.

4 Linkable Ring Signature Based on Ideal-Lattices

The strong similarity in the construction between a lattice-based signature and DLP-based one (see Lyubashevsky's signature [18] and the Schnorr's signature [32,33]) implies that the latter can help us to design the former, *e.g.*, using the work in [17] or [18], we can easily obtain a linkable ring signature based on lattices with signature size of $O(N)$, where N is the number of participants of the ring. However, such a construction is not efficient enough for a practical cryptocash system. In this section, we will present an ideal-lattice-based linkable ring signature of size $O(\log N)$ using the idea in [12]. We start this section with a brief recall on their work.

4.1 A Brief Recall

In [12], Groth and Kohlweiss proposed an efficient Sigma-protocol, which can be used as an ad-hoc group identification scheme. Their ring signature scheme is a direct transformation of the identification scheme with the Fiat-Shamir heuristic [8]. As the transmission of the identification scheme involves only $O(\log N)$ commitments, the resulting ring signature scheme is of size $O(\log N)$.

Their work starts from homomorphic commitments scheme such as the Pedersen commitment scheme (*i.e.*, $\mathrm{com}(m; r) = h^m g^r$). The first step is to design a

Sigma-protocol Σ_1 to prove in zero-knowledge that such a commitment is opened to 0 or 1. Once the subroutine Σ_1 is established, to design a ad-hoc group identification scheme is to construct a Sigma-protocol Σ_2 to show in zero-knowledge that one of N commitments is opened to 0. Here, a commitment to 0 is the public key of a user and the randomness is the corresponding secret key. If the ℓth user of the ad-hoc group $\{\text{user}_0, \cdots, \text{user}_{N-1}\}$ wants to identify himself secretly, Σ_2 first commits the integer ℓ bit by bit and runs Σ_1 to prove in zero-knowledge that those $\log N$ commitments are opened to 0 or 1. Then Σ_2 proves in zero-knowledge that the ℓth user can open the ℓth public key (*i.e.*, a commitment to 0) to 0, with the help of the intermediate parameters used in the foregoing Σ_1's. By replacing the challenge message with the hash value of all initial messages in Σ_2, we obtain a non-interactive zero-knowledge proof system which can be regarded as a ring signature. For the details of the generic construction of Σ_1 and Σ_2, we refer readers to the literature [12].

It is worth mentioning that the underlying homomorphic commitment is the corner stone for both the construction and the security proof of the foregoing ring signature. As a counterpart of their work, our scheme also contains an ideal-lattice-based commitment scheme (*i.e.*, $\text{com}(\mathbf{S}; \mathbf{X}) = \mathbf{HS} + \mathbf{GX}$). The details of the commitment scheme is left to Sect. 4.3.

4.2 Our Construction

To construct an $O(\log N)$ ring signature, Groth and Kohlweiss proposed a technique to compute the coefficients of a polynomial in the indeterminate x over finite field $\mathbb{Z}_q$ in advance, where x is a hash value computed later [12]. We extend their method to handle the polynomial with coefficients belonging to a ring of square matrices. The major difference is that the multiplication of matrices is not commutative. This is the reason why we restrict x in our scheme to be a 1×1 matrix. Since the scalar multiplication is commutative, we have the following result.

Let integer ℓ be in the interval $[0, N-1]$, and $M = \lceil \log N \rceil$. Given matrices $\mathbf{B}_j$, set $\mathbf{W}_j = \ell_j x\mathbf{I} + \mathbf{B}_j$. Let $\mathbf{W}_{j,1} = \mathbf{W}_j = \ell_j x\mathbf{I} + \mathbf{B}_j = \delta_{1\ell_j} x\mathbf{I} + \mathbf{B}_j$ and $\mathbf{W}_{j,0} = x\mathbf{I} - \mathbf{W}_j = (1-\ell_j)x\mathbf{I} - \mathbf{B}_j = \delta_{0\ell_j} x\mathbf{I} - \mathbf{B}_j$. Then for each $i \in [0, N-1]$, the product $\prod_{j=1}^{M} \mathbf{W}_{j,i_j}$ is a polynomial in x of the form

$$P_i(x) = \prod_{j=1}^{M} (\delta_{i_j \ell_j} x\mathbf{I}) + \sum_{k=0}^{M-1} \mathbf{P}_{i,k} x^k = \delta_{i\ell} x^M \mathbf{I} + \sum_{k=0}^{M-1} \mathbf{P}_{i,k} x^k, \tag{1}$$

where $\mathbf{P}_{i,k}$ is the coefficient of the kth degree term, and can be efficiently computed if $\{\mathbf{B}_j\}_{j=1}^{M}$, i and ℓ are given.

The linkable ring signature scheme consists of a tuple of efficient procedures $\mathcal{LRS} = (\mathbf{Setup}, \mathbf{KGen}, \mathbf{Sign}, \mathbf{Vfy}, \mathbf{Link})$. Let N be the maximum size of the ring, $M = \lceil \log N \rceil$, and n be a power of 2. The details of those procedures are shown as follows:

Setup$(1^n, N)$: On input N and security parameter n, the procedure initiates a hash function introduced in [18] as a random oracle $\mathcal{H} : \{0,1\}^* \to \{v : v \in \{-1,0,1\}^n, \|v\|_1 \leq p\}$, such that $2^p \cdot \binom{n}{p} \geq 2^{100}$. It sets $\varepsilon = 3$, $t = n^{\Omega(1)}$, $\beta \geq \max\{\frac{t(p^{M+1}-1)}{(p-1)}, 2mnt^2\}$, $m = \omega(1)$. Pick a prime q such that $(2\beta)^m > q > 2\varepsilon\beta mn^{1.5}\log n$. All operations in this system are done in $R = \mathbb{Z}_q[X]/\langle f\rangle$, for $f = X^n + 1$. Let $Q = \{g \in R : \|g\| \leq t\}$ and $\tilde{Q} = \{g \in R : \|g\| \leq t-1\}$. Relying on those parameters, this procedure samples matrices $\mathbf{G}, \mathbf{H} \in R^{1\times m}$ uniformly at random. Finally it outputs $pp = (n, m, \mathbf{G}, \mathbf{H}, \mathcal{H}, q, t, N)$ as the global parameters.

KGen(pp): For the ith user, this procedure randomly chooses $\mathbf{X}_i \leftarrow Q^{m\times m}$ and computes $\mathbf{Y}_i = \mathbf{G}\mathbf{X}_i$. The ith user's verifying key is $vk_i = \mathbf{Y}_i$ and the singing key is $sk_i = \mathbf{X}_i$.

Sign(pp, sk_ℓ, μ, L): Without loss of generality, let $L = (\mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_{N-1})$ be the ensemble of a ring with the largest size. On input a message μ, the ℓth user's signature on behalf of L is generated as follows

- Compute $\mathbf{R}_\ell = \mathbf{H}\mathbf{X}_\ell$.
- For j from 1 to M
  - sample $\mathbf{K}_j, \mathbf{C}_j, \mathbf{D}_j \leftarrow Q^{m\times m}$,
  - if $\ell_j = 0$, randomly pick $\mathbf{B}_j \leftarrow Q^{m\times m}$,
    else if $\ell_j = 1$, randomly draw $\mathbf{B}_j \leftarrow \tilde{Q}^{m\times m}$,
  - compute $\mathbf{V}_{\ell_j} = \mathbf{H}(\ell_j\mathbf{I}) + \mathbf{G}\mathbf{K}_j$,
  - compute $\mathbf{V}_{a_j} = \mathbf{H}\mathbf{B}_j + \mathbf{G}\mathbf{C}_j$,
  - compute $\mathbf{V}_{b_j} = \mathbf{H}(\ell_j\mathbf{B}_j) + \mathbf{G}\mathbf{D}_j$.
- For k from 0 to $M-1$
  - sample $\mathbf{E}_k \leftarrow Q^{m\times m}$
  - compute $\mathbf{V}_{d_k} = (\sum_{i=0}^{N-1} \mathbf{Y}_i\mathbf{P}_{i,k}) + \mathbf{G}\mathbf{E}_k$, where $\mathbf{P}_{i,k}$ is introduced in (1),
  - compute $\mathbf{V}'_{d_k} = \mathbf{H}\mathbf{E}_k$.
- Let $S_1 = \{\mathbf{V}_{\ell_j}, \mathbf{V}_{a_j}, \mathbf{V}_{b_j}, \mathbf{V}_{d_{j-1}}, \mathbf{V}'_{d_{j-1}}\}_{j=1}^M$ and then compute hash value $x = \mathcal{H}(pp, \mu, L, S_1, \mathbf{R}_\ell)$.
- For j from 1 to M, compute
  1. $\mathbf{W}_j = \ell_j x\mathbf{I} + \mathbf{B}_j$,
  2. $\mathbf{Z}_{a_j} = \mathbf{K}_j(x\mathbf{I}) + \mathbf{C}_j$,
  3. $\mathbf{Z}_{b_j} = \mathbf{K}_j(x\mathbf{I} - \mathbf{W}_j) + \mathbf{D}_j$.
- Compute $\mathbf{Z}_d = \mathbf{X}_\ell(x^M\mathbf{I}) - \sum_{k=0}^{M-1} \mathbf{E}_k x^k$.
- Let $S_2 = \{\mathbf{W}_j, \mathbf{Z}_{a_j}, \mathbf{Z}_{b_j}\}_{j=1}^M$. Publish the signature $\sigma = \{S_1, S_2, \mathbf{Z}_d, \mathbf{R}_\ell, L\}$.

Vfy(pp, μ, σ): On input signature σ and message μ, this procedure does as follows to test the validity of σ.

1. Compute hash value $x = \mathcal{H}(pp, \mu, L, S_1, \mathbf{R}_\ell)$.
2. For j from 1 to M, consider the following inequalities
   - $\|\mathbf{W}_j\| \leq t$,
   - $\|\mathbf{Z}_{a_j}\| \leq (p+1)t$,
   - $\|\mathbf{Z}_{b_j}\| \leq tp + t^2nm + t$,

- $\|\mathbf{Z}_d\| \leq \frac{t(p^{M+1}-1)}{p-1}$.

If any of them does not hold, output 0 and abort.

3. For j from 1 to M, consider the following equations
   - $\mathbf{V}_{\ell_j}(x\mathbf{I}) + \mathbf{V}_{a_j} = \mathbf{H}\mathbf{W}_j + \mathbf{G}\mathbf{Z}_{a_j}$,
   - $\mathbf{V}_{\ell_j}(x\mathbf{I} - \mathbf{W}_j) + \mathbf{V}_{b_j} = \mathbf{G}\mathbf{Z}_{b_j}$.

   If any of the aforementioned equations does not hold, output 0 and abort.
4. If the equation $\mathbf{R}_\ell(x^M\mathbf{I}) + \sum_{k=0}^{M-1} \mathbf{V}'_{d_k}(-x^k) = \mathbf{H}\mathbf{Z}_d$ does not hold, output 0 and abort.
5. Inspect whether

$$\sum_{i=0}^{N-1}(\mathbf{Y}_i \prod_{j=1}^{M} \mathbf{W}_{j,i_j}) + \sum_{k=0}^{M-1} \mathbf{V}_{d_k}(-x^k) = \mathbf{G}\mathbf{Z}_d$$

   satisfies. If not, output 0; otherwise output 1 (accept).

Link(pp, σ_1, σ_2): For two signatures $\sigma_1 = (\ldots, \mathbf{R}_1, L_1)$ and $\sigma_2 = (\ldots, \mathbf{R}_2, L_2)$, if $\mathbf{R}_1 = \mathbf{R}_2$, return 1 (linked) for concluding that they are generated by the same signer; otherwise, return 0 (unlinked).

Correctness: To see that the signature generated by the **Sign** procedure always passes the **Vfy** procedure, we first observe the four equations in the **Vfy** procedure. The equations in step 3 are to prove in zero-knowledge that the signer is the ℓth user (some $\ell \in [0, N-1]$). The correctness of those equations is shown directly through a simple deduction. The equation in step 4 is to prove that the parameter for linking is correct. For a valid signature, it holds since

$$\begin{aligned} & \mathbf{R}_\ell(x^M\mathbf{I}) + \textstyle\sum_{k=0}^{M-1} \mathbf{V}'_{d_k}(-x^k) \\ = \; & \mathbf{H}\mathbf{X}_\ell(x^M\mathbf{I}) + \textstyle\sum_{k=0}^{M-1} \mathbf{H}\mathbf{E}_k(-x^k) \\ = \; & \mathbf{H}(\mathbf{X}_\ell(x^M\mathbf{I}) - \textstyle\sum_{k=0}^{M-1} \mathbf{E}_k x^k) \\ = \; & \mathbf{H}\mathbf{Z}_d \end{aligned} .$$

The equation in step 5 is to prove in zero-knowledge that the anonymous signer holds the secret key of the ℓth user. To see the correctness of the last equation, observe that $P_i(x) = \prod_{j=1}^{M} \mathbf{W}_{j,i_j}$ introduced in (1) is a polynomial in x of degree M, only if $i = \ell$. With this fact in mind, we have

$$\begin{aligned}
&\textstyle\sum_{i=0}^{N-1}(\mathbf{Y}_i\prod_{j=1}^{M}\mathbf{W}_{j,i_j})+\sum_{k=0}^{M-1}\mathbf{V}_{d_k}(-x^k)\\
&=\textstyle\sum_{i=0}^{N-1}\mathbf{Y}_i(\delta_{i\ell}x^M\mathbf{I}+\sum_{k=0}^{M-1}\mathbf{P}_{i,k}x^k)+\sum_{k=0}^{M-1}((\sum_{i=0}^{N-1}\mathbf{Y}_i\mathbf{P}_{i,k})+\mathbf{G}\mathbf{E}_k)(-x^k)\\
&=\textstyle\sum_{i=0}^{N-1}\sum_{k=0}^{M-1}(\mathbf{Y}_i\mathbf{P}_{i,k}x^k-\mathbf{Y}_i\mathbf{P}_{i,k}x^k)+\mathbf{Y}_\ell(\delta_{\ell\ell}x^M\mathbf{I})+\sum_{k=0}^{M-1}\mathbf{G}\mathbf{E}_k(-x^k)\\
&=\mathbf{G}(\mathbf{X}_\ell(x^M\mathbf{I})-\textstyle\sum_{k=0}^{M-1}\mathbf{E}_kx^k)\\
&=\mathbf{G}\mathbf{Z}_d
\end{aligned}$$

It remains to show that $\{\mathbf{W}_j,\mathbf{Z}_{a_j},\mathbf{Z}_{b_j}\}_{j=1}^{M}$, and $\mathbf{Z}_d$ are short enough to pass step 2 of the **Vfy** procedure.

Note that for polynomials $a,b\in R$, the norm of their product is bounded by $\|a\|\cdot\|b\|\cdot n$. For $a\in R$ and $b\in\{v:v\in\{-1,0,1\}^n,\|v\|_1\le p\}$ the norm of $a\cdot b$ is not larger than $\|a\|\cdot\|b\|\cdot p$. Depending on the above two facts and the triangle inequality, the correctness of the inequalities in step 2 can be validated easily. For example, $\|\mathbf{Z}_{b_j}\|=\|\mathbf{K}_j(x\mathbf{I}-\mathbf{W}_j)+\mathbf{D}_j\|\le\|\mathbf{K}_jx\mathbf{I}\|+\|\mathbf{K}_j(-\mathbf{W}_j)\|+\|\mathbf{D}_j\|\le tp+t^2nm+t$ and $\|\mathbf{Z}_d\|\le\|\mathbf{X}_\ell(x^M\mathbf{I})\|+\|\sum_{k=0}^{M-1}\mathbf{E}_kx^p\|\le tp^M+\|\mathbf{E}_0x^0\|+\|\mathbf{E}_1x^1\|+\cdots+\|\mathbf{E}_{M-1}x^{M-1}\|\le tp^M+t+tp+\cdots+tp^{M-1}=\frac{t(p^{M+1}-1)}{p-1}$.

Even though the foregoing linkable ring signature is designed over ideal lattices, a classic edition of this signature can be built by instead using any cyclic group as long as its underlying DLP is hard. We propose a linkable ring signature based on the ECDLP, and discuss how to implement this signature with ECC in the full version of this paper [38]. It is easy to see that the RingCT [26] (later strengthened by Sun *et al.* [35]), which is adopted in Monero to hide the amount of a transaction, is trivially achievable with the ECDLP-based signature.

4.3 Security Proof

Groth and Kohlweiss have proved that the generic construction of their ring signature is secure in the random oracle model, if its underlying commitment scheme is perfectly hiding and computationally binding [12]. Since our $\mathcal{LRS}$ is designed over the framework of their generic construction, in order to prove the security of $\mathcal{LRS}$, it is sufficient enough to prove the binding and hiding properties of the commitment scheme applied in $\mathcal{LRS}$.

Theorem 3 ([12]). *The generic construction of the ring signature scheme in [12] is perfect anonymity if the underlying commitment scheme is perfectly hiding. It is unforgeable in the random oracle model if the commitment scheme is perfectly hiding and computationally binding.*

A non-interactive commitment scheme allows a sender to construct a commitment to a value. The sender may later open the commitment and reveal the value so that the receiver can verify the opening and check if it is the value that was committed at the beginning. A commitment scheme is said to be hiding, only

if it reveals nothing about the committed value. The binding property ensures that a sender cannot open the commitment to two different values.

The non-interactive commitment scheme adopted in our $\mathcal{LRS}$ consists of a pair of efficient algorithms $\mathcal{CMT} = (\mathbf{Gen, Com})$.

Gen(1^n)**:** As the commitment scheme is a subroutine of $\mathcal{LRS}$, the setup algorithm runs $\mathcal{LRS}.\mathbf{Setup}(1^n)$ to obtain $\mathcal{LRS}.pp$ and picks $pp = (n, m, \mathbf{G}, \mathbf{H}, q, t)$ out of $\mathcal{LRS}.pp$ to be the global parameters of the commitment scheme. Values and randomness are elements in $Q^{m\times m}$. All operations are done in R.

Com$(pp, \mathbf{S})$**:** On input a value $\mathbf{S} \in Q^{m\times m}$, this algorithm samples $\mathbf{X} \leftarrow Q^{m\times m}$ uniformly at random, and generates the commitment by computing $\mathbf{C} = \mathbf{HS} + \mathbf{GX}$. $\mathbf{C}$ can later be opened by unveiling the short $\mathbf{S}$ and $\mathbf{X}$.

The correctness of the foregoing commitment scheme $\mathcal{CMT}$ is obvious. It remains to prove that $\mathcal{CMT}$ is hiding and biding.

Theorem 4 (Binding and Hiding). *For any committed value* $\mathbf{S} \in Q^{m\times m}$ *and uniformly chosen randomness* $\mathbf{X} \leftarrow Q^{m\times m}$*, the commitment* $\mathbf{C} = \mathbf{HS} + \mathbf{GX}$ *reveals nothing about* $\mathbf{S}$*. Moreover, the sender cannot open* $\mathbf{C}$ *to* $\mathbf{S}' \neq \mathbf{S}$*, if the collision problem* $Col_{\mathcal{K}}(h_{\mathbf{H}})$ *with respect to the generalized knapsack function family* $\mathcal{K}(R, D, m)$ *is intractable to solve, where* $D = \{g \in R : \|g\| \leq \beta\}$*.*

Proof. Given a matrix $\mathbf{G} \in R^{1\times m}$ sampled uniformly at random, we obtain a uniformly random instance $f_{\mathbf{G}} : Q^m \rightarrow R$ from the generalized knapsack function family $\mathcal{F}(R, Q, m)$. Let $\mathbf{x}_i$ symbolize the ith column of the matrix $\mathbf{X} \in Q^{m\times m}$, so that it is a vector sampled from $Q^{m\times 1}$ uniformly at random. Note that R can be regarded as $\mathbb{Z}_q^n$ and $\mathbb{Z}_q$ is a finite field. Additionally, we have $q > 2\varepsilon\beta mn^{1.5}\log n$, $t = n^{\Omega(1)}$, $m = \omega(1)$ in our setting. Therefore, according to Theorem 1, the distribution of $f_{\mathbf{G}}(\mathbf{x}_i) = \mathbf{Gx}_i$ is almost uniform over $\mathbb{Z}_q^n$ (namely R), and $f_{\mathbf{G}}(\mathbf{X}) = \mathbf{GX}$ is almost uniform over $R^{1\times m}$. The same result is also suitable for $f_{\mathbf{H}}(\mathbf{S}) = \mathbf{HS}$. Then, for any $\mathbf{S} \in Q^{m\times m}$, $\mathbf{C} = \mathbf{HS} + \mathbf{GX}$ is almost uniformly distributed over $R^{1\times m}$ and reveals nothing about the $\mathbf{S}$.

We proceed to prove the binding property. From our parameter settings $(2\beta)^m \geq q$, we have $m > \frac{\log q}{\log 2\beta}$. Depending on Theorem 2, to solve the collision problem $\mathrm{Col}_{\mathcal{K}}(h_{\mathbf{H}})$ with respect to the generalized knapsack function family $\mathcal{K}(R, D, m)$ is as hard as to solve the $\mathcal{I}(f)$-SVP$_\gamma$ problem, where $\gamma = 8\varepsilon^2\beta mn\log^2 n$ is a polynomial in security parameter n. Suppose for the sake of contradiction that a PPT adversary $\mathcal{A}$ can break the binding property of $\mathcal{CMT}$. We will design an algorithm $\mathcal{B}$ to solve $\mathrm{Col}_{\mathcal{K}}(h_{\mathbf{H}})$ with respect to $\mathcal{K}(R, D, m)$.

After $\mathcal{B}$ receiving an instance $h_{\mathbf{H}}$ labeled by $\mathbf{H} \in R^{1\times m}$ from $\mathcal{K}(R, D, m)$, it selects $\mathbf{T} \leftarrow Q^{m\times m}$ uniformly at random, and computes $\mathbf{G} = \mathbf{HT}$. Relying on the discussion in the proof of hiding property, $\mathbf{G}$ is uniformly distributed in $R^{1\times m}$. Subsequently, $\mathcal{B}$ simulates a commitment scheme for $\mathcal{A}$ by publishing $pp = (n, m, \mathbf{G}, \mathbf{H}, q, t)$ as the global parameters. Note that the distributions of $\mathbf{G}$ and $\mathbf{H}$ are the same as that in the original scheme. Consequently, by the hypothesis, $\mathcal{A}$ could return $\mathbf{S}, \mathbf{S}', \mathbf{X}, \mathbf{X}' \in Q^{m\times m}$, such that $\mathbf{S} \neq \mathbf{S}$ and

$\mathbf{HS} + \mathbf{GX} = \mathbf{HS}' + \mathbf{GX}'$ in a non-negligible probability. $\mathcal{B}$ considers the two possible cases.

Case 1: If $\mathbf{HS} = \mathbf{HS}'$, then $\mathbf{S}$ and $\mathbf{S}'$ are a pair of collisions with respect to $h_{\mathbf{H}}$, since $\|\mathbf{S}\| \leq t < \beta$, $\|\mathbf{S}'\| \leq t < \beta$.

Case 2: If $\mathbf{HS} \neq \mathbf{HS}'$, then $\mathbf{X} \neq \mathbf{X}'$ and we have $\mathbf{H}(\mathbf{S} - \mathbf{S}') = \mathbf{G}(\mathbf{X}' - \mathbf{X})$. By using $\mathbf{G} = \mathbf{HT}$, we can deduce that $\mathbf{H}(\mathbf{S} - \mathbf{S}') = \mathbf{HT}'$, where $\mathbf{T}' = \mathbf{T}(\mathbf{X}' - \mathbf{X})$. Since $\|\mathbf{T}'\| \leq 2nmt^2 < \beta$, $\|\mathbf{S} - \mathbf{S}'\| \leq \|\mathbf{S}\| + \|\mathbf{S}'\| \leq 2t < \beta$, $\mathbf{S} - \mathbf{S}'$ and $\mathbf{T}'$ are a pair of collisions with respect to $h_{\mathbf{H}}$.

Both **Case 1** and **Case 2** yield a contradiction to the assumption that $\mathrm{Col}_{\mathcal{K}}(h_{\mathbf{H}})$ with respect to the generalized knapsack function family $\mathcal{K}(R, D, m)$ is intractable to solve. Consequently, the commitment scheme is binding. □

Since our underlying commitment scheme $\mathcal{CMT}$ is binding and hiding, the anonymity and unforgeability of the linkable ring signature $\mathcal{LRS}$ can be shown according to Theorem 3. For a complete discussion of the security proof, we refer readers to the full version of this paper [38]. Actually, most of the techniques follows that of [12] and x has the unique multiplicative inverse in R.

The next is to prove that our linkable ring signature is linkable.

Theorem 5 (Linkability). *Our linkable ring signature $\mathcal{LRS}$ is linkable. Formally, given a key pair $(\mathbf{X}, \mathbf{Y} = \mathbf{GX})$, it is impossible to generate a valid signature $\sigma = \{S_1, S_2, \mathbf{Z}_d, \mathbf{R}, L\}$, such that $\mathbf{Y} \in L$ and $\mathbf{R} \neq \mathbf{HX}$.*

Proof. Assume that a user with $(\mathbf{X}, \mathbf{Y})$ generates a signature $\sigma = \{S_1, S_2, \mathbf{Z}_d, \mathbf{R}, L\}$ on behalf of L, such that $\mathbf{Y} \in L$ and $\mathbf{R} \neq \mathbf{HX}$. As σ is a valid signature, from step 4 of the **Vfy** procedure, we have

$$\mathbf{R}(x^M \mathbf{I}) + \sum_{k=0}^{M-1} \mathbf{V}'_{d_k}(-x^k) = \mathbf{HZ}_d \quad . \tag{2}$$

Additionally, since $\mathbf{Z}_d$ can pass step 5 of the **Vfy** procedure, it must be generated by using the signing key $\mathbf{X}$, *i.e.*, $\mathbf{Z}_d = \mathbf{X}(x^M \mathbf{I}) - \sum_{k=0}^{M-1} \mathbf{E}_k x^k$. Otherwise, it yields a contradiction to the unforgeability of the signature scheme. Subsequently, we can deduce from (2) that

$$(\mathbf{HX} - \mathbf{R})(x^M \mathbf{I}) + \sum_{k=0}^{M-1} (\mathbf{V}'_{d_k} - \mathbf{HE}_k)(x^k) = \mathbf{0} \quad . \tag{3}$$

If $\mathbf{R} \neq \mathbf{HX}$, the left side of (3) is a polynomial in x of degree M. Once $\mathbf{R}$, $\mathbf{V}'_{d_k}$, $\mathbf{X}$ and $\mathbf{E}_k$ are given, Eq. (3) has at most M solutions. However, x is obtained by computing the hash function $\mathcal{H}(pp, \mu, L, \{\mathbf{V}_{\ell_j}, \mathbf{V}_{a_j}, \mathbf{V}_{b_j}, \mathbf{V}_{d_{j-1}}, \mathbf{V}'_{d_{j-1}}\}_{j=1}^{M}, \mathbf{R})$, and involves $2^n \cdot \binom{n}{p} \geq 2^{100}$ possible values. Consequently, the probability that the hash value is the solution of (3) is negligible and the only sensible condition for (3) to be satisfied is $\mathbf{HX} = \mathbf{R}$ and $\mathbf{V}'_{d_k} = \mathbf{HE}_k$. □

5 APQC Based on Linkable Ring Signatures

In CryptoNote, the author suggested using stealth addresses to protect the privacy of receivers in all transactions. A stealth address is a one-time address (a verifying key which is also called a destination key) for a receiver to receive coins. It is generated by the sender of a transaction, and only the real receiver could determine the one-time address and recover the corresponding signing key.

In this section, we will introduce a key-generation protocol to handle stealth addresses. By combining this protocol and the linkable ring signature presented in the previous section, we describe the standard transaction of APQC in detail at last.

5.1 Key-Generation Protocol

The key-generation protocol is responsible for three purposes. Firstly, it generates public and private keys for a user that initially joins the cryptocash system. Secondly, if Alice wants to pay coins to Bob, this protocol generates a fresh one-time address for Bob by using the random values chosen by Alice and the public key of Bob. Note that the one-time address is essentially a verifying key of the linkable ring signature scheme. Thirdly, since Alice broadcasts the transaction labeled with the destination address, the key-generation protocol helps Bob to efficiently recognize this transaction and to recover the corresponding signing key.

This protocol is formalized as four efficient procedures $\mathcal{KG}$=(**Setup**, **UKeyGen**, **DKeyGen**, **DKeyRec**) which are short forms for setup, user keys generation, destination keys generation, and destination keys recovery, respectively.

Setup$(1^n, 1^\lambda)$: On input security parameter, this procedure generates global parameters pp for the whole cryptocash system which means this procedure also runs $\mathcal{LRS}$.**Setup**(1^n) and $\mathcal{ES}$.**Setup**(1^n) as subroutines so that the signature scheme and encryption scheme are accurately initiated (see Sects. 2.3 and 4.2 for details). Let $(n, m, \mathbf{G}, \mathbf{H}, \mathcal{H}, q, t, N)$ be the global parameters of the linkable ring signature, and $R = \mathbb{Z}[X]_q/\langle X^n + 1\rangle$. Besides that, it chooses a cryptographic hash function $hash : \{0,1\}^* \to \{0,1\}^\lambda$. Let $D = \{g \in R : \|g\| \leq t/2\}$.

UKeyGen(pp)**:** When a user wants to join the cryptocash system, he executes this procedure. This procedure first generates the keys for public key encryption scheme $(epk, esk) \leftarrow \mathcal{ES}$.**KGen**$(pp)$. It then generates a partial key pair for the linkable ring signature scheme $\mathbf{X} \leftarrow D^{m\times m}$, $\mathbf{Y} = \mathbf{GX}$. Note that the norm of the partial signing key $\mathbf{X}$ is a little smaller than the original one of the linkable ring signature. $(epk, \mathbf{Y})$ and $(esk, \mathbf{X})$ are the public and private keys held by the user.

DKeyGen$(pp, epk, \mathbf{Y})$**:** If Alice wants to send coins to Bob who holds keys $(epk, \mathbf{Y})$, $(esk, \mathbf{X})$, she runs the procedure with epk and $\mathbf{Y}$. This procedure samples $\mathbf{X}_p \leftarrow D^{m\times m}$ and generates the destination key $\mathbf{Y}_d = \mathbf{GX}_p + \mathbf{Y}$ for Bob. $\mathbf{X}_p$ is a part of the signing key with respect to the destination key $\mathbf{Y}_d$, but no one

except Bob can recover the integral signing key. This procedure proceeds to pick an AES secret key k uniformly at random. It then computes $c_1 = \mathcal{ES}.\mathbf{Enc}_{epk}(k)$ with the public key encryption and computes $c_2 = \mathbf{AES}_k(hash(epk)\|\mathbf{X}_p)$ with the AES algorithm. Finally, it outputs the destination key $\mathbf{Y}_d$, and the auxiliary information c_1, c_2. The process of **DkeyGen** procedure is depicted in Fig. 1.

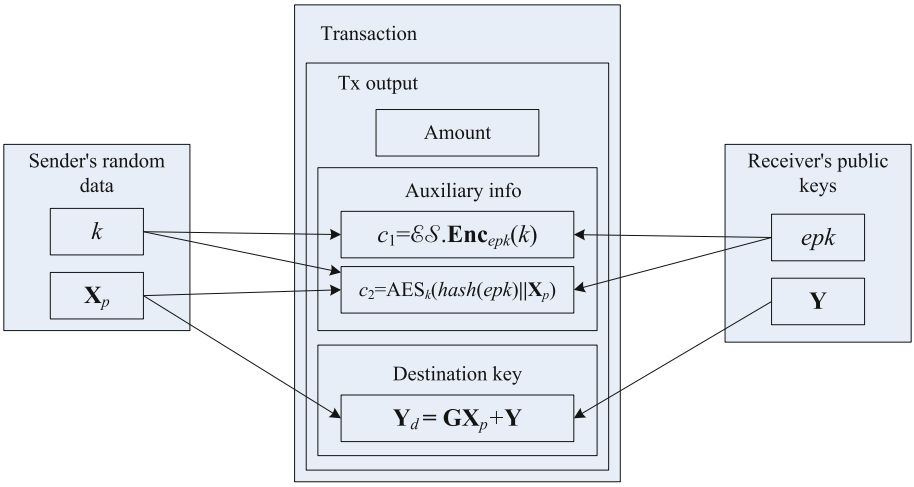


Fig. 1. DKeyGen procedure

DKeyRec$(pp, epk, esk, \mathbf{Y}, \mathbf{X}, (\mathbf{Y}_d, c_1, c_2))$**:** Bob runs this procedure to check $(\mathbf{Y}_d, c_1, c_2)$ of a passing transaction. If it is a transaction with Bob as recipient, it will be that (1) $k = \mathcal{ES}.\mathbf{Dec}_{esk}(c_1)$; (2) $(hash(epk)\|\mathbf{X}_p) = \mathbf{AES}_k(c_2)$. If this procedure finds that the first part of the plaintext of c_2 is not the hash value of Bob's public encryption key epk, then this procedure aborts and outputs 0. Otherwise, Bob computes $\mathbf{X}_d = \mathbf{X}_p + \mathbf{X}$ and $\mathbf{Y}'_d = \mathbf{GX}_d$. If $\mathbf{Y}'_d = \mathbf{Y}_d$, this procedure outputs 1 and admits the validity of the destination key $\mathbf{Y}_d$ and its signing key $\mathbf{X}_d$. Since $\|\mathbf{X}_d\| \leq \|\mathbf{X}_p\| + \|\mathbf{X}\| \leq t$, $\mathbf{X}_d$ is a valid signing key with correspondence to the destination key $\mathbf{Y}_d$. The process of this procedure is briefly shown in Fig. 2.

5.2 Transactions

We proceed to introduce transactions in APQC. Let Bob and Alice be two users of our APQC. Bob will runs $\mathcal{KG}.\mathbf{UKeyGen}$ to generates his public and private keys $(epk_{\text{Bob}}, \mathbf{Y}_{\text{Bob}})$, $(esk_{\text{Bob}}, \mathbf{X}_{\text{Bob}})$, when he initially joins the system. Similarly, $(epk_{\text{Alice}}, \mathbf{Y}_{\text{Alice}})$, $(esk_{\text{Alice}}, \mathbf{X}_{\text{Alice}})$ are the keys held by Alice. Besides the user keys, Alice and Bob maintain their own wallet addresses, respectively.

Assume that the destination address $\mathbf{Y}_{Bj}$ and its signing key $\mathbf{X}_{Bj}$ are in Alice's wallet, and she wants to send coins of this address to Bob.

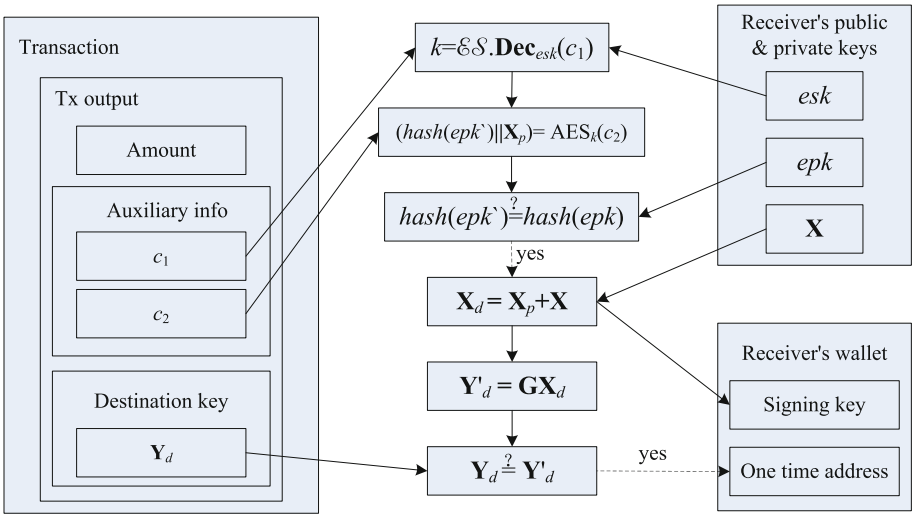


Fig. 2. DKeyRec procedure

Alice will specify $N-1$ foreign outputs (Output_{B1}, ..., $\text{Output}_{B(j-1)}$, $\text{Output}_{B(j+1)}$, ..., Output_{BN}) in which the amount is equivalent to that of Output_{Bj}. She proceeds to find Bob's public key $(epk_{\text{Bob}}, \mathbf{Y}_{\text{Bob}})$ and runs $\mathcal{KG}$.**DkeyGen**$(pp, epk_{\text{Bob}}, \mathbf{Y}_{Bob})$ to generate the destination key $\mathbf{Y}_{Cj}$ and its auxiliary information c_1, c_2 for Bob (see Fig. 1). She then pushes (1) Tx input including $\{\text{Output}_{Bi}\}_{i=1}^{N}$ and the amount she sends to Bob, (2) the destination key $\mathbf{Y}_{Cj}$ and auxiliary information c_1, c_2 she generated for Bob, (3) all previous transactions with output $\{\text{Output}_{Bi}\}_{i=1}^{N}$, into the hash function to obtain a hash digest, μ, of the transaction. Subsequently, she signs the hash digest by running $\sigma \leftarrow \mathcal{LRS}$.**Sign**$(pp, \mathbf{X}_{Bj}, \mu, \mathbf{Y}_{B1}, \ldots, \mathbf{Y}_{BN})$, where $\mathbf{Y}_{Bi}$ is the destination key of Output_{Bi}. Finally she broadcasts the transaction.

Bob checks all passing transactions. For each transaction, he extracts the destination key and auxiliary information $(\mathbf{Y}_d, c_1, c_2)$, and runs the procedure $\mathcal{KG}$.**DKeyRec**$(pp, epk_{Bob}, esk_{Bob}, \mathbf{Y}_{Bob}, \mathbf{X}_{Bob}, (\mathbf{Y}_d, c_1, c_2))$. If this transaction is the one that Alice sent to Bob, the foregoing procedure will return the signing key $\mathbf{X}_{Cj}$ for the destination key $\mathbf{Y}_d = \mathbf{Y}_{Cj}$. If this happens, Bob accepts this transaction and records $\mathbf{X}_{Cj}$, $\mathbf{Y}_d$ into his wallet. Bob can later spend the coin stored in the destination address $\mathbf{Y}_d$ because he has the signing key $\mathbf{X}_{Cj}$.

The standard transaction is also briefly depicted in the full version [38].

6 Conclusions and Future Works

While a lot of lattice-based ring signature and standard signature have been designed recently, linkable ring signature over lattices has not been to the best of our knowledge. The strong similarity in the construction between a lattice-based

signature and DLP-based one, *e.g.*, the signature in [18] and the Schnorr's signature [32,33], can help us to design the lattice-based counterparts of DLP-based linkable ring signatures. In this paper, using the techniques in [12], we construct a linkable ring signature from ideal-lattices in which the size of a signature, on behalf of a ring with N participants, is $O(\log N)$. Based on the proposed signature scheme, we present an anonymous post-quantum cryptocash system by following the major ideas in CryptoNote and Monero. In order to generate stealth addresses (verifying keys) and recover corresponding signing keys for the linkable ring signature, we provide a key-generation protocol as a subroutine of the cryptocash system. By combining all those techniques together, our cryptocash protocol obtains a new level anonymity comparing to the original Bitcoin system. Furthermore, the new designed cryptocash system has the potential to resist quantum attacks.

Recently, the unlinkability and untraceability of Monero were analyzed by [24] and [14]. Some of them were blamed on the abuses of users, *e.g.* signing a transaction on behalf of a ring with only 1 participant. Besides, there are still a few inherent weakness in Monero, *e.g.* for an overwhelming proportion of input addresses, a user cannot find enough addresses with the same value to hide his real address, especially in the early time of the system. Next, we shall trace these problems and discuss what should be done to make our cryptocash system secure under these analyses. A full cryptocash system will be implement to test the communication and computation costs. And if possible, we would like to contribute our system to the cryptocash community for public usage.

Acknowledgements. This work was supported by the National Key R&D Program of China (2017YFB0802503), the National Natural Science Foundation of China (No. 61672550) and the Fundamental Research Funds for the Central Universities (No.17lgjc45).

The authors are grateful to the anonymous reviewers for their valuable suggestions and comments on this paper.

References

1. Aharonov, D., Regev, O.: Lattice problems in NP ∩ coNP. J. ACM **52**(5), 749–765 (2005)
2. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: STOC 1996, pp. 99–108. ACM (1996)
3. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better – how to make Bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_29
4. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_33
5. Cai, J.Y., Nerurkar, A.P.: An improved worst-case to average-case connection for lattice problems. In: FOCS 1997, pp. 468–477. IEEE, October 1997

6. Cayrel, P.-L., Lindner, R., Rückert, M., Silva, R.: A lattice-based threshold ring signature scheme. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 255–272. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14712-8_16
7. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
8. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
9. Fujisaki, E., Suzuki, K.: Traceable ring signature. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 181–200. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_13
10. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC 2008, pp. 197–206. ACM (2008)
11. Goldreich, O., Goldwasser, S.: On the limits of non-approximability of lattice problems. In: STOC 1998, pp. 1–9. ACM (1998)
12. Groth, J., Kohlweiss, M.: One-out-of-many proofs: or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 253–280. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_9
13. Halevi, S.: A sufficient condition for key-privacy. Cryptology ePrint Archive, Report 2005/005 (2005)
14. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of Monero's blockchain. Cryptology ePrint Archive, Report 2017/338 (2017)
15. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_1
16. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for Ad Hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_28
17. Liu, J.K., Wong, D.S.: Linkable ring signatures: security models and new schemes. In: Gervasi, O., et al. (eds.) ICCSA 2005. LNCS, vol. 3481, pp. 614–623. Springer, Heidelberg (2005). https://doi.org/10.1007/11424826_65
18. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
19. Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_13
20. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In: FOCS 2002, pp. 356–365 (2002)
21. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Comput. Complex. **16**(4), 365–411 (2007)

22. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007)
23. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: anonymous distributed e-cash from Bitcoin. In: SP 2013, pp. 397–411, May 2013
24. Miller, A., Möser, M., Lee, K., Narayanan, A.: An empirical analysis of linkability in the Monero blockchain. eprint arXiv:1704.04299 (2017)
25. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. URL: http://www.bitcoin.org/bitcoin.pdf (2012)
26. Noether, S.: Ring signature confidential transactions for Monero. Cryptology ePrint Archive, Report 2015/1098 (2015)
27. Ober, M., Katzenbeisser, S., Hamacher, K.: Structure and anonymity of the Bitcoin transaction graph. Future Internet **5**(2), 237–250 (2013)
28. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Altshuler, Y., Elovici, Y., Cremers, A., Aharon, N., Pentland, A. (eds.) Security and Privacy in Social Networks, pp. 197–223. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-4139-7_10
29. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_32
30. Ron, D., Shamir, A.: Quantitative analysis of the full Bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_2
31. Saberhagen, N.V.: Cryptonote v2.0 (2013). https://cryptonote.org/whitepaper.pdf
32. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 688–689. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_68
33. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
34. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_36
35. Sun, S.-F., Au, M.H., Liu, J.K., Yuen, T.H.: RingCT 2.0: a compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 456–474. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_25
36. Wang, C., Wang, H.: A new ring signature scheme from NTRU lattice. In: ICCIS 2012, pp. 353–356. IEEE, August 2012
37. Wang, J., Sun, B.: Ring signature schemes from lattice basis delegation. In: Qing, S., Susilo, W., Wang, G., Liu, D. (eds.) ICICS 2011. LNCS, vol. 7043, pp. 15–28. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25243-3_2
38. Zhang, H., Zhang, F., Tian, H., Au, M.H.: Anonymous post-quantum cryptocash. Cryptology ePrint Archive, Report 2017/716 (2017)

SpaceMint: A Cryptocurrency Based on Proofs of Space

Sunoo Park[1], Albert Kwon[1], Georg Fuchsbauer[2], Peter Gaži[3], Joël Alwen[4], and Krzysztof Pietrzak[4(✉)]

[1] MIT, Cambridge, USA
[2] Inria, ENS, CNRS and PSL, Villeneuve d'Ascq, France
[3] IOHK, Hong Kong, China
[4] IST Austria, Klosterneuburg, Austria
pietrzak@ist.ac.at

Abstract. Bitcoin has become the most successful cryptocurrency ever deployed, and its most distinctive feature is that it is decentralized. Its underlying protocol (Nakamoto consensus) achieves this by using *proof of work*, which has the drawback that it causes the consumption of vast amounts of energy to maintain the ledger. Moreover, Bitcoin mining dynamics have become less distributed over time.

Towards addressing these issues, we propose SpaceMint, a cryptocurrency based on *proofs of space* instead of proofs of work. Miners in SpaceMint dedicate *disk space* rather than computation. We argue that SpaceMint's design solves or alleviates several of Bitcoin's issues: most notably, its large energy consumption. SpaceMint also rewards smaller miners fairly according to their contribution to the network, thus incentivizing more distributed participation.

This paper adapts proof of space to enable its use in cryptocurrency, studies the attacks that can arise against a Bitcoin-like blockchain that uses proof of space, and proposes a new blockchain format and transaction types to address these attacks. Our prototype shows that initializing 1 TB for mining takes about a day (a one-off setup cost), and miners spend on average just a fraction of a second per block mined. Finally, we provide a game-theoretic analysis modeling SpaceMint as an *extensive game* (the canonical game-theoretic notion for games that take place over time) and show that this stylized game satisfies a strong equilibrium notion, thereby arguing for SpaceMint's stability and consensus.

1 Introduction

E-cash was first proposed by Chaum [8] in 1983, but did not see mainstream interest and deployment until the advent of Bitcoin [26] in 2009. With a market cap of over 300 trillion US dollars by December 2017, Bitcoin has given an unprecedented demonstration that the time was ripe for digital currencies.

In an early version, our proposal was called "Spacecoin". We changed it to "SpaceMint" due to name conflicts.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 480–499, 2018.
https://doi.org/10.1007/978-3-662-58387-6_26

On the flip side, Bitcoin's dramatic expansion has provoked serious questions about the currency's long-term sustainability. Bitcoin miners produce proofs of work (PoW) to add blocks to the *blockchain*, the public ledger of all transactions. For each block added, there is a reward of newly minted coins. One concern is that proofs of work deplete large amounts of natural resources: by some estimates from December 2017, the Bitcoin network consumed over 30 terawatt-hours per year, which exceeds Denmark's energy consumption. Moreover, most mining is currently done by specialized ASICs, which have no use beyond Bitcoin mining.

A related concern is the emergence of a "mining oligarchy" controlled by a handful of powerful entities. One of the original ideas behind basing Bitcoin mining on computing power was that anyone could participate in the network by dedicating their spare CPU cycles, incurring little cost as they would be repurposing idle time of already-existing personal computers. However, modern Bitcoin mining dynamics have become starkly different [31]: the network's mining clout is overwhelmingly concentrated in large-scale mining farms using special-purpose hardware for Bitcoin mining, often in collaboration with electricity producers. As a result, mining with one's spare CPU cycles today would result in net *loss* due to electricity costs. This phenomenon undermines the stability and security intended by the original decentralized design.

In light of these issues, there has been increasing interest in cryptocurrencies based on alternatives to proofs of work. The most explored alternative is *proofs of stake* (PoStake), in which a miner's probability of successfully creating a block increases with the amount of currency he holds, rather than the amount of computation he performs. This concept has several incarnations, from ad-hoc implementations in existing cryptocurrencies [9,22] to designs with rigorous security proofs in various models [10,11,21,23]. While these are innovative proposals, the early constructions have variously suffered from attacks that arise due to the inexpensive nature of mining. On the other hand, the more recent proposals are fairly complex, usually running some kind of Byzantine agreement protocol among a sufficiently large subset of stakeholders, and thus diverge substantially from the simplicity of the original Nakamoto design. Such schemes also typically fail in case of low participation (i.e., if stakeholders are not mostly online).

In this paper, we propose SpaceMint, a cryptocurrency that uses *proofs of space* (PoSpace) [4,14,30] to address the aforementioned issues that occur in Bitcoin and alternative proposals such as PoSpace-based currencies. To mine blocks in SpaceMint, miners invest *disk space* rather than computing power, and dedicating more disk space yields a proportionally higher expectation of successfully mining a block. SpaceMint has several advantages compared to a PoW-based blockchain like Bitcoin, summarized below.

- *Ecological:* Once the dedicated space for mining is initialized, the cost of mining is marginal: a few disk accesses with minimal computation.
- *Economical:* Unused disk space is readily available on many personal computers today, and the marginal cost of dedicating it to SpaceMint mining

would be small (by the previous point).[1] We thus expect that space will be dedicated towards mining even if the reward is much smaller than the cost of buying disk space for mining. In contrast, in PoW-based blockchains rational miners will stop mining if the reward does not cover the energy cost.
- *Egalitarian:* Bitcoin mining is done almost entirely on application-specific integrated circuits (ASIC) and by large "mining farms," to the point that small-scale participation (e.g., based on general-purpose hardware) is impossible. We believe SpaceMint to be less susceptible to specialized hardware than Bitcoin, as discussed in Sect. 5.
  Another cause of centralization of mining power in Bitcoin is mining pools. This paper does not address that problem directly, but an elegant and simple idea [25] to discourage mining pools in PoW-based blockchains – namely, having the mining process require the secret key to redeem the block reward – can be straightforwardly adapted for SpaceMint.[2]

1.1 Challenges and Our Contributions

In order to "replace" PoW by PoSpace to achieve consensus on the blockchain, the following problems must be addressed.

- *Interactivity:* PoSpace, as originally defined [14], is an interactive protocol. Although the same is true for the original definition of PoW [13], there the interaction was very simple (i.e., a two-message, public-coin protocol). PoSpace requires more interaction, thus it is more challenging to adapt PoSpace to the blockchain setting.
- *Determine the winner:* In a PoW-based blockchain like Bitcoin, the probability of a miner being the first to find an eligible next block increases with its hashing power. The Bitcoin protocol prescribes that once an eligible next block is announced, all miners should append that block to the blockchain and continue mining on the new longest chain. Generating a PoSpace, on the other hand, is deliberately computationally cheap. We thus need some way to determine which of many different proofs "wins". Moreover, the probability of any miner winning should be proportional to the space it dedicates, and we want a miner to learn if he is a likely winner without any interaction.
- *"Nothing-at-stake" problems:* When replacing PoW by proofs that are computationally easy to generate (such as PoStake or PoSpace), a series of problems arise known as *nothing-at-stake problems* [16].[3] The computation-intensive nature of Bitcoin mining is a key property that, informally, ensures that all

[1] By marginal cost we mean the cost of using disk space that otherwise would just sit around unused.

[2] In a PoW this can be achieved by, e.g., not applying the hash function to a nonce directly, but to its signature. In the PoS [14] used for SpaceMint, this can be achieved by augmenting each "label" that is stored with its signature.

[3] Although PoSpace-based currencies share some of the issues PoStake-based currencies have, they are robust to others, in particular, PoSpace does not share the tricky *participation* problem of PoStake.

miners are incentivized to concentrate their mining efforts on a single chain, which leads to consensus. When mining is computationally cheap however, miners can intuitively (1) mine on multiple chains simultaneously, not just the one the protocol specifies, and (2) try creating many different blocks with a single proof (of space or of stake) by altering the block contents slightly (e.g., by using different transaction sets) before choosing the most favorable one to announce. The latter behavior is known as "(block) grinding". Those issues are undesirable as

1. they slow down consensus;
2. they potentially allocate a greater reward to cheating miners
3. they potentially enable double-spending attacks by an adversary controlling much less than 50% of the space.

– *Challenge grinding:* Yet another issue arises when the content of past blocks can influence which blocks are added to the blockchain in future. Then it may be possible for a miner to generate a long sequence of blocks whose earlier blocks might have proofs of low quality, but are generated in a biased way (by "grinding" through all the possible proofs) so that the miner can create high quality proofs later in the sequence. The problem arises when the overall sequence is of higher quality than would be expected from the miner's disk space size, due to the disproportionately high quality of later blocks. Challenge grinding may be considered a nothing-at-stake problem, but we state it separately as, unlike the other nothing-at-stake problems, we have not encountered it in other contexts.

To tackle the *interactivity* problem, SpaceMint uses the Fiat-Shamir paradigm (a standard technique to replace a public-coin challenge with a hash of the previous message, already used to adapt PoW for Bitcoin); additionally, we leverage the blockchain itself to record messages of the PoSpace protocol (concretely, we use a special type of transaction to record the commitment to its space a prover needs to send to the verifier in the initialization phase of the PoSpace).

To *determine the winner*, we define a *quality function*, which assigns a quality value to a PoSpace proof. This function can be computed by the miner locally, and is designed such that the probability of a miner having the highest-quality proof in the network is proportional to the space it dedicates.

The *nothing-at-stake* problems are more challenging to solve. To tackle these, we introduce several new ideas and leverage existing approaches. To disincentivize miners from extending multiple chains we ensure such behavior is detected and penalize it. To prevent *block grinding*, SpaceMint ensures that the PoSpace is "unique", i.e., a miner can generate exactly one valid proof for every given challenge, and this challenge itself is uniquely determined by the proofs that were used to mine a previous block. This is done by basically running two chains in parallel, a "proof chain" that contains the proofs, and a "signature chain" that contains the transactions.

Finally, to address *challenge grinding*, SpaceMint prescribes that past blocks influence the quality of *short sequences* of future blocks, thus exponentially driving down the probability that a miner could generate a sequence of blocks of

disproportionately high quality by exploiting the relationship between past and future blocks.

The idea of making the challenge for a block a deterministic function of a unique credential of the resource that "won" a previous block – in combination with having a quality function by which miners can locally decide if they are likely winners – has been used in subsequent blockchain proposals like Algorand [23] or the Chia Network [3].

We also implement and evaluate the modified PoSpace to demonstrate the effectiveness of our scheme. Even for space larger than 1 TB, we show that (1) miners need less than a second to check if they are likely to "win" and therefore should generate a candidate next block, (2) block generation takes less than 30 s, and (3) verifying the validity of a block takes a fraction of a second. Moreover, these numbers grow logarithmically with larger space.

Finally, we provide a game-theoretic analysis of SpaceMint modeled as an *extensive game*. To do this, we formally specify a stylized model of SpaceMint mining and show that adhering to the protocol is a *sequential equilibrium* for rational miners in this game (i.e., deviating from the protocol does not pay off). Our analysis works in a simplified model that serves to rule out certain classes of attacks (i.e., profitable deviations based on a simplistic set of possible actions), but does not capture all possible attack vectors by real miners.[4] To our knowledge, this is the first analysis of a cryptocurrency mining as an extensive game with the corresponding game-theoretic equilibrium concepts; though the model is simplistic, we hope that this framework for rigorously ruling out certain classes of attacks will serve as a useful base upon which to build more nuanced game-theoretic models to rule out larger classes of attacks, in this and other similar cryptocurrencies.

1.2 Related Work

We have already discussed *proofs of stake* above. Here, we briefly mention other related proposals. A more detailed discussion can be found in the full version [27].

Proof of storage/retrievability [6,7,12,18,19, and many more] are proof systems where a verifier sends a file to a prover and later requests a proof that the prover really stored the file. Proving storage of a (random) file does show that one dedicated space, but the verifier must send the entire file first. In contrast, PoSpace requires verifier computation and communication to be polylogarithmic in the prover's storage size.

Proof of secure erasure (PoSE), one-time computable functions [5,15,20,29] are proof systems where a prover convinces the verifier that is has access to some space. Additionally one can require that the proof implies that the space also was erased [20,29], or some function can only be computed in forward direction [15].

[4] For example, "selfish mining" [17] or block withholding is not captured by our simplified model, and SpaceMint is in fact susceptible to block withholding attacks to a similar extent to Bitcoin.

Those protocols have only one phase, and thus cannot be used as a PoSpace, i.e., to efficiently prove space usage over time.

Permacoin [24] is a cryptocurrency proposal that uses proofs of retrievability with a novel variant of PoW. While solutions to Bitcoin's PoW puzzles carry no intrinsic value, Permacoin makes proof-of-work mining serve a useful purpose: miners are incentivized to *store useful data* and thus the network serves as a data archive. Permacoin is however still fundamentally a PoW-based scheme. In contrast, in SpaceMint the dedicated storage does not store anything useful, but we completely avoid PoW and the associated perpetual computation.

Burstcoin [1] is the only cryptocurrency we are aware of in which disk space is the primary mining resource. However, Burstcoin's design allows *time/memory trade-offs*: i.e., a miner doing a little extra computation can mine at the same rate as an honest miner, while using just a small fraction (e.g., 10%) of the space. Moreover, Burstcoin requires a constant (albeit small) fraction (0.024%) of dedicated disk space to be read every time a block is mined, while SpaceMint requires only a logarithmic fraction. Finally, verification in Burstcoin is problematic: miners must hash over 8 million blocks to verify another miner's claim. The details on this attacks can be found in Appendix B of the full version [27].

Chia Network [3] is a very recent proposal of a blockchain based on PoSpace in combination with proofs of sequential work. In a nutshell, the better the quality of the PoSpace, the faster the block can be "finalized" by a proof of sequential work, and this proof tuple then can be used to create a block. By using proofs of sequential work on top of PoSpace, Chia is even more similar to Bitcoin than SpaceMint in several respects: for example, it requires no synchronization/clocks (except, as in Bitcoin, time-stamped blocks for the occasional re-calculation of the mining difficulty), while retaining the efficiency of a pure PoSpace-based currency. The PoSpace that was developed for Chia [4] is based on ideas completely different from the PoSpace [14] we use. It has worse asymptotic security guarantees, but unlike [14], it has a non-interactive initialization phase and extremely short and efficient proofs.

Outline

- *Cryptocurrency from proofs of space:* In (Sects. 2 and 3) we modify PoSpace [14] for the blockchain setting and present SpaceMint, a cryptocurrency based purely on proofs of space.
- *Addressing the "nothing-at-stake" problems:* After describing attacks that arise from nothing-at-stake problems and challenge grinding, we describe how our design uses novel approaches to overcome them (Sect. 4). Our solutions extend to other blockchain designs based on easy-to-generate proofs.
- *Evaluation of proof of space:* We evaluate our modified PoSpace in terms of time to initialize the space, to generate and verify blocks, and block size (Sect. 5).
- *Game theory of SpaceMint:* We model SpaceMint as an extensive game, and show that adhering to the protocol is an *ε-sequential Nash equilibrium* (Sect. 6).

Algorithm 1. Space commit

Common input: A hard-to-pebble graph G with n nodes and a function $\mathsf{hash}\colon \{0,1\}^* \to \{0,1\}^L$.

1. $\mathcal{P}$ generates a unique nonce μ and then computes and stores $(\gamma, S_\gamma) := \mathsf{Init}(\mu, n)$, and sends the nonce[9] μ and the commitment γ to $\mathcal{V}$. S_γ contains the labels of all the nodes of G computed using Eq. (1) and γ is a Merkle-tree commitment to these n labels. The total size of S_γ is $N = 2 \cdot n \cdot L$ (graph + Merkle tree).

2 Proof of Space in SpaceMint

A PoSpace [14] is a two-phase protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. After an *initialization phase*, $\mathcal{P}$ stores some data S_γ of size N, and $\mathcal{V}$ stores a short commitment γ to S_γ. Then, in the *execution phase*, $\mathcal{V}$ sends a challenge c to $\mathcal{P}$, who returns a short answer a after reading a small fraction of S_γ.

The PoSpace from [14,30] are specified a family of "hard-to-pebble" directed acyclic graphs of increasing size. The prover picks a graph $G = (V, E)$ from this family depending on the amount of space it wants to dedicate. $\mathcal{P}$ then stores a label l_i for each node $i \in V$, which is computed as

$$l_i := \mathsf{hash}(\mu, i, l_{p_1}, \ldots, l_{p_t}), \tag{1}$$

where $p_1, \ldots, p_t$ are the parents of node i and hash is a hash function (sampled by $\mathcal{V}$). In [14] two graph families are suggested, one for which any successful cheating prover must either use $\Omega(|V|/\log(|V|))$ space between the initialization and execution phase, or use $\Omega(|V|/\log(|V|))$ space during execution. The other graph family enforces either $\Theta(|V|)$ space between the phases (i.e., the same as the honest prover, up to a constant), or $\Theta(|V|)$ time during execution.

Formally, [14] specifies a PoSpace by a tuple of algorithms $\{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$, which specify a two-phase protocol between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. Init is used to initialize the space, Chal generates a challenge, Ans computes the response to a challenge and Vrfy verifies the response. The initialization phase consists of running Algorithm 1,[5] where $\mathcal{P}$ commits to its space, followed by Algorithm 2, where $\mathcal{P}$ proves that the commitment is computed "mostly correct". In the execution phase, given by Algorithm 3, $\mathcal{V}$ simply opens some of the committed labels to prove it has stored them.

The algorithms we give here are already made partially non-interactive for our blockchain application – in the actual PoSpace the challenges in Algorithms 2 and 3, as well as μ in Algorithm 1 are sampled by $\mathcal{V}$ and sent to $\mathcal{P}$.

[5] The nonce just ensures that the same space cannot be used for two different proofs [14]; thus in a single-verifier setting, $\mathcal{P}$ can generate the nonce.

Algorithm 2. Prove commit

Initial state: $\mathcal{V}$ holds commitment γ and nonce μ; $\mathcal{P}$ stores S_γ and μ. Both are given the challenges $c = (c_1, \ldots, c_{k_v})$ to be used.

1. $\mathcal{P}$ computes openings $b := (b_1, b_2, \ldots)$ of all the labels of the nodes $\{c_i\}_{i \in [k_v]}$ *and of all their parents* and sends them to $\mathcal{V}$. This is done using Ans where $\mathsf{Ans}(\mu, S_\gamma, c)$ returns the Merkle inclusion proof of label l_c w.r.t. γ.
2. $\mathcal{V}$ verifies these openings using Vrfy, where $\mathsf{Vrfy}(\mu, \gamma, c, a) = 1$ iff a is a correct opening for c. It then checks for all $i = 1, \ldots, k_v$ if the label l_{c_i} is correctly computed as in Eq. (1).

Algorithm 3. Prove space

Initial state: $\mathcal{V}$ holds commitment γ and nonce μ; $\mathcal{P}$ stores S_γ and μ. Both are given the challenges $c = (c_1, \ldots, c_{k_p})$ to be used.

1. $\mathcal{P}$ computes openings $\{a_i := \mathsf{Ans}(\mu, S_\gamma, c_i)\}_{i \in [k_p]}$ and sends them to $\mathcal{V}$.
2. $\mathcal{V}$ verifies these openings by executing $\mathsf{Vrfy}(\mu, \gamma, c_i, a_i)$.

3 SpaceMint Protocol

3.1 Mining

The mining process consists of two phases: initialization and mining.

Initialization. When a miner first joins the SpaceMint network and wants to contribute N bits of space to the mining effort, it first generates a public/secret key pair (pk, sk) and runs Algorithm 1 as $\mathcal{P}$, with nonce μ set to pk, to generate

$$(\gamma, S_\gamma) := \mathsf{Init}(pk, N).$$

The miner stores (S_γ, sk) and announces its space commitment (pk, γ) via a special transaction. We require miners to commit (pk, γ) to prevent a type of grinding attack: the problem is that the PoSpace we use [14] have the property that by making minor changes one can turn (pk, γ) into many other space commitments that re-use most of the space.

Once this transaction is in the blockchain, the miner can start mining.

Mining. Similar to Bitcoin, SpaceMint incentivizes mining (adding new blocks) through block rewards (freshly minted coins per block) and transaction fees. Once initialized, each miner attempts to add a block to the blockchain every time period. For time period i, a miner proceeds as follows:

1. Retrieve the hash value of the last block in the best chain so far, and a challenge c (we discuss how c is derived in Sect. 3.4), which serves as a short seed from which we derive two long random strings $\$_p, \$_v$.
2. Compute challenges $(c_1, \ldots, c_{k_p}) := \mathsf{Chal}(n, k_p, \$_p)$ for use in Algorithm 3.
3. Compute the proof of space $a = \{a_1, \ldots, a_{k_p}\}$ using Algorithm 3.

4. Compute the quality $\mathsf{Quality}(pk, \gamma, c, a)$ of the proof (details of the quality function are given in Sect. 3.5).
5. If the quality is high enough, so that there is a realistic chance of being the best answer in period i, compute the proof of correct commitment $b = \{b_1, \ldots, b_{k_v}\}$ using Algorithm 2; then create a block and send it to the network in an attempt to add it to the chain. This block contains the proofs a and b computed above and a set of transactions; the exact specification is in given Sect. 3.2 below.

Remark 1. (Postponing Algorithm 2). Note that unlike in the interactive PoSpace where one runs Algorithms 1 and 2 during initialization, we only require miners to execute Algorithm 2 if they want to add a block. This is done for efficiency reasons. For one thing, this way, the proof b (which is significantly larger than a or γ) must only be recorded in the blockchain once the corresponding space has actually been used to mine a block. Another more subtle advantage is that now the challenge for Algorithm 2 changes with every block; thus a cheating miner (who computed some of the labels incorrectly) will only know if he was caught cheating at the same time when he generates a potentially winning proof a (and if b does not pass, he cannot use a). This allow us to tolerate a much larger soundness error in Algorithm 2, which means we can choose a smaller k_v (concretely, it's ok if he passes the proof with large probability p, as long as this requires using at least a p times the space an honest miner would use).

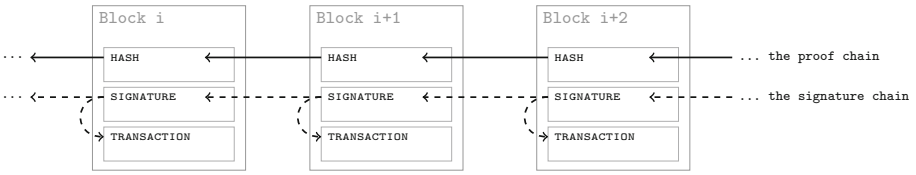


Fig. 1. Our blockchain consists of a proof chain that prevents grinding, and a signature chain that binds the transactions to the proof chain.

3.2 Blockchain Format

A blockchain in SpaceMint is a sequence of blocks $\beta_0, \beta_1, \ldots$ which serve as a public ledger of all transactions. Each block $\beta_i = (\phi_i, \sigma_i, \tau_i)$ consists of three parts, called "sub-blocks", which contain the index i that specifies the position of the block in the blockchain. The structures of sub-blocks are as follows:

- The HASH sub-block ϕ_i contains
  - the current block index i,
  - the miner's signature ζ_ϕ on ϕ_{i-1}, the $(i-1)$th HASH sub-block, and
  - a "space proof" containing the miner's pk.

- The TRANSACTION sub-block τ_i contains
  - the current block index i and
  - a list of *transactions* (Sect. 3.3).
- The SIGNATURE sub-block σ_i contains
  - the current block index i,
  - the miner's signature ζ_τ on τ_i, the ith TRANSACTION sub-block, and
  - the miner's signature ζ_σ on σ_{i-1}, the $(i-1)$th SIGNATURE sub-block.

The links between blocks in a blockchain are illustrated in Fig. 1. We will refer to the hash sub-blocks as the *proof chain*, and the signature sub-blocks with the transactions as the *signature chain*. While the signature and transaction sub-blocks are all linked, the hash sub-blocks are only linked to each other and not to any signature or transaction sub-blocks.

This design may seem to prevent any kind of consensus, as now we can have arbitrary many signature chains containing different transactions consistent with the same proof chain. The key observation is that once an honest miner adds the ith block (honest in the sense that he will only sign one block and keep its secret key secret), the transactions corresponding to this proof chain up to block i cannot be changed any more even by an adversary who controls all secret keys from miners that added the first $i-1$ blocks.

3.3 Transactions in SpaceMint

There are three types of transactions in SpaceMint: (1) payments, (2) space commitments, and (3) penalties. Every transaction is signed by the user generating the transaction, and sent to the miners to be added to the blockchain. Here, we specify the three types of transactions.

Payments. Coins in SpaceMint are held and transferred by parties identified by public keys. A payment transaction transfers coins from m benefactors to n beneficiaries and has the form

$$ctx = (\mathsf{payment}, txId, \boldsymbol{in}, \boldsymbol{out}).$$

- $txId$: A unique, arbitrary *transaction identifier*. That is, no two transactions in a blockchain can have the same identifier.
- $\boldsymbol{in}$: A list of input coins to the transaction. Specifically, $\boldsymbol{in} = (in_1, \ldots, in_n)$, a list of n *benefactors*, each being a triple: $in_j = (txId_j, k_j, sig_j)$, where:
  - $txId_j$ is the identifier of a past transaction,
  - k_j is an index that specifies a beneficiary pk_{k_j} of the transaction $txId_j$,
  - sig_j is a signature of $(txId, txId_j, k_j, \boldsymbol{out})$, which verifies under key pk_{k_j} proving ownership of the the k_jth beneficiary of transaction $txId_j$ and binding the coin to the beneficiaries.
- $\boldsymbol{out}$: A list of beneficiaries and the amount they receive. Specifically, $\boldsymbol{out} = (out_1, \ldots, out_m)$ with $out_i = (pk_i, v_i)$, where:
  - pk_i specifies a *beneficiary*, and
  - v_i is the number of coins that pk_i is to be paid.

For a transaction to be valid, we require that (1) all signatures in ***in*** verify correctly; (2) no benefactor is referenced by more than one subsequent transaction in the blockchain (to prevent double-spending); (3) the sum of the input values to the transaction is at least the sum of the amounts paid to beneficiaries.

Space Commitments. A space-commitment transaction

$$ctx = (\mathsf{commit}, txId, (pk, \gamma))$$

consists of pk, a public key, and γ which was computed as $(\gamma, S_\gamma) := \mathsf{Init}(pk, N)$. Thus, ctx is a space commitment to a space of size N.

Penalties. A penalty transaction

$$ctx = (\mathsf{penalty}, txId, pk, prf)$$

consists of pk, the public key of the transaction creator, and prf, a proof of penalty-worthy behavior by another miner. These transactions serve to penalize miners that engage in malicious behavior. The primary usage of penalties in SpaceMint is to disincentivize mining on multiple chains (e.g., the proof would contain two blocks of the same index signed by the same miner), but penalty transactions can be used to discourage other types of (detectable) behavior in blockchain-based currencies.

3.4 Where the Challenge Comes From

In Bitcoin, the PoW challenge for block i is simply the hash of block $i-1$. For SpaceMint, using block $i-1$ for the challenge can slow down consensus: If there are many different chains, miners can get different challenges for different chains. A rational miner would thus compute answers for many different chains (since it is easy to do), and if one of them is very good, try to add a block to the corresponding chain, even if this chain is not the best chain seen so far. If all miners behave rationally, this will considerably slow down consensus, as bad chains get extended with blocks of quality similar to the current best chain, and it will take longer for lower-quality chains to die off.

Instead, we derive the challenge for block i from the hash of block $i-\Delta$, for a reasonably large Δ: the probability of multiple chains surviving for more than Δ blocks decreases exponentially as Δ increases. Moreover, in contrast to Bitcoin, we only hash the block from the *proof chain*, but not the *signature chain* (Fig. 1): this serves to prevent block-grinding attacks, since there is nothing to grind on (the proof chain is fixed regardless of the set of transactions in the block). Finally, we will use the same challenge not just for one, but for δ consecutive blocks. This is done to prevent challenge-grinding attacks, as we explain in Sect. 4.

3.5 Quality of Proofs and Chains

Quality of a Proof. The block to be added to the chain at each time step is decided by a *quality* measure on the PoSpace proof included in each proposed

block. For a set of valid proofs $\pi_1 = (pk_1, \gamma_1, c_1, a_1), \ldots, \pi_m = (pk_m, \gamma_m, c_m, a_m)$, we require $\mathsf{Quality}(\pi_i)$ to be such that the probability that π_i has the *best* quality among $\pi_1, \ldots, \pi_m$ corresponds to the ith miner's fraction of the total space in the network. The probability is over the choice of the random oracle hash, which we use to hash answer a. We require:

$$\Pr_{\mathsf{hash}}\left[\forall j \neq i \,:\, \mathsf{Quality}(\pi_i) > \mathsf{Quality}(\pi_j)\right] = \frac{N_{\gamma_i}}{\sum_{j=1}^{m} N_{\gamma_j}},$$

where N_{γ_i} is the space committed to by γ_i.

Let D_N be a distribution that samples N values in $[0,1]$ at random and outputs the largest of them:

$$D_N \sim \max\left\{r_1, \ldots, r_N : r_i \leftarrow [0,1], i \in [N]\right\}. \tag{2}$$

Let $D_N(\tau)$ denote a sample from D_N with sampling randomness τ. For valid proofs we now define

$$\mathsf{Quality}(pk, \gamma, c, a) := D_{N_\gamma}(\mathsf{hash}(a)). \tag{3}$$

The $\mathsf{Quality}$ of an invalid proof is set to 0.

It remains to show how to efficiently sample from the distribution D_N for a given N. Recall that if F_X denotes the cumulative distribution function (CDF) of some random variable X over $[0,1]$. If the inverse F_X^{-1} exists, then $F_X^{-1}(U)$ for U uniform over $[0,1]$ is distributed as X. The random variable X sampled according to distribution D_N has CDF $F_X(z) = \Pr[X \leq z] = z^N$, since this is the probability that all N values r_i considered in (2) are below z. Therefore, if we want to sample from D_N, we can simply sample $F_X^{-1}(U)$ for U uniform over $[0,1]$, which is $U^{1/N}$. In (3) we want to sample $D_{N_{\gamma_i}}$ using randomness $\mathsf{hash}(a_i)$. To do so, we normalize the hash outputs in $\{0,1\}^L$ to a value in $[0,1]$, and get

$$D_{N_{\gamma_i}}(\mathsf{hash}(a_i)) := \left(\mathsf{hash}(a_i)/2^L\right)^{1/N}.$$

Quality of a Chain. In order to decide which of two given proof-chain branches is the "better" one, we also need to define the quality of a proof chain $(\phi_0, \ldots, \phi_i)$, which we denote by $\mathsf{QualityPC}(\phi_0, \ldots, \phi_i)$. Each hash sub-block ϕ_j contains a proof $(pk_j, \gamma_j, c_j, a_j)$, and the quality of the block is $v_j = D_{N_j}(\mathsf{hash}(a_j))$. For any quality $v \in [0,1]$, we define

$$\mathcal{N}(v) = \min\left\{N \in \mathbb{N} \,:\, \Pr_{w \leftarrow D_N}[v < w] \geq 1/2\right\},$$

the space required to obtain a proof with quality better than v on a random challenge with probability $1/2$. This quantity captures the amount of space required to generate a proof of this quality.

In order to prevent challenge-grinding attacks, it is desirable for the chain quality to depend *multiplicatively* on constituent block qualities (described in more detail in Sect. 4), and moreover it is useful to weight the contribution of

the jth block for a chain of length i by a *discount factor* Λ^{i-j}. From these motivations we derive the following quality function. Note that we have used a sum of logarithms, rather than a product, to achieve the multiplicativity.

$$\mathsf{QualityPC}(\phi_0, \ldots, \phi_i) = \sum_{j=1}^{i} \log(\mathscr{N}(v_j)) \cdot \Lambda^{i-j}. \tag{4}$$

4 Nothing-at-Stake Problems and Solutions

In this section we discuss the "nothing-at-stake" issues, which were already mentioned in the introduction. We describe them here in more detail, and outline how SpaceMint defends against them.

Recall that the difficulty arises due to the ease of computing multiple candidate blocks: in a PoSpace (or PoStake) based currency, a miner can compute *many* proofs (either extending different chains, or computing different proofs for the same chain) at little extra cost. Deviating from the protocol like this can be rational for a miner as it might lead to higher expected rewards. PoW-based blockchains also suffer from such "selfish mining" attacks [17], and basing the blockchain on efficiently computable proofs like PoSpace or PoStake can further aggravate this problem. Such behavior can significantly slow down consensus as well as push the scheme to follow energy expenditure trends similar to PoW-based schemes, which arise whenever there is an advantage to be gained by doing extra computation.

An even more serious issue is double-spending attacks, which become possible if a miner can create a sufficiently long chain in private which has better quality than the honestly mined chain. In all known blockchain proposals, a miner controlling more than half of the mining resources (hashing power, stake or space) can do this. But it is considered problematic if a blockchain is susceptible to double spending by adversaries with significantly less than half of the network resources.

I. Grinding Blocks. *The problem:* In Nakamoto-style blockchains, the challenge for the proof computed by the miners (like PoW in Bitcoin or PoSpace in SpaceMint) is somehow derived from previous blocks. If it is computationally easy to generate proofs, a miner can try out many different blocks (for example by including different transactions) until it finds an advantageous one that will allow him to generate good proofs for future blocks. This is an issue for selfish-mining and double-spending attacks.
The solution: We decouple proofs from transactions as shown in Fig. 1. This eliminates the problem of block grinding, as now challenges depend only on the *proof chain*. Moreover, our PoSpace are "unique" in that a prover can generate at most one valid proof per challenge. Hence, the only degree of freedom that a miner has in influencing future challenges is to either publish its proof (so it might end up in the chain), or to withhold it.

II. Mining on Multiple Chains. *The problem:* In Bitcoin, rational miners will always work towards extending the longest known chain. However, when

mining is computationally easy, it can be rational to mine on *all* (or at least many) known chains in parallel, to "hedge one's bets" across all chains that might eventually become part of the public ledger. Again, this is an issue for selfish-mining and double-spending attacks.

The solution: To address this problem in the context of selfish-mining attacks (we discuss double-spending later), we derive the challenge for block i from block $i - \Delta$ for some parameter Δ (Sect. 3.4). Let us consider two cases, depending on whether mining is done on two or more chains that forked *more* or *less* than Δ blocks in the past.

Case 1: chains forked less than Δ ago. In this case, the miner will get the same challenge for both chains. SpaceMint uses penalties (Sect. 3.3) to disincentivize miners from extending multiple chains in this case; without the penalties, a rational miner with a good-quality PoSpace proof could announce blocks on multiple chains to maximize his chances of winning. Concretely, suppose a miner pk' attempts to mine concurrently on two chains whose most recent blocks are β_j and β'_j, by announcing β_{j+1} and β'_{j+1} (which have the same quality and were mined using the same space). Then anyone who observes this can generate a transaction $(\mathsf{penalty}, txId, pk, \{pk', \beta_{j+1}, \beta'_{j+1}\})$ to penalize pk'. This transaction can be added to a chain extending β_{j+1} (or β'_{j+1}), and its meaning is that half of the reward (block reward and transaction fees) that should go to the miner who announced β_{j+1}, is now going to pk (the "accuser") instead, and the other half of the reward is destroyed, i.e., cannot be redeemed by any party. We destroy half of the reward so the penalty hurts even if the cheating miner can be reasonably sure to be able to accuse itself. For this to work, mining rewards can only be transferred by a miner some time after the block was added, so that there is enough time for other miners to claim the penalty.[6]

Case 2: chains forked more than Δ ago. In this case the miner receives different challenges for different chains, leading to proofs of different quality for the two chains. In this case, even with our penalty scheme in place, a rational miner can still get an advantage by deviating: instead of only trying to extending the highest-quality chain, it also generates proofs for the lesser chain. As the challenges differ, so will the two proofs, and if the proof on the lesser chain has very high quality, the rational miner would publish it, hoping that this chain will become the best chain and survive.

We address this problem by arguing that it is extremely unlikely (the probability is exponentially small in Δ) that this case occurs, as a weaker branch of the chain would have to "survive" for Δ blocks despite a strong incentive (via our punishment scheme) for miners to only extend the chain of highest quality.

III. Grinding Challenges. *The problem:* Challenge grinding is a type of attack that can be used for double-spending, by generating a long chain in private that

[6] The idea of penalizing miners for extending multiple chains goes back at least to slasher https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm. Unlike previous penalty-based proposals, we do not need the miners to make a deposit up-front; instead, they will simply lose their mining reward if they cheat.

is of higher quality than what would result when using one's resources honestly. It arises from the fact that an adversary can split its space into m smaller chunks. As discussed in Sect. 3.5, the quality of a block is purposely designed such that splitting a fixed amount of space into smaller chunks and choosing the highest-quality block among them does *not* affect the expected quality of the block generated. However, a miner can examine all possible chains of some given length, and then pick the chain that gives it the most favorable challenges for future blocks.

Concretely, consider our setting, where the challenge for block i is determined by block $i - \Delta$. An adversary can generate a sequence of length 2Δ where the first half of the blocks is chosen to provide the most favorable challenges for the later half of the sequence.[7] Note that the first half of this sequence would be of even poorer quality than the expected quality from honest mining given the adversary's total amount of space; however, the benefit gained in the second half of the sequence can outweigh this loss in quality in the first half. The adversary can then release this high-quality chain (all at once) in an attempt to overtake the current best chain.

Note that in this attack the adversary explores multiple chains in parallel, which we have addressed already using a penalizing scheme. But penalizing does not protect against double-spending attacks in which the adversary never actually published two proofs for the same slot. And even he would, a double-spending attack can be profitable even if one loses some mining rewards due to the penalizing scheme.

The solution: As mentioned in Sect. 3.5, the problem with this attack is exacerbated if the metric for determining the quality of a chain is a sum or any other linear function. Thus, to prevent this attack, (1) we define the quality of a chain as the *product* of the amounts of space needed for the proofs in it, rather than their *sum*; and (2) we use the same block to derive challenges for δ future blocks (i.e., use $\mathsf{hash}(\beta_i, nonce)$ for $nonce \in [1, \delta]$ as challenges for time $i + \Delta$ through $i + \Delta + \delta$).

Intuitively, (1) makes it harder for the adversary to find a good chain of length 2Δ, as worse blocks are weighted more; and (2) is helpful because it means that a challenge-grinding adversary would have to choose "early" blocks to optimize their chances over sequences of δ future challenges rather than just a single future challenge, thus making it exponentially harder (in δ) to find a "good" challenge that will yield δ high-quality blocks at once. Another way to see this is that by the Chernoff bound, the average of δ independent random variables deviates less from its expectation as δ grows. So for large δ, even the ability to select between multiple challenges (each giving a sample of the average of δ i.i.d. variables) is not very useful to find one where this value deviates by

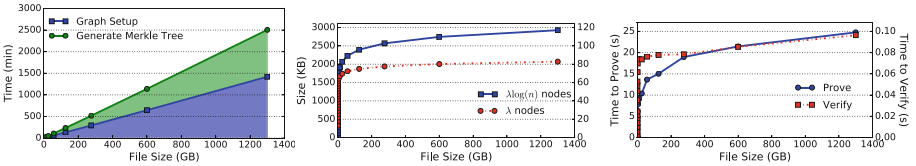
[7] As for each of the Δ blocks there are m distinct challenges, the search space here is of huge size m^{Δ}. Consequently, this attack might seem artificial, but by pruning and just considering the most promising sub-chains at every level, one will probably not miss the best one.

a lot from its expectation. A more detailed discussion of this attack and our defense is given in the full version [27].

5 Evaluation

To evaluate SpaceMint, we focused on space initialization time, proof size, and time for proof generation and verification. We implemented a prototype in Go using SHA3 in 256-bit mode as the hash function and the graphs from [28], which forces cheating provers to store $\Omega(N/\log(N))$ bits to efficiently generate proofs. We used a desktop computer equipped with an Intel i5-4690K Haswell CPU and 8 GB of memory, and an off-the-shelf 2 TB hard disk drive with 64 MB cache.

Figure 2a shows initialization time for spaces from 8 KB to 1.3 TB, which involves computing hashes of the nodes and a Merkle tree over them and is only done once. For 1.3 TB this takes approximately 41 h, which prevents re-using the same space for different commitments.



(a) Time to initialize space.

(b) Proof size when opening $\lambda \log(n)$ (left) or λ (right) nodes for $\lambda = 30$.

(c) Time for proving and verifying for opening $\lambda \log(n)$ nodes for $\lambda = 30$.

Fig. 2. Results of evaluation

A proof consists of the Merkle inclusion proof for a set of node labels. The number of nodes to be opened is $\lambda \cdot \log(n) + 1$ (as $k_v = \lambda \cdot \log(n)$ in Algorithm 2 and $k_p \ll k_v$ in Algorithm 3), where λ is a statistical security parameter. Every node has at most two parents, and each opening of a node is $\log(n) \cdot 32$ bytes. The overall proof size is thus at most $3 \cdot \lambda \cdot \log^2(n) \cdot 32$ bytes. (Opening fewer than $\lambda \log(n)$ nodes was not shown to be secure, but we are not aware of concrete attacks even for opening λ nodes; we believe that the number required for security lies somewhere in between, closer to opening λ nodes.) Fig. 2b shows the proof size for opening $\lambda \log(n)$ or only λ nodes, for $\lambda = 30$.

Assuming a miner is storing the space correctly, she only needs to open a small k_p number of nodes in the Merkle tree to check the quality of her solution (Sect. 3.1). As it takes <1 ms to read a hash from the disk, this takes just a fraction of a second. In the rare event that the answer is of very good quality, she generates the full proof, which takes less than 30 s. Since a miner must open k_p nodes in every time slot, we want k_p to be small. In practice $k_p = 1$ seems secure; for safety, we set it to a small constant. Proofs in SpaceMint are substantially bigger than in Bitcoin and require more than one hash evaluation to verify. However, for an active currency, the transactions contained in a block

will largely dominate block size and verification time. Figure 2c shows that while it takes seconds to generate the proof, it only takes a fraction of a second to verify it.

In terms of power consumption of the network, currently, the Bitcoin miners consume more than 32 TWh of energy annually, or around 220 GJ/min. Our prototype was evaluated on a full CPU, but a cost-conscious miner would mine on a more energy-efficient device, such as a Raspberry Pi [2], which consumes less than 10 W of power. Furthermore, we could set the quality threshold such that only a small number of miners, e.g., 1,000 miners, have a realistic chance of winning (i.e., have to generate a full proof). In such a scenario, the estimated energy consumption of a SpaceMint network per block would be

$$10W \cdot M \cdot 0.01s + 10W \cdot 1000 \cdot 20s = (200,000 + 0.1M)J/\text{block},$$

where M is the number of miners in the network. In such scenario, even with a billion miners in the network and with a block being added every minute, SpaceMint network would use less than 1% of the power of the current Bitcoin network.

Impact of Storage Medium. Almost all modern Bitcoin mining is done by clusters of application-specific integrated circuits (ASICs), which can compute hashes for a tiny fraction of the hardware and energy cost of a general-purpose processor. We believe that SpaceMint mining would not be as susceptible to advantages from specialized hardware as Bitcoin, and that regular hard disk drives are well-suited to serve as SpaceMint mining equipment. Let us consider existing categories of storage devices. Although hard disks are expensive compared to other storage devices, most notably tapes, devices like tapes are not adequate for mining as we require frequent random accesses to answer the PoSpace challenges. Solid state drives do allow for (fast) random accesses, but are more expensive than hard disks and do not provide any benefit since the rate of lookups required for mining is very low. Notably, SpaceMint mining hinges on doing a few random lookups every minute. The required frequency is so low that speed is a non-issue: cheap, *slow* random access is what SpaceMint miners are after.

6 Game Theory of SpaceMint

The miners in a cryptocurrency are strategic agents who seek to maximize the reward that they get for mining blocks. As such, it is a crucial property of a cryptocurrency that "following the rules" is an equilibrium strategy: i.e., it is important that the protocol is designed so that miners are never in a situation where deviating from the protocol is more profitable than behaving honestly.

To establish by a rigorous analysis that no deviations are beneficial is a natural theoretical goal. In game theory, such analyses are done in the context of a model specifying the "actions" that players can take during the course of a game in which each player's goal is to maximize her own utility. In our context,

utility corresponds to profit from mining rewards. It is assumed that "actions" describe all behavior that players may exhibit: in our context, this means that the notion of "following the protocol," as well as any possible deviations from the protocol, are assumed to be expressible by an action or sequence of actions.

To our knowledge, this paper presents the first rigorous equilibrium analysis in which cryptocurrency mining is modeled as an *extensive game*, the canonical game-theoretic notion for games that take place over time. Our analysis works in a simplified model that serves to rule out certain classes of attacks (i.e., profitable deviations by modeled actions). In our stylized game, play occurs over a series of discrete time steps, in each of which a block is added to the blockchain. At each time step, each player (miner) must choose a strategy, specified by: (1) which blocks to extend (if any), and (2) which extended blocks to publish (if any).

Our analysis herein is intended as a basic framework to model mining in blockchain-based cryptocurrencies as an extended game, and does not claim to comprise an exhaustive modeling of all possible attack vectors. In particular, our stylized model does not capture some important aspects, most notably block withholding, which is used in "selfish mining." Nevertheless, we believe that our simple modeling framework for cryptocurrency as an extended game can serve as a useful base upon which to build more nuanced game-theoretic models.

Due to space constraints, we defer to the full version [27] the details of the modeling of mining as an extensive game and the proof of equilibrium. Here, we give just an informal statement of our main equilibrium theorem.

Theorem 1. *It is a* sequential equilibrium *of the SpaceMint game (defined in* [27, Sect. 7]*) for all computationally bounded players to adhere to the mining protocol, provided that no player holds more than 50% of all space.*

Showing that adhering to the protocol is an *equilibrium* of such a game means that rational miners are not incentivized to deviate from the protocol when playing the game: from this, it follows that rational miners will reach consensus on a single chain, as they would not be able to get an advantage by using a "cheating" strategy.

Acknowledgements. We would like to thank Andrew Miller and Bram Cohen for bringing the challenge-grinding attack to our attention. We thank Srini Devadas for useful feedback on draft versions of this paper, and Ethan Heilman for an interesting discussion about costs of modern Bitcoin mining.

Sunoo's research is supported by the following grants: NSF MACS (CNS-1413920), DARPA IBM (W911NF-15-C-0236), and SIMONS Investigator Award Agreement Dated June 5th, 2012. Georg is supported by the French ANR EfTrEC project (ANR-16-CE39-0002). Joël and Krzysztof are supported by the European Research Council (ERC) consolidator grant 682815 - TOCNeT.

References

1. Burstcoin. http://burstcoin.info
2. Raspberry Pi. www.raspberrypi.org

3. Chia network (2017). https://chia.network
4. Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.: Beyond Hellman's time-memory trade-offs with applications to proofs of space. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 357–379. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_13
5. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of space: when space is of the essence. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 538–557. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_31
6. Ateniese, G., et al.: Provable data possession at untrusted stores. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM CCS 2007, pp. 598–609. ACM Press, October 2007
7. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: CCSW, pp. 43–54 (2009)
8. Chaum, D. (ed.): Advances in Cryptology. Proceedings of Crypto 82, pp. 199–203. Springer, Boston (1983). https://doi.org/10.1007/978-1-4757-0602-4
9. T. N. Community. Nxt whitepaper, July 2014. https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf
10. Daian, P., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919 (2016). http://eprint.iacr.org/2016/919
11. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573 (2017). http://eprint.iacr.org/2017/573
12. Di Pietro, R., Mancini, L., Law, Y.W., Etalle, S., Havinga, P.: LKHW: a directed diffusion-based secure multicast scheme for wireless sensor networks. In: Proceedings of 2003 International Conference on Parallel Processing Workshops, pp. 397–406 (2003)
13. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_10
14. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 585–605. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_29
15. Dziembowski, S., Kazana, T., Wichs, D.: One-time computable self-erasing functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 125–143. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_9
16. Ethereum. Problems. https://github.com/ethereum/wiki/wiki/Problems
17. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
18. Golle, P., Jarecki, S., Mironov, I.: Cryptographic primitives enforcing communication and storage complexity. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 120–135. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36504-4_9
19. Juels, A., Kaliski Jr., B.S.: PORs: proofs of retrievability for large files. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM CCS 2007, pp. 584–597. ACM Press, October 2007
20. Karvelas, N.P., Kiayias, A.: Efficient proofs of secure erasure. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 520–537. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_30

21. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
22. King, S., Nadal, S.: PPcoin: peer-to-peer crypto-currency with proof-of-stake (2012)
23. Micali, S.: ALGORAND: the efficient and democratic ledger. CoRR, abs/1607.01341 (2016)
24. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: repurposing bitcoin work for data preservation. In: 2014 IEEE Symposium on Security and Privacy, pp. 475–490. IEEE Computer Society Press, May 2014
25. Miller, A., Kosba, A.E., Katz, J., Shi, E.: Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In: Ray, I., Li, N., Kruegel, C., (eds.), ACM CCS 2015, pp. 680–691. ACM Press, October 2015
26. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009). http://bitcoin.org/bitcoin.pdf
27. Park, S., Kwon, A., Fuchsbauer, G., Gaži, P., Alwen, J., Pietrzak, K.: SpaceMint: a cryptocurrency based on proofs of space. IACR Cryptol. ePrint Arch. **2015**, 528 (2015)
28. Paul, W.J., Tarjan, R.E., Celoni, J.R.: Space bounds for a game on graphs. Math. Syst. Theory **10**(1), 239–251 (1976)
29. Perito, D., Tsudik, G.: Secure code update for embedded devices via proofs of secure erasure. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 643–662. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_39
30. Ren, L., Devadas, S.: Proof of space from stacked expanders. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 262–285. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_11
31. S. Valfells and J. H. Egilsson. Minting money with Megawatts: How to mine Bitcoin profitably, September 2015. http://www.researchgate.net/publication/278027487_Minting_Money_With_Megawatts_-_How_to_Mine_Bitcoin_Profitably

Kaleidoscope: An Efficient Poker Protocol with Payment Distribution and Penalty Enforcement

Bernardo David[1(✉)], Rafael Dowsley[2,3], and Mario Larangeira[1,3]

[1] Tokyo Institute of Technology, Tokyo, Japan
{bernardo,mario}@c.titech.ac.jp
[2] Aarhus University, Aarhus, Denmark
rafael@cs.au.dk
[3] IOHK, Central, Hong Kong

Abstract. The two main challenges in deploying real world secure poker protocols lie in enforcing the distribution of rewards and dealing with misbehaving/aborting parties. Using recent advances in cryptocurrencies and blockchain techniques, Kumaresan et al. (CCS 2015) and Bentov et al. (ASIACRYPT 2017) were able to solve those problems for the general case of secure multiparty computation. However, in the specific case of secure poker, they leave major open problems in terms of efficiency and security. This work tackles these problems by presenting the *first* full-fledged *simulation-based* security definition for secure poker and the first *fully-simulatable* secure poker protocol that provably realizes such a security definition. Our protocol provably enforces rewards distribution and penalties for misbehaving parties, while achieving efficiency comparable to previous tailor-made poker protocols, which do not have formal security proofs and rewards/penalties enforcement. Moreover, our protocol achieves reduced on-chain storage requirements for the penalties and rewards enforcement mechanism.

1 Introduction

Shamir, Rivest and Adleman, soon after their seminal work on the RSA cryptosystem, started exploring new ideas on cryptography inspired by everyday activities such as playing games. In particular, they started investigating how to play poker remotely [26], a problem related to very interesting questions in the distributed setting. For example, securely shuffling with remote parties requires

B. David and M. Larangeira—This work was supported by the Input Output Cryptocurrency Collaborative Research Chair, which has received funding from Input Output HK.

R. Dowsley—This project has received funding from the European research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme (grant agreement No. 669255).

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 500–519, 2018.
https://doi.org/10.1007/978-3-662-58387-6_27

every player to participate in the procedure; otherwise, security may not be assured at all for the participants.

Mental Poker, Cryptography and the Gambling Market: Since its origins the research on mental poker and card games worked as a drive for the research in cryptography. The original work of Shamir et al. inspired a number of follow-ups, starting in the eighties with the works on the feasibility of playing mental games, e.g. [11]. The first protocols for mental poker faced several limitations due to poor efficiency, which was improved in the following decades several by a number of works, e.g. [3,9,25,28,29,32,33].

In economic terms, online poker has been a strong industry since the "Poker Boom" of the 2000s, as described in prestigious economic venues [15]. Much of the strong interest in online gambling has its advent due to the appearance of online casinos. Despite legal restrictions imposed by new US legislation, players resort to websites based in other countries. For example, a Financial Times report [1] describes how UK firms filled the vacuum left by the US counterparts in the estimated 40 billion dollars global market of international online gambling (with one of the major online casino reporting 22 millions users and revenue of 2.5 billions dollars).

The current model of online gambling is based on trusted casinos, which are responsible for generating the randomness used to shuffle the cards and for enforcing the proper execution of the game. In contrast, a real world poker game requires almost no trust among the players, or between players and third parties like casinos. In the current model, a malicious casino or an insider attacker working for a casino can greatly influence the outcome of the game by manipulating the randomness used for shuffling or by leaking additional information to the players. And such cases have already happened (see Section "Integrity and Fairness" of [30] for more details). This state of affairs represents a clear disadvantage from online poker in comparison with a game played face-to-face. Techniques from mental poker can be used to overcome this problem and securely play poker online without the need of trusted casinos.

Challenges Preventing Deployment: Two central problems preventing deployment of secure poker protocols that were not addressed in the literature until very recently are protecting against aborts and ensuring that winners get their rewards. The first problem consists in players who leave the game prematurely (i.e. abort the protocol execution) causing the protocol to freeze. Castellà-Roca et al. [9] investigated this scenario and proposed a protocol that we show to be flawed (details in the full version of this paper [12]). The second problem of ensuring that a player actually gets a reward if it wins has only been tackled very recently after the advent of cryptocurrencies and blockchain technologies. Kumaresan et al. [20] addressed the problem with the help of Bitcoin and blockchains following the approach of [2,6]. They concurrently also dealt with the abort problem in a far more satisfactory way by imposing financial penalties on the aborting parties and using the collected money to compensate the remaining players.

Basically, the protocol of Kumaresan et al. [20] uses an unfair multiparty computation protocol along with many simple smart contracts and Bitcoin deposits to ensure that the rewards are distributed to players whenever the relevant conditions are fulfilled, and to enforce financial penalties on aborting/misbehaving parties. Using this strategy, a specific poker protocol was also designed, although with inefficiencies (for a more detailed discussion see [7, Sect. 6]). A significant improvement was obtained by Bentov et al. [7] by leveraging the power of stateful contracts to greatly improve the efficiency, solving some of the bottlenecks in the previous protocol. While the protocol in [20] requires $O(n^2)$ rounds of interaction with the cryptocurrency network and an amount of collateral linear in the number of messages exchanged during the protocol, the protocol of Bentov et al. [7] requires $O(1)$ rounds of interaction with the cryptocurrency network and an amount of collateral equal to the compensation the players would receive. The central idea for improving the performance and decreasing the amount of collateral is to use a single stateful contract that keeps all the deposits and executes the unfair protocol off-chain. After the initial deposits, this contract is only involved in two situations: for the cash distribution, or if a problem happens.

Lack of Strong Security Proofs: Even though efficient solutions are known for different components of card games, most have not been formally proven secure in a strong security model. In fact, it has been observed in [25] that the protocols of [32,33] are broken and we describe in the full version of this work [12] new concrete flaws that we have identified in the protocols proposed in [9] and [3]. Out of the few protocols that have been suggested, it seems that only [20] and its follow-up work [7] present a more detailed security proof in a strong, simulation-based security model. However, in [7] only the general solution based on enhanced trapdoor permutations has a full security proof (but incurs high computational and communication costs due to its generality). Bentov et al. [7, Sect. 7] argue that, instead of the general protocol, the tailor-made protocol of Wei and Wang [28,29] can be used as a building block and coupled with their techniques for dealing with aborts and cash distribution in order to obtain more efficient poker protocols. However they do not present a proof for this claim, and not even define the security properties that such tailor-made poker protocol would have to satisfy in order for the overall solution to be secure. In fact, the security models used in [28,29] are not formally defined and seem to be rather weak (judging by the informal descriptions given in these works).

General Requirements for Useful Poker Protocol: The current state of the art is unsatisfactory as there is no solution that meets all the following criteria necessary in a deployment in a real world scenario in which money is at stake: **(1) Efficiency:** performance that is comparable to tailor-made poker protocols; **(2) Security:** a simulation-based, formal proof of security; **(3) Penalties:** avoiding aborts/misbehavior or penalizing the misbehaving players; **(4) Rewards:** securely distributing the rewards to the players.

The works that are closer to achieve these criteria are [20] and [7], which made fundamental progress towards providing viable solutions to satisfy conditions (3) and (4). Nevertheless, none of their solutions meet simultaneously conditions (1)

and (2). The solutions in [20] as well as the general solution in [7] do not meet condition (1), while the solution in [7] using tailor-made protocol improves on condition (1) but does not address (2) as it lacks a security proof.

1.1 Our Contribution

We present our protocol, *Kaleidoscope*, named after the homonymous poker themed movie from the sixties [18]. Given the earlier discussion, our main goal in this work is to design a poker protocol that concurrently meets all four criteria above. In designing our solution we face two main challenges: 1. constructing an efficient off-chain protocol without sacrificing provable security guarantees as in previous tailor-made poker protocols, 2. reducing the amount of data stored in the blockchain, which is a highly constrained resource. In summary, our contributions are: (1) First full-fledged simulation-based security definition for poker (check full version [12]); (2) First fully-simulatable poker protocol (Sect. 3), which provably realizes our security definition; (3) Improved concrete computational and communication complexities for off-chain card operations (around 10 times better than previous works) and reduced on-chain storage requirements for the penalties and rewards enforcement mechanism (estimated in Sect. 4).

As our goal is to provide a strong security guarantee, we first specify a poker functionality that encompasses the whole game execution, penalizes aborting parties and guarantees the distribution of the rewards. Such modeling of the whole poker game as an ideal functionality is, to the best of our knowledge, novel. Then we design a tailor-made protocol that provably realizes such functionality in a simulation-based security model. Our protocol is designed with both off-chain and on-chain efficiency in mind. We focus on the case where players act honestly and the on-chain protocol execution is used as a last resort to recover from malicious actions. In this context, we meet criteria (1) and (2) by designing an off-chain protocol that is highly efficient while providing *compact* witnesses to be posted to the blockchain for claiming rewards or enforcing penalties. Our protocol represents cards as ciphertexts of a threshold version of the well known El Gamal cryptosystem as proposed by Barnett and Smart [3] but significantly differs from their work in the techniques we employ for distributed key generation and card shuffling. Namely, we use a technique for distributed key generation of threshold El Gamal public keys that addresses the security issue we found in the protocol of [3] (described in the full version of this work [12]) without sacrificing efficiency. Moreover, we significantly improve the efficiency of the card shuffling procedure by leveraging recent advances in zero-knowledge proofs for correctness of shuffles [4]. This initial protocol itself is unfair, meaning that an adversarial abort can cause the execution to fail without consequences. In order to meet criteria (3) and (4), we build on top of the ideas in [20] and [7], financially penalizing an adversary and rewarding honest players through a stateful smart contract. We optimize their general rewards/penalties mechanism for the specific case of poker and define concrete compact witnesses of correct behavior, resulting in a smaller on-chain footprint.

1.2 Overview and Intuition of Our Protocol

Next we present a more detailed overview of our protocol. Due to the fact that it is not reasonable to assume that the majority of the players are honest in a poker game, the secure poker protocol will not be able to guarantee fairness. Instead, we follow the approach of imposing a financial penalty on the party that interrupts the correct execution of the protocol, and use this money to compensate the honest parties. A stateful contract is used to enforce these properties. As it is highly desirable to decrease the burden on the blockchain as much as possible (thus improving the efficiency and decreasing the impact on other users), the execution of the protocol is performed mostly off-chain and the parties only go back on-chain for the cash distribution or if some problem happens. When the protocol goes back on-chain, the parties need to present witnesses to the stateful contract to validate the state of the game. It is important to decrease the size of these witnesses that need to be stored by the players, as well as the verification costs for the stateful contract. In this regard, a key characteristic of poker is that the future execution is independent from the past when conditioned on a few variables that keep track of the current status. Hence, if all participants sign these variables at a *checkpoint*, then this constitutes a *witness* witness that can be delivered to the stateful contract in order to prove the state of the game at this particular point. Therefore, at the checkpoints, the players can delete all other previous witnesses, saving space for the players and verification efforts for the stateful contract. The general overview of the protocol is:

1. Initially the parties lock into the stateful contract functionality an amount of money equal to the sum of the collateral and the money that they will use for the bets. A few initialization procedures are also executed during this stage.
2. The players run our novel unfair tailor-made poker protocol off-chain. During this stage, an aborting adversary can cause the off-chain protocol to fail, so the players need to record a few witnesses that must be sent to the stateful contract in the case of problems that require its intervention. All messages are signed by the senders, and at some checkpoints a few variables that summarize the status of the game are signed by all players, constituting a *compact* witness of correct execution.
3. If the protocol finishes correctly off-chain, then the final payout amounts will have been signed by all players, and so the parties only come back on-chain for the cash distribution that is performed by the stateful contract.
4. If some problem happens and a player requests the intervention of the stateful contract, each party that does not want to get penalized handles their respective recorded witnesses to the stateful contract, which is then able to verify the latest status of the protocol execution and continue the execution (on-chain) under its mediation. During the mediated execution, it penalizes any participant that does not follow the protocol rules or abort.

Note that on Step 2, the adopted technique is used in order to decrease the size of the witnesses that the players need to store after the checkpoint as well as to reduce the amount of on-chain verification that needs to be performed in case

of intervention (thus reducing the burden on the blockchain, which affects all cryptocurrency's users). The safe deposit d that each of the n participants lock into the contract should be enough to pay the compensation amount q for all the other parties, i.e., $d \geq q(n-1)$. Obviously, the monetary compensation q should be related to the maximum possible bet amount m at each hand; otherwise the corrupted parties would have an incentive to abort the protocol if they notice that one hand will end up badly for them.

2 Preliminaries

We now define some building blocks used in our protocols. For details about the Decision Diffie Hellman problem and digital signatures check the full version [12].

Security Model, Adversarial Model and Setup Assumptions: We prove our protocol secure in the real/ideal simulation paradigm with sequential composition. This is an intuitive paradigm that provides strong security guarantees for the protocols that are proven secure according to it. For more details, check the full version of this work [12]. We consider *malicious* adversaries that may deviate from the protocol in arbitrary ways. Moreover we consider the *static* case, where the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. Our protocol uses the Random Oracle Model (ROM) [5] and assumes the existence of a stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$ (that is described in Sect. 3 and can be implemented using blockchain techniques).

Non-interactice Zero-Knowledge Proofs: We will need a NIZK of knowledge of a value $\alpha \in \mathbb{Z}_p$ such that $x = g^\alpha$ and $y = h^\alpha$ given g, x, h, y. For this we use the Fiat-Shamir transformation on the protocol of Chaum and Pedersen [10], which we denote by $\mathsf{DLEQ}(g, x, h, y)$. We will also need a simpler NIZK of knowledge of a value $\alpha \in \mathbb{Z}_p$ such that $x = g^\alpha$ given g, x. For this we use the Fiat-Shamir transformation on the protocol of Schnorr [24], which we denote by $\mathsf{DLOG}(g, x)$. We give a full description of these NIZKs in the full version [12].

A central component of our protocol is a zero-knowledge proof that an ordered set of ElGamal ciphertexts has been obtained by re-randomizing each ciphertext and permuting the resulting ciphertexts in a previous ordered set (an operation called a *Shuffle*). Formally, we want to prove knowledge of a permutation $\pi \in \Sigma_N$ and randomness $\mathbf{r} = (r_1, \ldots, r_N)$ such that for the vectors of ciphertexts $\mathbf{c} = (c_1, \ldots, c_N)$ and $\mathbf{c'} = (c'_1, \ldots, c'_N)$ we have $c'_i = \mathsf{TEG.ReRand}(c_{\pi(i)}, r_i)$. An efficient zero-knowledge argument for correctness of this kind of shuffle has been proposed in [4] and it can be turned into the required zero-knowledge proof through the Fiat-Shamir heuristic [17,22]. We denote this NIZK by $\mathsf{ZKSH}(\pi, \mathbf{r}, \mathbf{c}, \mathbf{c'})$ and refer interested readers to [4] for details on its construction and proof. Further discussion of this NIZK's efficiency and distributed generation of setup parameters is presented in the full version [12].

(n,n)-Threshold ElGamal Cryptosystem: A cryptosystem with (t,n)-threshold allow a group of n parties to jointly generate a public key that is then used to encrypt plaintext messages in such a way that they can only be recovered from the ciphertexts if at least t parties cooperate [13]. In our card deck generation procedure we employ a (n,n)-threshold version of the ElGamal cryptosystem [16] based on the constructions of [14,21] with a verifiable decryption protocol similar to the Verifiable Threshold Masking Functions (VTMF) of [3]. The final goal is to encode card information as Threshold ElGamal ciphertexts as in the VTMF based construction of [3]. However, we employ different techniques for distributed key generation in order to address the security issues we have identified in [3]. Moreover, we do not require the verifiable masking and verifiable re-masking (*rerandomization*) operations because the verification that these ciphertexts are correctly re-randomized is handled by the zero-knowledge proofs of correctness of a shuffle [4] presented in the next section. We do use the fact that this scheme is additively homomorphic (and rerandomizable) and a verifiable decryption procedure, where it is possible to verify that each user is providing a valid decryption share. We now present the (n,n)-Threshold ElGamal cryptosystem with verifiable decryption TEG and refer interested readers to [3,14,21] for a full discussion:

- **Key Generation** $\mathsf{TEG.Gen}(1^\lambda)$ **:** Each party $\mathcal{P}_i$ generates a random secret-key share $\mathsf{TEG}.sk_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts $h_i = g^{\mathsf{TEG}.sk_i}$ along with a proof $\mathsf{DLOG}(g, h_i)$[1]. Once all n parties have broadcast their public key share h_i, each party $\mathcal{P}_i$ verifies the accompanying proofs $\mathsf{DLOG}(g, h_j)$ (aborting if invalid) and then saves all h_j, for $i \neq j$, reconstructing the public key by computing $\mathsf{TEG}.pk = h = \prod_{i=1}^{n} h_i = g^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}$.
- **Encryption** $\mathsf{TEG.Enc}_{\mathsf{TEG}.pk}(m, r)$ **:** The encryption of a message $m \in \mathbb{G}$ under a public-key $\mathsf{TEG}.pk$ with randomness $r \in \mathbb{Z}_p$ is carried out as a regular ElGamal encryption. Namely, a ciphertext $c = (c_1 = g^r, c_2 = h^r m)$ is generated.
- **Re-Randomization** $\mathsf{TEG.ReRand}(c, r')$ **:** For fresh randomness r', a ciphertext $c = (c_1, c_2)$ is re-randomized by computing $c' = (g^{r'} c_1, h^{r'} c_2)$.
- **Verifiable Decryption** $\mathsf{TEG.Dec}_{\mathsf{TEG}.sk_1,\ldots,\mathsf{TEG}.sk_n}(c)$**:** Parse $c = (c_1, c_2)$. Each party $\mathcal{P}_i$ broadcast a decryption share $d_i = c_1^{\mathsf{TEG}.sk_i}$ and a proof $\mathsf{DLEQ}(g, h_i, c_1, d_i)$ showing that they have correctly used their secret-key share $\mathsf{TEG}.sk_i$. Once all n parties have broadcast their decryption share d_i, each party $\mathcal{P}_i$ checks that the $\mathsf{DLEQ}(g, h_j, c_1, d_j)$ proofs are correct for all $i \neq j$ (aborting otherwise) and retrieves the message by computing

$$\frac{c_2}{\prod_{i=1}^{n} d_i} = \frac{c_2}{c_1^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = \frac{m \cdot \mathsf{TEG}.pk^r}{g^{r \sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = \frac{m\left(g^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}\right)^r}{g^{r \sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = m.$$

Smart Contracts: The concept of smart contracts was introduced by Szabo [27] and recently popularized by the Ethereum plaftorm [8,31], which implements

[1] This zero-knowledge proof of the knowledge of the exponent solves the issue in [3] that was pointed out in the introduction.

smart contracts based on blockchain techniques. Basically, smart contracts allow a user to specify much richer conditions for transactions to be approved over a cryptocurrency scheme, mimicking contracts in real life. Besides ensuring that an amount of money is paid to a certain party who manages to fulfill a given static set of conditions, smart contracts can also maintain an evolving *state* that is taken into consideration when evaluating conditions for contract fulfillment.

In Ethereum, smart contracts can be written using Solidity, a Turing complete programming language specially designed for this purpose. In order to avoid denial-of-service attacks, the amount of computation involved in verifying fulfillment of a contract is bounded by how much a user is willing to pay have the contract checked. This payment is made by means of an auxiliary cryptocurrency called *gas*, which is given to the miners who verify a contract. Basically, more complex contracts require larger amounts of gas to be verified so that the miners receive compensation for computationally heavy contract verification.

The use of Ethereum based stateful contracts for rewards/penalties enforcement in secure multiparty computation protocols was first proposed in [7]. Their approach consists in having parties provide a deposit of a certain number of coins before protocol execution, later receiving a refund in case they behave honestly. Our protocol follows the same approach and consists mainly of operations over a cyclic group (where the Discrete Logarithm and DDH assumptions are believed to be hard). It has been estimated in [23] that a modular exponentiation over such a group (computed as a scalar multiplication over an elliptic curve) costs 40000 gas (0.075 US Dollars) while [7] estimated the DLEQ NIZK [10] to cost 1287858 gas (0.30 US Dollars) assuming an exponentiation cost of 300000 gas. Such estimates provide good evidence that our protocol could be implemented in a smart contract platform such as Ethereum at a reasonable price.

3 Poker Protocol

For an overview of the poker game and the game formalization using the ideal functionality $\mathcal{F}_{\mathsf{poker}}$, check the full version of this work [12].

Our protocol represents cards as ciphertexts of a threshold ElGamal cryptosystem, similarly to the scheme of [3], but employs different techniques for distributed key generation in order to address the security issues we have identified in [3] and a highly improved procedure for shuffling cards based on recent advances in zero-knowledge proofs of shuffle correctness [4]. In order to generate the representation of a shuffled deck, the parties first run a distributed key generation algorithm to obtain a public-key (while each holds a share of the secret-key). Next, they start a shuffling procedure that involves rerandomizing and the randomly permuting ciphertexts that encrypt the numbers assigned to each card (1 to 52), which is executed by all parties in a round-robin manner. The parties also provide to each other proofs that the shuffling was correctly executed, meaning that the resulting ciphertexts are indeed rerandomized and permuted version of ciphertexts provided by the previous party, which prevents adversaries from injecting ciphertexts representing arbitrary cards. When cards

are intended to be revealed publicly, each party broadcasts a decryption share of the ciphertext representing the card along with a zero-knowledge proof showing its correctness. If a covered card is to be given to one specific party, each of the other parties sends their decryption shares and proofs directly to that party through a private channel. The main efficiency improvement in our protocol is obtained by employing an a compact zero-knowledge proof of a shuffle introduced in [4] (made non-interactive by the Fiat-Shamir transform), instead of the cut-and-choose technique employed by [3]. This proof is compatible with ElGamal ciphertexts and achieves the same security level of the one in [3] with only a fraction of the computational and communication complexities.

The main new feature of our protocols is a mechanism for detecting and (financially) punishing cheaters without requiring the whole protocol to the executed on chain. This mechanism requires that the parties first deposit of a number of coins for "collateral", *i.e.* they lose these coins if they are detected as cheaters or abort. The protocol execution has a series of *checkpoints* where parties cooperate to generate a witness that the execution was correct up to that point. The witness is a signature by all parties agreeing on the current state of the execution. If at any point a protocol malfunction occurs (a party either does not receive a message or receives a invalid message), the party who detected it posts a complaint to the blockchain along with the last checkpoint witness and the protocol messages generated after the checkpoint. All the other are required to do the same or face punishment otherwise. This procedure verifies the current state of the protocol and then the execution continues in the blockchain until the next checkpoint. Any misbehavior or abort in this on-chain execution is punished financially. After the protocol execution reaches the next checkpoint and the parties obtain the corresponding witnesses, the protocol is again executed off-chain.

Smart Contract Functionality $\mathcal{F}_{\mathsf{SC}}$: Our poker protocol π_{Poker} makes use of a stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$, described in Fig. 1, that models blockchain transactions used to keep collateral deposits and enforce punishment of players who misbehave, as well as ensuring that winners get their rewards. It is important to emphasize that the $\mathcal{F}_{\mathsf{SC}}$ functionality can be easily implemented via smart contracts over a blockchain. More formally, using a public available *ledger*. Moreover, our construction (for protocol π_{Poker}) requires only simple operations, *i.e.*, verification of signatures and discrete logarithm operations over cyclic groups. The regular operation of our protocol is performed entirely off-chain, without intervention of the contract. However in the event that any problem happen or in the case that any participant in the game claim problems in the execution, any player can publish their agreed status of the game in the chain, via short witnesses (to be detailed in the protocol description).

Protocol π_{Poker}: The protocol is executed by n players $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ interacting with the stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$, and is parametrized by the small sb and big bb blind bets amount, the initial stake t, the maximum bet m per hand, the security deposit d and a timeout limit τ. In addition to the stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$, the other setup assumption is the random oracle

Functionality $\mathcal{F}_{\mathsf{SC}}$

The functionality is executed with n players $\mathcal{P}_1, \ldots, \mathcal{P}_n$. It is parametrized by the small sb and big bb blind bets amount, the initial stake t, the maximum bet m per hand, the security deposit d, the compensation amount q, a protocol verification mechanism pv and a timeout limit τ.

Players Check-in: Wait to receive from each player $\mathcal{P}_i$ a message $(\text{CHECKIN}, \mathcal{P}_i, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ containing the necessary coins, its signature verification key, its share of the threshold ElGamal public-key and the zero-knowledge proof of knowledge of the secret-key's share. Record the values and send $(\text{CHECKEDIN}, \mathcal{P}_i, \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ to all players. Allow the players to dropout and reclaim their coins if a player fails to check-in within the timeout limit τ. Once all check-ins are done, order the players by picking a random permutation and announce the ordered sequence of players by $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ to them. Mark all players as active.

Player Check-out: Upon receiving $(\text{CHECKOUT}, \mathsf{active}, \mathsf{balance}, \sigma)$ from $\mathcal{P}_i$, verify that σ contains valid signatures by all active players on active and $\mathsf{balance}$ and that $\mathsf{active}[i] = 0$. If everything is correct, for $w = \mathsf{balance}[i] + d$, send $(\text{PAYOUT}, \mathsf{coins}(w))$ to $\mathcal{P}_i$ and mark him as inactive. Send $(\text{CHECKEDOUT}, i, w)$ to the other players.

Recovery: Upon receiving a recovery request $(\text{REPORT}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ from $\mathcal{P}_i$ containing some checkpoint witnesses and current phase witnesses, send to each $\mathcal{P}_j \neq \mathcal{P}_i$ $(\text{REQUEST}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$. Upon getting $(\text{RESPONSE}, \mathcal{P}_j, \mathsf{Checkpoint}_j, \mathsf{CurrPhase}_j)$ from some player $\mathcal{P}_j$ with checkpoint and phase witnesses (which are not necessarily relative to the same checkpoint as received from other players) or an acknowledgement of previous submitted witnesses, forward this information to the other parties. Upon getting replies from all players or reaching the timeout limit τ, determine the current phase by verifying the most recent checkpoint that has valid witnesses. Verify the last valid point of the protocol execution using the current phase witnesses and pv. If there exists some $\mathcal{P}_i$ who sent misbehaving messages (together with a signature) in the current phase, then for each $\mathcal{P}_j \neq \mathcal{P}_i$ who has not checked-out, send $(\text{COMPENSATION}, \mathsf{coins}(d + q + \mathsf{balance}[j] + \mathsf{bets}[j]))$ to him. Send any leftover coins after the compensation for $\mathcal{P}_i$ and halt. Otherwise, mediate the execution of the protocol until the next checkpoint. This is done by using $(\text{NXT-STP}, \mathsf{phase}, \mathsf{round})$ to request an action from the next party that is supposed to act and using pv to verify the answer $(\text{NXT-STP-RSP}, \mathsf{msg}_{\mathsf{phase},\mathsf{round}})$. All messages are delivered to all players. If during this mediated execution a player misbehaves or does not answer within the timeout limit τ, penalize him and compensate the others as above, and halt. Otherwise send $(\text{RECOVERED}, \mathsf{phase}, \mathsf{Checkpoint})$ to the parties once the next checkpoint is reached.

Fig. 1. The stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$.

model. We assume that the parties agree on a generator g of a group $\mathbb{G}$ of order p for the (n, n)-Threshold ElGamal cryptosystem TEG and also on a $\mathsf{EUF\text{-}CMA}$ secure digital signature scheme SIG. Moreover, a *nonce* unique to each protocol execution and protocol round (*e.g.* a hash of the public protocol transcript up to the current round) is implicitly attached to every signed message to avoid replay attacks. The protocol proceeds in *phases* as described below:

- **Recovery Triggers:** Whenever a signature or NIZK proof is received, its validity is tested. If the test fails, the party proceeds to the recovery phase. The same happens if a party does not receive an expected message until a timeout limit τ. These triggers will be omitted henceforth.
- **Players Check-in:** For $i = 1, \ldots, n$, player $\mathcal{P}_i$ proceeds as follows:
  1. generates the keys of the signature scheme $(\mathsf{SIG}.vk_i, \mathsf{SIG}.sk_i) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\lambda)$.
  2. generates TEG's key shares by sampling $\mathsf{TEG}.sk_i \xleftarrow{\$} \mathbb{Z}_p$, setting $h_i = g^{\mathsf{TEG}.sk_i}$ and generating a proof $\mathsf{DLOG}(g, h_i)$.
  3. sends $(\text{CHECKIN}, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ to $\mathcal{F}_{\mathsf{SC}}$ and waits until getting from $\mathcal{F}_{\mathsf{SC}}$ the check-in confirmation $(\text{CHECKEDIN}, \mathcal{P}_j, \mathsf{SIG}.vk_j, h_j, \mathsf{DLOG}(g, h_j))$ of each player and the parties' order $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ that is used henceforth in the protocol. If not received until the timeout limit τ, contact $\mathcal{F}_{\mathsf{SC}}$ to dropout and reclaim the deposited coins.
  4. verifies each $\mathsf{DLOG}(g, h_j)$ for $j \neq i$, reconstructs the initial public key $\mathsf{TEG}.pk = \prod_{j=1}^{n} h_j$, record all h_j, and initializes a vector $\mathsf{balance} = (t, \ldots, t)$, a vector $\mathsf{bets} = (0, \ldots, 0)$, a counter $psb = 1$ and a counter $pbb = 2$.

- **Hand Execution - Shuffle:** As the first step in executing a hand, the parties generate a randomly shuffled deck of closed cards $\mathsf{c}_1, \ldots, \mathsf{c}_{52}$. For $i = 1, \ldots, n$, $\mathcal{P}_i$ proceeds as follows (w.l.o.g. we assume all parties are active, the adaptation to the other cases is the straightforward one):
  1. If $\mathcal{P}_i = \mathcal{P}_1$, it sets $\mathbf{c}^0 = (\mathsf{c}_1^0, \ldots, \mathsf{c}_{52}^0)$ where $\mathsf{c}_j^0 = \mathsf{TEG.Enc}_{\mathsf{TEG}.pk}(j, 1)$. Otherwise, $\mathcal{P}_i$ considers the cards $\mathbf{c}^{i-1} = (\mathsf{c}_1^{i-1}, \ldots, \mathsf{c}_{52}^{i-1})$ received from $\mathcal{P}_{i-1}$. Notice that these initial ciphertexts just encrypt the number of each card (in increasing order) under deterministic randomness 1, allowing $\mathcal{P}_2$ to locally compute the initial set of ciphertexts for verification.
  2. $\mathcal{P}_i$ samples uniformly at random a permutation $\pi \in \Sigma_{52}$ and $\mathbf{r} = (r_1, \ldots, r_{52})$ where $r_j \xleftarrow{\$} \mathbb{Z}_p$, and sets $\mathsf{c}_j^i = \mathsf{TEG.ReRand}_{\mathsf{TEG}.pk}(\mathsf{c}_{\pi(j)}^{i-1}, r_j)$, obtaining a new set $\mathbf{c}^i = (\mathsf{c}_1^i, \ldots, \mathsf{c}_{52}^i)$. Notice that this new set of ciphertexts representing cards simply contains rerandomized versions of the previous ciphertexts in a random order.
  3. $\mathcal{P}_i$ generates a zero-knowledge proof of correctness of shuffle $\mathsf{ZKSH}(\pi, \mathbf{r}, \mathbf{c}^{i-1}, \mathbf{c}^i)$ and broadcasts it with the shuffled deck $\mathbf{c}^i$. All other parties verify this zero-knowledge proof.

After all parties have participated in the shuffling procedure, the shuffled deck for the current hand is set to be $\mathcal{D} = \mathbf{c}^n$. All parties sign it by computing $\sigma_{\mathcal{D}}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk}(\texttt{DECK} - \texttt{READY}, \mathcal{D})$, broadcasts $\sigma_{\mathcal{D}}^i$ and verifies all signatures. *Checkpoint Witness:* The previous checkpoint witness concatenated with the deck $\mathcal{D}$ and corresponding signatures $\sigma_{\mathcal{D}}^i$.

- **Hand Execution - Small and Big Blinds:** After the shuffle is done, all parties wait for the small blind, *i.e.* for $\mathcal{P}_{psb}$ to broadcast a signature

$\sigma_{sb}^{psb} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_{psb}}(\mathsf{SB})$ as well as signatures on vectors $\mathsf{balance}$ and bets, where $\mathsf{balance}[psb]$ is decreased by sb coins, $\mathsf{bets}[psb]$ is increased by sb coins, while all other coordinates remain the same. Upon receiving the signatures, each party $\mathcal{P}_i$ broadcasts a signature $\sigma_{sb}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathsf{SB})$ as well as signatures on $\mathsf{balance}$ and bets. All signatures are verified. Proceed analogously for the big blind. *Checkpoint Witness:* The previous checkpoint witness with the updated $\mathsf{balance}$ and bets (and signatures on them) concatenated with all signatures σ_{sb}^i and σ_{bb}^i.

- **Hand Execution - Drawing Cards and Private Cards Distribution:** Two private cards $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ for each active party $\mathcal{P}_i$ as well as the community cards $\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3, \mathsf{cc}_4, \mathsf{cc}_5$ are drawn from $\mathcal{D}$ according to the rules of poker. For $i = 1, \ldots, n$, $\mathcal{P}_i$ proceeds as follows to open cards $\mathsf{pc}_{j,1}, \mathsf{pc}_{j,2}$ towards $\mathcal{P}_j$ for $j = 1, \ldots, n$ and to obtain its own private cards (here all parties act in parallel):
  1. $\mathcal{P}_i$ computes its decryption shares for $\mathsf{pc}_{j,1}, \mathsf{pc}_{j,2}$ by parsing $\mathsf{pc}_{j,k}$ as $(c_{j,k,1}, c_{j,k,2})$ and computing $d_{j,k,i} = c_{j,k,1}^{\mathsf{TEG}.sk_i}$ and a NIZK $\mathsf{DLEQ}(g, h_i, c_{j,k,1}, d_{j,k,i})$ for $k \in \{1, 2\}$. $\mathcal{P}_i$ sends the decryption shares $d_{j,1,i}, d_{j,2,i}$ along with their corresponding proofs to $\mathcal{P}_j$ through a private channel.
  2. Once it has received all $d_{i,1,j}, d_{i,2,j}$ and corresponding DLEQ proofs from the other parties, $\mathcal{P}_i$ checks that the proofs are valid. Finally, $\mathcal{P}_i$ learns its private cards by computing $\mathsf{pc}'_{i,k} = \frac{c_{i,k,2}}{\prod_{i=1}^{n} d_{i,k,j}}$ for $k \in \{1, 2\}$.
  3. $\mathcal{P}_i$ broadcasts $\sigma_{pc}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathtt{PRIVATE-CARDS})$ after retrieving its private cards. Remember the signature implicitly includes a nonce unique to this protocol execution and specific round. Once signatures σ_{pc}^j from all parties have been received, verify them.

Checkpoint Witness: The previous checkpoint witness, except for the signatures σ_{sb}^i and σ_{bb}^i, concatenated with all σ_{pc}^i.

- **Hand Execution - Main Flow:** After cards are drawn and private cards are distributed, all parties proceed to the main flow of playing a hand, where a number of community cards will be opened and a number of betting rounds will be played, both according to the community card opening and betting round procedures. All parties continue the main flow by proceeding as follows:
  - Execute a betting round starting with the closest active successor of $\mathcal{P}_{pbb}$.
  - Execute a community card opening procedure for flop cards $\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3$.
  - Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.
  - Execute a community card opening procedure for turn card cc_4.
  - Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.
  - Execute a community card opening procedure for river card cc_5.
  - Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.

- Proceed to showdown starting with the last player who increased the bet in the last round, if there is one; otherwise, the closest active successor of $\mathcal{P}_{psb-1}$.

– **Community Card Opening:** In the steps of π_{Poker} where a community card $\mathsf{cc} \in \{\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3, \mathsf{cc}_4, \mathsf{cc}_5\}$ has to be opened, party $\mathcal{P}_i$, for $i = 1, \ldots, n$, proceeds as follows:
  1. $\mathcal{P}_i$ parses $\mathsf{cc} = (cc_1, cc_2)$ and broadcasts its decryption shares $d_i = cc_1^{\mathsf{TEG}.sk_i}$ along with a NIZK $\mathsf{DLEQ}(g, h_i, cc_1, d_i)$.
  2. After all decryption shares d_j and corresponding DLEQ NIZKs are received from all parties, $\mathcal{P}_i$ verifies if all NIZKs are valid. $\mathcal{P}_i$ opens cc by computing $\frac{cc_2}{\prod_{i=1}^{n} d_j}$.
  3. After opening cc, $\mathcal{P}_i$ broadcasts $\sigma^i_{\mathsf{cc}} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk}(\mathtt{COMMUNITY-OPEN}, \mathsf{cc})$ in order to communicate it has successfully opened cc. Once all signatures σ^j_{cc} from other parties have been received, $\mathcal{P}_i$ verifies that they are all valid.

  Checkpoint Witness: The previous one together with all signatures σ^i_{cc}.

– **Betting Round:** In the steps of π_{Poker} that require a betting round starting from party $\mathcal{P}_s$, each party $\mathcal{P}_i$ communicates its betting action $\text{ACTION}_i \in \{\text{FOLD}, \text{CALL}, (\text{RAISE}, r), \text{ALL-IN}, \text{CHECK}\}$ (as defined in $\mathcal{F}_{\mathsf{poker}}$) in a round robin manner starting from $\mathcal{P}_s$ and following the order $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ received from $\mathcal{F}_{\mathsf{SC}}$, proceeding as follows until the conditions specified in $\mathcal{F}_{\mathsf{poker}}$ for finishing the betting round are met:
  - When it is $\mathcal{P}_i$'s turn to state its bet, $\mathcal{P}_i$ updates vectors bets and $\mathsf{balance}$ according to its action ACTION_i, *i.e.* it increases (resp. decreases) $\mathsf{bets}[i]$ (resp. $\mathsf{balance}[i]$) by the amount of coins required by ACTION_i as defined in $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{P}_i$ generates a signature $\sigma^i_{bet} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\text{ACTION}_i, \mathsf{bets}[i], \mathsf{balance}[i])$ and broadcasts $(\text{ACTION}_i, \mathsf{bets}[i], \mathsf{balance}[i], \sigma^i_{bet})$.
  - Upon receiving $(\text{ACTION}_j, \mathsf{bets}[j], \mathsf{balance}[j], \sigma^j_{bet})$ from party $\mathcal{P}_j$ for $j \neq i$, $\mathcal{P}_i$ checks the validity of σ^j_{bet}. Next, $\mathcal{P}_i$ verifies that $\mathsf{bets}[j]$ and $\mathsf{balance}[j]$ are consistent with ACTION_j according to the rules defined in $\mathcal{F}_{\mathsf{poker}}$. If not, $\mathcal{P}_i$ proceeds to the recovery phase. If both checks succeed, $\mathcal{P}_i$ updates its local copy of bets and $\mathsf{balance}$ with the new values of $\mathsf{bets}[j]$ and $\mathsf{balance}[j]$, and proceeds in the betting round.

When the conditions for ending the betting round specified in $\mathcal{F}_{\mathsf{poker}}$ are met, each party $\mathcal{P}_i$ broadcasts a signature $\sigma^i_{betstate} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathsf{bets}, \mathsf{balance})$ on its local copy of vectors bets and $\mathsf{balance}$. $\mathcal{P}_i$ waits until all signatures $\sigma^j_{betstate}$ are received from every other party $\mathcal{P}_j$ for $j \neq i$ and verifies that they are valid signatures on their local vectors bets and $\mathsf{balance}$ (verifying that all parties agree on the final bets and $\mathsf{balance}$). *Checkpoint Witness:* The previous checkpoint witness with the updated vectors bets and $\mathsf{balance}$, along with all signatures $\sigma^i_{betstate}$ on the updated vectors.

- **Showdown:** The parties proceed in a round-robin way. If a party $\mathcal{P}_i$ wishes to open its private cards $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ during showdown, $\mathcal{P}_i$ broadcasts the decryption shares $d_{i,1,j}, d_{i,2,j}$ along with their corresponding DLEQ proofs, for $j = 1, \ldots, n$. For every party $\mathcal{P}_i$ who opens its private cards during showdown, the other parties $\mathcal{P}_j$ decrypt $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ by following the same procedure used for reconstructing their own private cards. If decryption fails, $\mathcal{P}_j$ proceed to the recovery phase. If a party $\mathcal{P}_i$ wishes to muck during showdown, it broadcasts a signature $\sigma^i_{muck} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{MUCK})$, the other parties verify the signature. Once all parties have either opened or mucked, the parties proceed to the pot distribution.
- **Pot Distribution:** Each party $\mathcal{P}_i$ uses the opened cards, chronological order of folded/mucked hands and current vectors $\mathsf{balance}$ and bets to locally compute the updated $\mathsf{balance}$ for all parties according to the rules of poker. It also zeros out bets. $\mathcal{P}_i$ broadcast signatures on $\mathsf{balance}$ and bets. Upon receiving these values from each party $\mathcal{P}_j$, $\mathcal{P}_i$ verifies that it is a valid signature on its own local updated vectors $\mathsf{balance}$ and bets. A party $\mathcal{P}_i$ who wishes to continue playing broadcasts a signature $\sigma^i_{cont} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{CONTINUE})$. A party $\mathcal{P}_i$ who no longer wishes to play or who has $\mathsf{balance}[i] = 0$ broadcasts a signature $\sigma^i_{chko} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_j}(\texttt{CHECKOUT})$. Each party $\mathcal{P}_i$ checks that all other parties' signatures are valid. For all parties $\mathcal{P}_j$ who choose to check-out, mark party $\mathcal{P}_j$ as inactive. After determining which parties remain active and which check out, each party $\mathcal{P}_i$ constructs a vector active such that $\mathsf{active}[j] = 1$ if party $\mathcal{P}_j$ is active in the next hand or $\mathsf{active}[j] = 0$ if $\mathcal{P}_j$ is checking out. $\mathcal{P}_i$ broadcasts a signature $\sigma^i_{act} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathsf{active})$. $\mathcal{P}_i$ checks that signatures σ^j_{act} by all other parties $\mathcal{P}_j$ are valid signatures on the same active vector, otherwise it proceeds to the recovery phase. If there were check-outs, update the public key as $\mathsf{TEG}.pk = \prod_{j=1 \text{ s.t. j is active}}^{n} h_j$. Increment *psb* and *pbb* using the order among the active players. A signature on these values are also generated by each party and checked by the others. *Checkpoint Witness:* Vectors $\mathsf{balance}$, bets and active, counters *psb* and *pbb*, as well as all signatures on these values.
- **Player Check-out:** If $\mathcal{P}_i$ was marked as checking out in the pot distribution phase, it sends a message $(\text{CHECKOUT}, \mathsf{active}, \mathsf{balance}, \sigma)$ to $\mathcal{F}_{\mathsf{SC}}$, where σ contains all signatures on active and $\mathsf{balance}$, waits for confirmation from $\mathcal{F}_{\mathsf{SC}}$ and stops execution.
- **Recovery Request:** If a party $\mathcal{P}_i$ enters the recovery phase at any step of a given phase, it sends a message $(\text{REPORT}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ to $\mathcal{F}_{\mathsf{SC}}$, where $\mathsf{Checkpoint}_i$ is the checkpoint witness from the previous phase and $\mathsf{CurrPhase}_i$ is the transcript of the current phase so far (*i.e.* only the messages that received and sent by $\mathcal{P}_i$ after the last checkpoint).
- **Responding to a Recovery Request:** Upon receiving a message $(\text{REQUEST}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ from $\mathcal{F}_{\mathsf{SC}}$ containing the checkpoint witness and current phase transcript included in the REPORT message of $\mathcal{P}_i$, every other party $\mathcal{P}_j$ sends a message $(\text{RESPONSE}, \mathcal{P}_j, \mathsf{Checkpoint}_j, \mathsf{CurrPhase}_j)$ to $\mathcal{F}_{\mathsf{SC}}$ containing their own most recent checkpoint witness and transcript of the current phase if they are different from the ones already

submitted by other parties. Otherwise, it simply acknowledges the one that is equal. Once all parties have responded to the recovery request, all parties have learned each other checkpoint witnesses and the transcripts of the current phase. For $i = 1, \ldots, n$, party $\mathcal{P}_i$ proceeds as follows:

- Upon receiving the message (NXT-STP, phase, round) from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ computes its message $\mathsf{msg}_{\mathsf{phase,round}}$ for the round specified by round of the phase specified by phase and sends (NXT-STP-RSP, $\mathsf{msg}_{\mathsf{phase,round}}$) to $\mathcal{F}_{\mathsf{SC}}$ following the protocol.
- Upon receiving (RECOVERED, phase, Checkpoint) from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ records the checkpoint witness of the phase specified by phase and returns to the regular execution of next phase as described in the protocol by communicating directly to the other parties.

The security of Protocol π_{Poker} is captured in the following theorem whose proof is presented in the full version of this work [12] due to space limitations.

Theorem 1. *Assuming that the DDH problem is hard and that the digital signature scheme* SIG *is EUF-CMA secure, protocol* π_{Poker} *securely computes* $\mathcal{F}_{\mathsf{poker}}$ *in the* $\mathcal{F}_{\mathsf{SC}}$*-hybrid, random oracle model in the presence of malicious static adversaries.*

4 Concrete Complexity Analysis

In this section we analyze the concrete communication and computational complexities of π_{Poker}. We estimate (off-chain) communication and computational complexities for the case where no user cheats (thus never triggering the recovery phase). The exact cost of performing recovery will depend on the exact point of the protocol where the recovery request happened, since the players are required to post their protocol messages generated in each round after the latest checkpoint witness. Nevertheless, we discuss why our on-chain space complexity is generally low given that we explicitly define compact witnesses for intermediate step of the protocol (even inside poker rounds). On the other hand, previous works in [20] and [7] only mention (but not define) intermediate witnesses for each round of the poker game. Moreover, we exclude the cost of generating and sending the messages between the parties and $\mathcal{F}_{\mathsf{SC}}$, since these messages are basically transactions being posted in the blockchain and their size and generation cost may vary depending on the concrete implementation.

Estimating Complexity: We estimate computational complexity in terms of the number of exponentiations that each party has to perform in each phase of the protocol. On the other hand, we estimate communication complexity in terms of the total number of group (*i.e.* $\mathbb{G}$) elements and ring (*i.e.* $\mathbb{Z}_p$) elements transferred by all parties in each phase of the protocol. Most of the messages exchanged in the protocol are broadcast to all parties[2]. However, during private cards distribution, decryption shares for each card are sent directly to its

[2] We remark that, in our scenario, broadcasts can achieved by having parties communicate directly with each other due to the low number of parties (typically $n \leq 10$).

owner through a private channel. We denote messages transmitted through private channels by [private] and messages broadcast through public channels by [broadcast]. Messages that are not explicitly marked are assumed to be broadcast by public channels. Both the Betting Round and Showdown phases have complexities that fully depend on the behavior of each player in the game of poker and other conditions such as the stake of the game. For example, a user can choose to keep raising his bet in a Betting Round and users can choose whether to show their cards or muck in Showdown. Those choices are perfectly honest and permitted in the game but they result in different final complexities for these phases of π_{Poker}. In the case of the Betting Round phase, we estimate the complexity for the case where all players speak once, which can be easily used to compute the complexity in cases where each player speaks multiple times. In the case of the Showdown phase, we estimate the complexity for the worst case (in terms of complexity), where all players choose to show their cards.

Instantiating the Building Blocks: In this analysis we instantiate ZKSH (NIZK of correctness of a shuffle) with parameters $k = 4$ and $l = 13$, which results in 208 exponentiations for the prover and 208 exponentiations for the verifier, with a proof size of 44 elements of $\mathbb{G}$ and 65 elements of $\mathbb{Z}_p$. Notice that this estimation is actually an upper bound for concrete communication complexity, since it pertains to the interactive version of ZKSH, which is significantly improved in terms of concrete communication complexity after applying the Fiat-Shamir heuristic. We instantiate the signature scheme SIG with the ECDSA scheme [19], where a public key consists of a elliptic curve point (that we count as an element of $\mathbb{G}$) and a signature consists of two scalars (we count as elements of $\mathbb{Z}_p$). The ECDSA scheme requires one elliptic curve point multiplication by a scalar for generating a key pair, one for signing and two for signature verification (without optimizations), which we count as group exponentiations since π_{Poker} is written in terms of groups with multiplicative notation. The concrete communication and computational complexities of are presented in Table 1.

On-Chain Space Complexity: Considering that players act honestly throughout the protocol, information is only stored in the blockchain when a player wishes to redeem its rewards. In this case, the player must post a witness showing that all players agree that the protocol was correctly executed. This witness consists of a simple digital signature. In case a malicious player does cheat and an honest player triggers the recovery mechanism, players are required to post to the blockchain their latest checkpoint witness (if they disagree with the witnesses posted by other players) and the protocol messages generated after that witness. Notice that this checkpoint witness is also a simple digital signature and that the bulk of the data posted on the blockchain actually depends on which phase of the protocol is currently being executed. For example, if recovery is triggered during the Main Flow phase of Hand Execution, only the latest checkpoint witness and short messages required in that phase would have to be posted to the blockchain, excluding the long messages previously sent in the Shuffle and Drawing Cards phase. On the other hand, previous protocols in [20]

Table 1. Concrete communication and computational complexities of π_{Poker} in terms of number of exponentiations executed per player and number of elements of $\mathbb{G}$ and $\mathbb{Z}_p$ transmitted by all players in total for each phase with n players. During private cards distribution, some messages are sent through a private channel, which we denote by [private]. All the other messages in the protocol are broadcast through public channels, which we denote by [broadcast]. Messages that are not explicitly marked are assumed to be broadcast by public channels.

Phase	Exponentiations (Per Player)	Communication (Total)	
		$\mathbb{G}$	$\mathbb{Z}_p$
Players Check-in	$2n + 1$	$2n$	$2n$
Hand Execution - Shuffle	$209n + 104$	$148n$	$67n$
Hand Execution - Blinds	$12n$	0	$24n$
Hand Execution - Drawing/Private Cards Distribution	$16n - 13$	$2(n^2 - n)$ [private]	$4(n^2 - n)$ [private], $2n$ [broadcast]
Hand Execution - Main Flow	$52n - 20$	$5n$	$28n$
Showdown (Worst Case)	$8(n-1)^2$	$2n^2$	$4n^2$
Pot Distribution	$2n$	0	$4n$
Total	$8n^2 + 271n + 82$	$2n^2 + 155n$ [broadcast], $2(n^2 - n)$ [private]	$4n^2 + 127n$ [broadcast], $4(n^2 - n)$ [private]

and [7] only mention that intermediate witnesses could be generated after a full round of poker, incurring in a much higher overhead in terms of blockchain storage when recovery happens. Moreover, such witnesses are not explicitly defined in [20] and [7].

Comparison with Previous Protocols: While we present estimated computational and communication complexities for each phase of a *complete poker game*, previous works only focus on individual card operations [3,9,25,28,29,32,33], making it hard to provide direct comparisons to our results. In order to provide a meaningful comparison, we will focus on the card shuffling phase, which is the main bottleneck of poker protocols. Considering a deck of 52 cards (necessary for a poker game) and a security parameter $k = 40$ for the cut-and-choose step (which is the lowest security parameter used for this kind of technique in modern cryptography), the protocol of [29] (used as a building block in [7]) requires $2120n$ exponentiations per player in the Shuffle phase where there are n players. With the same parameters, the Shuffle phase of the protocol proposed in [3] requires $6240(n-1) + 8320$ exponentiations, where n is the number of players. On the other hand, our protocol only requires $209n + 104$ exponentiations per player as detailed in Table 1, resulting in improvements of an order of (at least) 10 times.

5 Conclusion

We introduced the first specific purpose protocol for secure poker with payment distribution and penalty enforcement with fully-simulatable security. In order to argue about our protocol's security, we introduced the first formal simulation based security notions for such protocols, overlooked by previous works. Moreover, we identified concrete flaws in previously proposed protocols [3,9], showcasing the need for formal security definitions and proofs. Our work improves on previous heuristic approaches for constructing poker protocols and provides a more efficient alternative to general results that provide payment distribution and penalty enforcement for general MPC protocols, where generality comes at the cost of efficiency.

References

1. Ahmed, M.: How UK beat the odds to win at online gambling (2017). https://www.ft.com/content/044a3d9e-7d1a-11e7-9108-edda0bcbc928. Accessed 29 Aug 2017
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 443–458. IEEE Computer Society Press, May 2014
3. Barnett, A., Smart, N.P.: Mental poker revisited. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898, pp. 370–383. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40974-8_29
4. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_17
5. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 1993, pp. 62–73. ACM Press, November 1993
6. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
7. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 410–440. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_15
8. Buterin, V.: White paper (2013). https://github.com/ethereum/wiki/wiki/White-Paper. Accessed 5 Dec 2017
9. Castellà-Roca, J., Sebé, F., Domingo-Ferrer, J.: Dropout-tolerant TTP-free mental poker. In: Katsikas, S., López, J., Pernul, G. (eds.) TrustBus 2005. LNCS, vol. 3592, pp. 30–40. Springer, Heidelberg (2005). https://doi.org/10.1007/11537878_4
10. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7
11. Crépeau, C.: A zero-knowledge Poker protocol that achieves confidentiality of the players' strategy *or* how to achieve an electronic Poker face. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 239–247. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_18

12. David, B., Dowsley, R., Larangeira, M.: Kaleidoscope: an efficient poker protocol with payment distribution and penalty enforcement. Cryptology ePrint Archive, Report 2017/899 (2017). https://eprint.iacr.org/2017/899
13. Desmedt, Y.: Society and group oriented cryptography: a new concept. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_8
14. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_28
15. The Economist: A Big Deal (2007). http://www.economist.com/node/10281315#print. Accessed 24 Aug 2017
16. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_2
17. Fiat, A., Shamir, A.: How To prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
18. IMDb: Kaleidoscope (2017). http://www.imdb.com/title/tt0060581/. Accessed 12 Sept 2017
19. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
20. Kumaresan, R., Moran, T., Bentov, I.: How to use bitcoin to play decentralized poker. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 195–206. ACM Press, October 2015
21. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47
22. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_33
23. Reitwiessner, C.: EIP 196 (2017). https://github.com/ethereum/EIPs/blob/master/EIPS/eip-196.md. Accessed 13 Dec 2017
24. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
25. Sebe, F., Domingo-Ferrer, J., Castella-Roca, J.: On the security of a repaired mental poker protocol. In: Third International Conference on Information Technology: New Generations, pp. 664–668 (2006)
26. Shamir, A., Rivest, R.L., Adleman, L.M.: Mental poker. In: Klarner, D.A. (ed.) The Mathematical Gardner, pp. 37–43. Springer, Heidelberg (1981). https://doi.org/10.1007/978-1-4684-6686-7_5
27. Szabo, N.: Smart contracts: building blocks for digital markets (1996). http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Accessed 5 Dec 2017
28. Wei, T.J.: Secure and practical constant round mental poker. Inf. Sci. **273**, 352–386 (2014)
29. Wei, T.J., Wang, L.C.: A fast mental poker protocol. J. Math. Cryptol. **6**(1), 39–68 (2012)
30. Wikipedia: Online Poker (2017). https://en.wikipedia.org/wiki/Online_poker. Accessed 29 Aug 2017

31. Wood, G.: Ethereum: a secure decentralized transaction ledger (2014). http://gavwood.com/paper.pdf. Accessed 5 Dec 2017
32. Zhao, W., Varadharajan, V.: Efficient TTP-free mental poker protocols. In: ITCC 2005 - Volume II, vol. 1, pp. 745–750, April 2005
33. Zhao, W., Varadharajan, V., Mu, Y.: A secure mental poker protocol over the internet. In: ACSW Frontiers 2003, pp. 105–109. Australian Computer Society Inc., Darlinghurst (2003)

Blockchain Modeling

Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach

Anastasia Mavridou[1(✉)] and Aron Laszka[2]

[1] Vanderbilt University, Nashville, USA
anastasia.mavridou@vanderbilt.edu
[2] University of Houston, Houston, USA

Abstract. The adoption of blockchain-based distributed computation platforms is growing fast. Some of these platforms, such as Ethereum, provide support for implementing smart contracts, which are envisioned to have novel applications in a broad range of areas, including finance and the Internet-of-Things. However, a significant number of smart contracts deployed in practice suffer from security vulnerabilities, which enable malicious users to steal assets from a contract or to cause damage. Vulnerabilities present a serious issue since contracts may handle financial assets of considerable value, and contract bugs are non-fixable by design. To help developers create more secure smart contracts, we introduce *FSolidM*, a framework rooted in rigorous semantics for designing contracts as Finite State Machines (FSM). We present a tool for creating FSM on an easy-to-use graphical interface and for automatically generating Ethereum contracts. Further, we introduce a set of design patterns, which we implement as plugins that developers can easily add to their contracts to enhance security and functionality.

Keywords: Smart contract · Security · Finite state machine
Ethereum · Solidity · Automatic code generation · Design patterns

1 Introduction

The adoption and importance of blockchain based distributed ledgers are growing fast. For example, the market capitalization of Bitcoin, the most-popular cryptocurrency, has exceeded $70 billion in 2017.[1] While the first generation of blockchain systems were designed to provide only cryptocurrencies, later systems, such as Ethereum, can also function as distributed computing platforms [1,2]. These distributed and trustworthy platforms enable the implementation smart contracts, which can automatically execute or enforce their contractual terms [3]. Beyond financial applications, blockchains are envisioned to

[1] https://coinmarketcap.com/currencies/bitcoin/.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 523–540, 2018.
https://doi.org/10.1007/978-3-662-58387-6_28

have a wide range of applications, such as asset tracking for the Internet-of-Things [4]. Due to their unique advantages, blockchain based platforms and smart contracts are embraced by an increasing number of organizations and companies. For instance, the project HyperLedger[2], which aims to develop open-source blockchain tools, is backed by major technology companies and financial firms, such as IBM, Cisco, J.P. Morgan, and Wells Fargo [5].

At the same time, smart contracts deployed in practice are riddled with bugs and security vulnerabilities. A recent automated analysis of 19,336 smart contracts deployed on the public Ethereum blockchain found that 8,333 contracts suffer from at least one security issue [6]. While not all of these issues lead to security vulnerabilities, many of them enable cyber-criminals to steal digital assets, such as cryptocurrencies. For example, the perpetrator(s) of the infamous 2016 "The DAO" attack exploited a combination of vulnerabilities to steal 3.6 million Ethers, which was worth around $50 million at the time of the attack [7]. More recently, $31 million worth of Ether was stolen due to a critical security flaw in a digital wallet contract [8]. Furthermore, malicious attackers might be able to cause damage even without stealing any assets, e.g., by leading a smart contract into a deadlocked state, which does not allow the rightful owners to spend or withdraw their assets.

Security vulnerabilities in smart contracts present a serious issue for multiple reasons. Firstly, smart contracts deployed in practice handle financial assets of significant value. For example, at the time of writing, the combined value held by Ethereum contracts deployed on the public blockchain is 12,205,760 Ethers, which is worth more than $3 billion.[3] Secondly, smart-contract bugs cannot be patched. By design, once a contract is deployed, its functionality cannot be altered even by its creator. Finally, once a faulty or malicious transaction is recorded, it cannot be removed from the blockchain ("code is law" principle [9]). The only way to roll back a transaction is by performing a hard fork of the blockchain, which requires consensus among the stakeholders and undermines the trustworthiness of the platform [10].

In practice, these vulnerabilities often arise due to the semantic gap between the assumptions contract writers make about the underlying execution semantics and the actual semantics of smart contracts [6]. Prior work focused on addressing these issues in existing contracts by providing tools for verifying correctness [9] and for identifying common vulnerabilities [6]. In this paper, we explore a different avenue by proposing and implementing *FSolidM*, a novel framework for creating secure smart contracts:

- We introduce a formal, finite-state machine (FSM) based model for smart contracts. We designed our model primarily to support Ethereum smart contracts, but it may be applied on other platforms as well.
- We provide an easy-to-use graphical editor that enables developers to design smart contracts as FSMs.

[2] https://www.hyperledger.org/.
[3] https://etherscan.io/accounts/c.

– We provide a tool for translating FSMs into Solidity code.[4]
– We provide a set of plugins that implement security features and design patterns, which developers can easily add to their model.

Our tool is open-source and available online (see Sect. 6 for details).

The advantages of our approach, which aims to help developers create secure contracts rather than to fix existing ones, are threefold. First, we provide a formal model with clear semantics and an easy-to-use graphical editor, thereby decreasing the semantic gap and eliminating the issues arising from it. Second, rooting the whole process in rigorous semantics allows the connection of our framework to formal analysis tools [11,12]. Finally, our code generator—coupled with the plugins provided in our tool—enables developers to implement smart contracts with minimal amount of error-prone manual coding.

The remainder of this paper is organized as follows. In Sect. 2, we give a brief overview of related work on smart contracts and common vulnerabilities. In Sect. 3, we first present blind auction as a motivating example problem, which can be implemented as a smart contract, and then introduce our finite-state machine based contract model. In Sect. 4, we describe our FSM-to-Solidity code transformation. In Sect. 5, we introduce plugins that extend the contract model with additional functionality and security features. In Sect. 6, we describe our FSolidM tool and provide numerical results on computational cost. Finally, in Sect. 7, we offer concluding remarks and outline future work.

2 Related Work

2.1 Common Vulnerabilities and Design Patterns

Multiple studies investigate and provide taxonomies for common security vulnerabilities and design patterns in Ethereum smart contracts. In Table 1, we list the vulnerabilities that we address and the patterns that we implement in our framework using plugins.

Table 1. Common smart-contract vulnerabilities and design patterns

Type	Common name	FSolidM plugin
Vulnerabilities	Reentrancy [6,13]	Locking (Sect. 5.1)
	Transaction ordering [6] a.k.a. unpredictable state [13]	Transition counter (Sect. 5.2)
Patterns	Time constraint [14]	Timed transitions (Sect. 5.3)
	Authorization [14]	Access control (Sect. 5.4)

[4] Solidity is the most widely used high-level language for developing Ethereum contracts. Solidity code can be translated into Ethereum Virtual Machine bytecode, which can be deployed and executed on the platform.

Atzei et al. provide a detailed taxonomy of security vulnerabilities in Ethereum smart contracts, identifying twelve distinct types [13]. For nine vulnerability types, they show how an attacker could exploit the vulnerability to steal assets or to cause damage. Luu et al. discuss four of these vulnerability types in more detail, proposing various techniques for mitigating them (see Sect. 2.2) [6]. In this paper, we focus on two types of these common vulnerabilities:

- Reentrancy Vulnerability: Reentrancy is one of the most well-known vulnerabilities, which was also exploited in the infamous "The DAO" attack. In Ethereum, when a contract calls a function in another contract, the caller has to wait for the call to finish. This allows the callee, who may be malicious, to take advantage of the intermediate state in which the caller is, e.g., by invoking a function in the caller.
- Transaction-Ordering Dependence: If multiple users invoke functions in the same contract, the order in which these calls are executed cannot be predicted. Consequently, the users have uncertain knowledge of the state in which the contract will be when their individual calls are executed.

Bartoletti and Pompianu identify nine common design patterns in Ethereum smart contracts, and measure how many contracts use these patterns in practice [14]. Their results show that the two most common patterns are *authorization* and *time constraint*, which are used in 61% and 33% of all contracts, respectively. The also provide a taxonomy of Bitcoin and Ethereum contracts, dividing them into five categories based on their application domain. Based on their categorization, they find that the most common Ethereum contracts deployed in practice are financial, notary, and games.

2.2 Verification and Automated Vulnerability Discovery

Multiple research efforts attempt to identify and fix these vulnerabilities through verification and vulnerability discovery. For example, Hirai first performs a formal verification of a smart contract that is used by the Ethereum Name Service [15].[5] However, this verification proves only one particular property and it involves relatively large amount of manual analysis. In later work, Hirai defines the complete instruction set of the Ethereum Virtual Machine in Lem, a language that can be compiled for interactive theorem provers [16]. Using this definition, certain safety properties can be proven for existing contracts.

Bhargavan et al. outline a framework for analyzing and verifying the safety and correctness of Ethereum smart contracts [9]. The framework is built on tools for translating Solidity and Ethereum Virtual Machine bytecode contracts into F^*, a functional programming language aimed at program verification. Using the F^* representations, the framework can verify the correctness of the Solidity-to-bytecode compilation as well as detect certain vulnerable patterns.

[5] The Ethereum Name Service is a decentralized service, built on smart contracts, for addressing resources using human-readable names.

Luu et al. propose two approaches for mitigating common vulnerabilities in smart contracts [6]. First, they recommend changes to the execution semantics of Ethereum, which eliminate vulnerabilities from the four classes that they identify in their paper. However, these changes would need to be adopted by all Ethereum clients. As a solution that does not require changing Ethereum, they provide a tool called Oyente, which can analyze smart contracts and detect certain security vulnerabilities.

Fröwis and Böhme define a heuristic indicator of control flow immutability to quantify the prevalence of contractual loopholes based on modifying the control flow of Ethereum contracts [17]. Based on an evaluation of all the contracts deployed on Ethereum, they find that two out of five contracts require trust in at least one third party.

3 Defining Smart Contracts as FSMs

Let us consider a blind auction (similar to the one presented in [18]), in which a bidder does not send her actual bid but only a hashed version of it. The bidder is also required to make a deposit—which does not need to be equal to her actual bid—to prevent the bidder from not sending the money after she has won the auction. A deposit is considered valid if its value is higher than or equal to the actual bid. We consider that a blind auction has four main *states*:

1. `AcceptingBlindedBids`, in which blind bids and deposits are accepted by the contract;
2. `RevealingBids`, in which bidders reveal their bids, i.e., they send their actual bids and the contract checks whether the hash value is the same as the one provided during the `AcceptingBlindedBids` state and whether sufficient deposit has been provided;
3. `Finished`, in which the highest bid wins the auction. Bidders can withdraw their deposits except for the winner, who can withdraw only the difference between her deposit and bid;
4. `Canceled`, in which bidders can retract bids and withdraw their deposits.

Our approach relies on the following observations. Smart contracts have *states* (e.g., `AcceptingBlindedBids`, `RevealingBids`). Furthermore, contracts provide functions that allow other entities (e.g., contracts or users) to invoke *actions* and change the state of the smart contracts. Thus, smart contracts can be naturally represented by FSMs [19]. An FSM has a finite set of states and a finite set of transitions between these states. A transition forces a contract to take a set of actions if the associated conditions, which are called the *guards* of the transition, are satisfied. Since such states and transitions have intuitive meaning for developers, representing contracts as FSMs provides an adequate level of abstraction for reasoning about their behavior.

Figure 1 presents the blind auction example in the form of an FSM. For simplicity, we have abbreviated `AcceptingBlindedBids`, `RevealingBids`, `Finished`, and `Canceled` to `ABB`, `RB`, `F`, and `C`, respectively. `ABB` is the initial

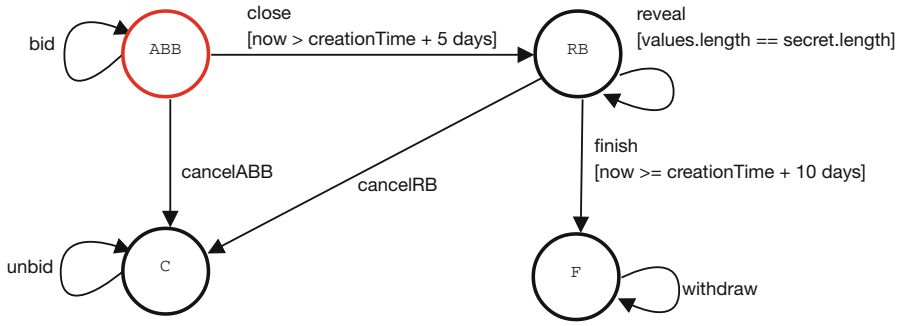


Fig. 1. Example FSM for blinded auctions.

state of the FSM. Each transition (e.g., `bid`, `reveal`, `cancel`) is associated to a set of actions that a user can perform during the blind auction. For instance, a bidder can execute the `bid` transition at the `ABB` state to send a blind bid and a deposit value. Similarly, a user can execute the `close` transition, which signals the end of the bidding period, if the associated guard `now >= creationTime + 5 days` evaluates to true. To differentiate transition names from guards, we use square brackets for the latter. A bidder can reveal her bids by executing the `reveal` transition. The `finish` transition signals the completion of the auction, while the `cancelABB` and `cancelRB` transitions signal the cancellation of the auction. Finally, the `unbid` and `withdraw` transitions can be executed by the bidders to withdraw their deposits. For ease of presentation, we omit from Fig. 1 the actions that correspond to each transition. For instance, during the execution of the `withdraw` transition, the following action is performed `amount = pendingReturns[msg.sender]`.

Guards are based on a set of variables, e.g., `creationTime`, `values`, and actions are also based on a set of variables, e.g., `amount`. These variable sets store data, that can be of type:

- *contract data*, which is stored within the contract;
- *input data*, which is received as transition input;
- *output data*, which is returned as transition output.

We denote by C, I, and O the three sets of the contract, input, and output variables of a smart contract. We additionally denote:

$\mathbb{B}[C, I]$, the set of Boolean predicates on contract and input variables;
$\mathbb{E}[C, I, O]$, the set of statements that can be defined by the full Solidity syntax.

Notice that $\mathbb{E}[C, I, O]$ represents the set of actions of all transitions. Next, we formally define a contract as an FSM.

Definition 1. *A Smart Contract is a tuple* $(S, s_0, C, I, O, \rightarrow)$*, where:*

- S *is a finite set of states;*

- $s_0 \in S$ *is the initial state;*
- C, I, *and* O *are disjoint finite sets of, respectively, contract, input, and output variables;*
- $\rightarrow \subseteq S \times \mathcal{G} \times \mathcal{F} \times S$ *is a transition relation, where:*
  - $\mathcal{G} = \mathbb{B}[C, I]$ *is a set of guards;*
  - $\mathcal{F}$ *is a set of action sets, i.e., a set of all ordered powersets of* $\mathbb{E}[C, I, O]$.

4 FSM-to-Solidity Transformation

To automatically generate a contract using our framework, developers can provide the corresponding FSM in a graphical form. Each transition of the FSM is implemented as a Solidity function, where an element of $\mathcal{G}$ and a list of statements from $\mathcal{F}$ form the body. The input I and output C variables correspond to the arguments and the `return` values, respectively, of these functions. In this section, we describe the basic transformation formally, while in Sect. 5, we present a set of extensions, which we call *plugins*.

First, let us list the input that must be provided by the developer:

- *name*: name of the FSM;
- S: set of states;
- $s_0 \in S$: initial state;
- C: set of contract variables;
- for each contract variable $c \in C$, $access(c) \in \{\texttt{public}, \texttt{private}\}$: visibility of the variable;
- $\rightarrow$: set of transitions;
- for each transition $t \in \rightarrow$:
  - t^{name}: name of the transition;
  - $t^{guards} \in \mathcal{G}$: guard conditions of the transition;
  - $t^{input} \subseteq I$: input variables (i.e., parameters) of the transition;
  - $t^{statements} \in \mathcal{F}$: statements of the transition;
  - $t^{output} \subseteq O$: output (i.e., return values) of the transition;
  - $t^{from} \in S$: previous state;
  - $t^{to} \in S$: next state;
  - $t^{tags} \subseteq \{payable, admin, event\}$: set of transition properties specified by the developer (note that without plugins, only *payable* is supported);
- $\mathcal{T}^{custom}$: set of complex types defined by the developer in the form of `structs`.

For any variable $v \in C \cup I \cup O$, we let $type(v) \in \mathcal{T}$ denote the domain of the variable, where $\mathcal{T}$ denotes the set of all built-in Solidity types and developer-defined `struct` types.

We use `fixed-width` font for the output generated by the transformation, and *italic* font for elements that are replaced with input or specified later. An FSM is transformed into a Solidity contract as follows:

$$
\begin{aligned}
\mathit{Contract} ::= \ & \texttt{contract}\ \mathit{name}\ \{ \\
& \quad \mathit{StatesDefinition} \\
& \quad \texttt{uint private creationTime = now;} \\
& \quad \mathit{VariablesDefinition} \\
& \quad \mathit{Plugins} \\
& \quad \mathit{Transition}(t_1) \\
& \quad \ldots \\
& \quad \mathit{Transition}(t_{|\rightarrow|}) \\
& \}
\end{aligned}
$$

where $\{t_1, \ldots, t_{|\rightarrow|}\} = \ \rightarrow$. Without any security extensions or design patterns added (see Sect. 5), *Plugins* is empty. The complete generated code for the blind-auction example presented in Fig. 1 can be found in [20] (with the locking and transition-counter security-extension plugins added).

$$
\begin{aligned}
\mathit{StatesDefinition} ::= \ & \texttt{enum States}\ \{s_0, \ldots, s_{|S|-1}\} \\
& \texttt{States private state = States.}s_0\texttt{;}
\end{aligned}
$$

where $\{s_0, \ldots, s_{|S|-1}\} = S$.

Example 1. The following snippet of Solidity code presents the *StatesDefinition* code generated for the blind auction example (see Fig. 1).

```
enum States {
    ABB,
    RB,
    F,
    C
}
States private state = States.ABB;
```

$$
\begin{aligned}
\mathit{VariablesDefinition} ::= \ & \mathcal{T}^{custom} \\
& \mathit{type}(c_1)\ \mathit{access}(c_1)\ c_1\texttt{;} \\
& \ldots \\
& \mathit{type}(c_{|C|})\ \mathit{access}(c_{|C|})\ c_{|C|}\texttt{;}
\end{aligned}
$$

where $\{c_1, \ldots, c_{|C|}\} = C$.

Example 2. The following snippet of Solidity code presents the *VariablesDefinition* code of the blind auction example (see Fig. 1).

```
struct Bid {
    bytes32 blindedBid;
    uint deposit;
}
mapping(address => Bid[]) private bids;
mapping(address => uint) private pendingReturns;
address private highestBidder;
uint private highestBid;
```

$$
\begin{aligned}
\mathit{Transition}(t) ::= \ & \texttt{function } t^{name}\texttt{(}\mathit{type}(i_1)\ i_1\texttt{, } \ldots\texttt{, } \mathit{type}(i_{|t^{input}|})\ i_{|t^{input}|}\texttt{)} \\
& \quad \mathit{TransitionPlugins}(t) \\
& \quad \mathit{Payable}(t)\ \mathit{Returns}(t)\ \texttt{\{} \\
& \texttt{require(state == States.}t^{from}\texttt{);} \\
& \mathit{Guards}(t) \\
& \mathit{Statements}(t) \\
& \texttt{state = States.}t^{to}\texttt{;} \\
& \texttt{\}}
\end{aligned}
$$

where $\{i_1, \ldots, i_{|t^{input}|}\} = t^{input}$. Without any security extensions or design patterns (see Sect. 5), $\mathit{TransitionPlugins}(t)$ is empty, similar to *Plugins*. If $payable \in t^{tags}$, then $\mathit{Payable}(t) =$ `payable`; otherwise, it is empty. If $t^{to} = t^{from}$ then the line `state = States.`t^{to}`;` is not generated.

If $t^{output} = \emptyset$, then $\mathit{Returns}(t)$ is empty. Otherwise, it is as follows:

$$
\mathit{Returns}(t) ::= \texttt{returns (}\mathit{type}(o_1)\ o_1\texttt{, } \ldots\texttt{, } \mathit{type}(o_{|t^{output}|})\ o_{|t^{output}|}\texttt{)}
$$

where $\{o_1, \ldots, o_{|t^{output}|}\} = t^{output}$.

Further,

$$
\begin{aligned}
\mathit{Guards}(t) ::= \ & \texttt{require((}g_1\texttt{) \&\& (}g_2\texttt{) \&\& } \ldots \texttt{ \&\& (}g_{|t^{guards}|}\texttt{));} \\
\mathit{Statements}(t) ::= \ & a_1 \\
& \ldots \\
& a_{|t^{statements}|}
\end{aligned}
$$

where $\{g_1, \ldots, g_{|t^{guards}|}\} = t^{guards}$ and $\{a_1, \ldots, a_{|t^{statements}|}\} = t^{statements}$.

Example 3. The following snippet of Solidity code shows the generated `bid` transition (see Fig. 1). The `bid` transition does not have any guards and the state of the FSM does not change, i.e., it remains `ABB` after the execution of the transition.

```
// Transition bid
function bid(bytes32 blindedBid)
  payable
{
    require(state == States.ABB);
    //Actions
    bids[msg.sender].push(Bid({
        blindedBid: blindedBid,
        deposit: msg.value
    }));
}
```

Example 4. The following snippet of Solidity code shows the generated `close` transition (see Fig. 1). The `close` transition does not have any associated actions but the state of the FSM changes from `ABB` to `RB` after the execution of the transition.

```
// Transition close
function close()
{
    require(state == States.ABB);
    //Guards
    require(now >= creationTime + 5 days);
    //State change
    state = States.RB;
}
```

5 Security Extensions and Patterns

Building on the FSM model and the FSM-to-Solidity transformation introduced in the previous sections, we next provide extensions and patterns for enhancing the security and functionality of contracts. These extensions and patterns are implemented as plugins, which are appended to the *Plugins* and *TransitionPlugins* elements. Developers can easily add plugins to a contract (or some of its transitions) using our tool, without writing code manually.[6]

5.1 Locking

To prevent reentrancy vulnerabilities, we provide a security plugin for locking the smart contract.[7] The locking feature eliminates reentrancy vulnerabilities in a "foolproof" manner: functions within the contract cannot be nested within each other in any way.

Implementation. If the locking plugin is enabled, then

$$\textit{Plugins} \mathrel{+}= \texttt{bool private locked = false;}$$
$$\texttt{modifier locking \{}$$
$$\texttt{require(!locked);}$$
$$\texttt{locked = true;}$$
$$\texttt{\_;}$$
$$\texttt{locked = false;}$$
$$\texttt{\}}$$

and for every transition t,

$$\textit{TransitionPlugins}(t) \mathrel{+}= \texttt{locking}$$

Before a transition is executed, the `locking` modifier first checks if the contract is locked. If it is not locked, then the modifier locks it, executes the transition, and unlocks it after the transition has finished. Note that the `locking` plugin must be applied before the other plugins so that it can prevent reentrancy vulnerabilities in the other plugins. Our tool always applies plugins in the correct order.

[6] Please note that we introduce an additional plugin in Appendix A.

[7] http://solidity.readthedocs.io/en/develop/contracts.html?highlight=mutex#function-modifiers.

5.2 Transition Counter

Recall from Sect. 2.1 that the state and the values of the variables stored in an Ethereum contract may be unpredictable: when a user invokes a function (i.e., transition in an FSM), she cannot be sure that the contract does not change in some way before the function is actually executed. This issue has been referred to as "transaction-ordering dependence" [6] and "unpredictable state" [13], and it can lead to various security issues. Furthermore, it is rather difficult to prevent since multiple users may invoke functions at the same time, and these function invocations might be executed in any order.

We provide a plugin that can prevent unpredictable-state vulnerabilities by enforcing a strict ordering on function executions. The plugin expects a transition number in every function as a parameter (i.e., as a transition input variable) and ensures that the number is incremented by one for each function execution. As a result, when a user invokes a function with the next transition number in sequence, she can be sure that the function is executed before any other state changes can take place (or that the function is not executed).

Implementation. If the transition counter plugin is enabled, then

Plugins +=
```
uint private transitionCounter = 0;
modifier transitionCounting(uint nextTransitionNumber) {
  require(nextTransitionNumber == transitionCounter);
  transitionCounter += 1;
  _;
}
```

and for every transition t,

TransitionPlugins(t) += `transitionCounting(nextTransitionNumber)`

Note that due to the inclusion of the above modifier, t^{input}—and hence the parameter list of every function implementing a transition— includes the parameter `nextTransitionNumber` of type `uint`.

5.3 Automatic Timed Transitions

Next, we provide a plugin for implementing time-constraint patterns. We first need to extend our FSM model: a Smart Contract with Timed Transitions is a tuple $C = (S, s_0, C, I, O, \rightarrow, \overset{T}{\rightarrow})$, where $\overset{T}{\rightarrow} \subseteq S \times \mathcal{G}_T \times \mathbb{N} \times \mathcal{F}_T \times S$ is a timed transition relation such that:

- $\mathcal{G}_T = \mathbb{B}[C]$ is a set of guards (without any input data);
- $\mathbb{N}$ is the set of natural numbers, which is used to specify the time of the transition in seconds;

– $\mathcal{F}_T$ is a set of action sets, i.e., a set of all ordered powerset of $\mathbb{E}[C]$.

Notice that timed transitions are similar to non-timed transitions, but (1) their guards and assignments do not use input or output data and (2) they include a number specifying the transition time.

We implement timed transitions as a modifier that is applied to every function. When a transition is invoked, the modifier checks whether any timed transitions must be executed before the invoked transition is executed. If so, the modifier executes the timed transitions before the invoked transition.

Writing such modifiers for automatic timed transitions manually may lead to vulnerabilities. For example, a developer might forget to add a modifier to a function, which enables malicious users to invoke functions without the contract progressing to the correct state (e.g., place bids in an auction even though the auction should have already been closed due to a time limit).

Implementation. For every timed transition $tt \in \xrightarrow{T}$, the developer specifies a time $tt^{time} \in \mathbb{N}$ at which the transition will automatically happen (given that the guard condition is met). This time is measured in the number of seconds elapsed since the creation (i.e., instantiation) of the contract. We let $tt_1, tt_2, \ldots$ denote the list of timed transitions in ascending order based on their specified times. When the plugin is enabled,

$$
\begin{aligned}
\mathit{Plugins} \mathrel{+}= \texttt{modifier timedTransitions } \{ \\
\quad \mathit{TimedTransition}(tt_1) \\
\quad \mathit{TimedTransition}(tt_2) \\
\quad \ldots \\
\quad \texttt{\_;} \\
\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathit{TimedTransition}(t) ::= \texttt{if ((state == States.}t^{from}\texttt{)} \\
\quad \texttt{\&\& (now >= creationTime + }t^{time}\texttt{)} \\
\quad \texttt{\&\& (}\mathit{Guard}(t)\texttt{)) \{} \\
\quad \mathit{Statements}(t) \\
\quad \texttt{state = States.}t^{to}\texttt{;} \\
\}
\end{aligned}
$$

Finally, for every non-timed transition $t \in \rightarrow$, let

$$
\mathit{TransitionPlugins}(t) \mathrel{+}= \texttt{timedTransitions}
$$

5.4 Access Control

In many contracts, access to certain transitions (i.e., functions) needs to be controlled and restricted.[8] For example, any user can participate in a typical blind auction by submitting a bid, but only the creator should be able to cancel the auction. To facilitate the enforcement of such constraints, we provide a plugin that (1) manages a list of administrators at runtime (identified by their addresses) and (2) enables developers to forbid non-administrators from accessing certain functions. This plugin implements management functions (`addAdmin`, `removeAdmin`) for only one privileged group, but it could easily be extended to support more fine-grained access control.

Implementation. If the access control plugin is enabled, then

Plugins +=
```
mapping(address => bool) private isAdmin;
uint private numAdmins = 1;

function name() {
  isAdmin[msg.sender] = true;
}

modifier onlyAdmin {
  require(isAdmin[msg.sender]);
  _;
}

function addAdmin(address admin) onlyAdmin {
  require(!isAdmin[admin]);
  isAdmin[admin] = true;
  numAdmins += 1;
}

function removeAdmin(address admin) onlyAdmin {
  require(isAdmin[admin]);
  require(numAdmins > 1);
  isAdmin[admin] = false;
  numAdmins -= 1;
}
```

[8] http://solidity.readthedocs.io/en/develop/common-patterns.html#restricting-access.

For transitions t such that $admin \in t^{tags}$ (i.e., transitions that are tagged "only admin" by the developer),

$$TransitionPlugins(t) \mathrel{+}= \texttt{onlyAdmin}$$

6 The FSolidM Tool

We present the FSolidM tool, which is build on top of WebGME [21], a web-based, collaborative, versioned, model editing framework. FSolidM enables collaboration between multiple users during the development of smart contracts. Changes in FSolidM are committed and versioned, which enables branching, merging, and viewing the history of a contract. FSolidM is open-source[9] and available online[10].

To use FSolidM, a developer must provide some input (see Sect. 4). To do so, the developer can use the graphical editor of FSolidM to specify the states, transitions, guards, etc. of a contract. The full input of the smart-contract code generator can be defined entirely through the FSolidM graphical editor. For the convenience of the developers, we have also implemented a Solidity code editor, since part of the input e.g., variable definitions and function statements, might be easier to directly write in a code editor. Figure 2 shows the two editors of the tool. We have integrated a Solidity parser[11] to check the syntax of the Solidity code that is given as input by the developers.

The FSolidM code editor cannot be used to completely specify the required input. Notice that in Fig. 2, parts of the code shown in the code editor are darker (lines 1–10) than other parts (lines 12–15). The darker lines of code include code that was generated from the FSM model defined in the graphical editor and are locked—cannot be altered in the code editor. The non-dark parts indicate code that was directly specified in the code editor.

FSolidM provides mechanisms for checking if the FSM is correctly specified (e.g., whether an initial state exists or not). FSolidM notifies developers of errors and provides links to the erroneous nodes of the model (e.g., a transition or a guard). Additionally, FSolidM provides an FSM-to-Solidity code generator and mechanisms for easily integrating the plugins introduced in Sect. 5. We present the FSolidM tool in greater detail in [20].

6.1 Numerical Results on Computational Cost

Plugins not only enhance security but also increase the computational cost of transitions. Since users must pay a relatively high price for computation performed on the public Ethereum platform, the computational cost of plugins is a critical question. Here, we measure and compare the computational cost of transitions in our blind-auction contract without and with the locking and transition

[9] https://github.com/anmavrid/smart-contracts.
[10] https://cps-vo.org/group/SmartContracts.
[11] https://github.com/ConsenSys/solidity-parser.

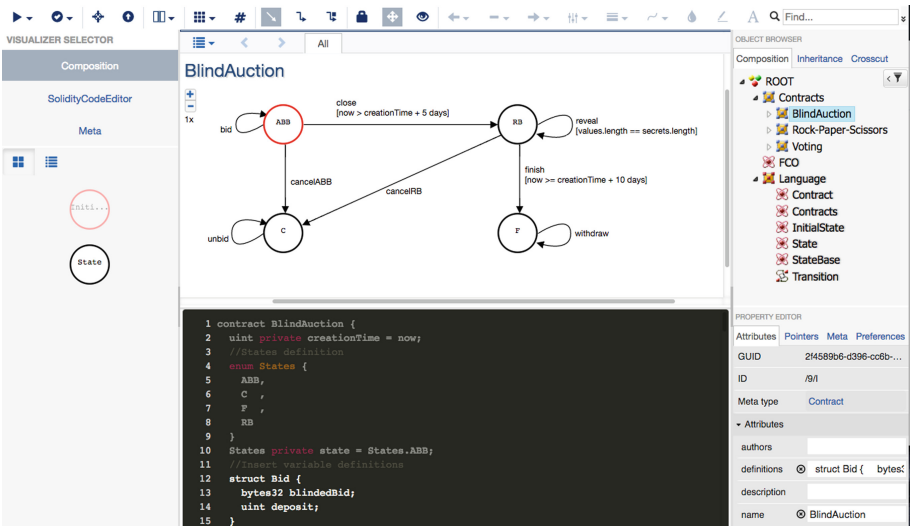


Fig. 2. The graphical and code editors provided by FSolidM.

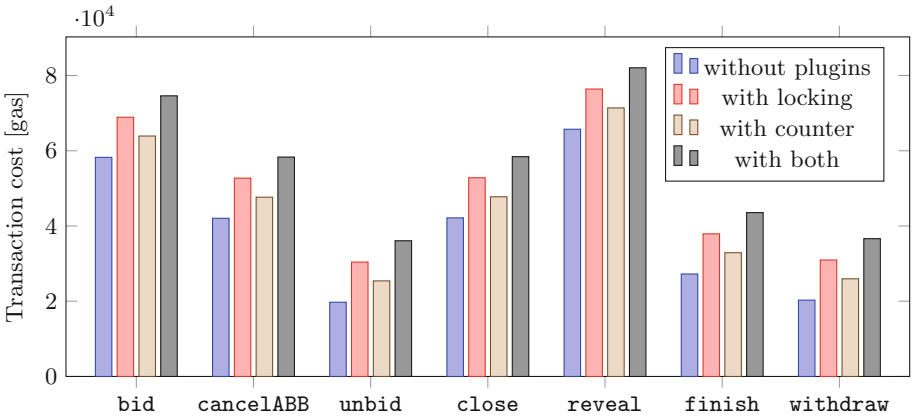


Fig. 3. Transaction costs in gas without plugins (**blue**), with locking plugin (**red**), with transition counter plugin (**brown**), and with both plugins (**dark gray**). (Color figure online)

counter plugins. We focus on these security feature plugins because they introduce overhead, while the design pattern plugins introduce useful functionality.

For this experiment, we use Solidity compiler version 0.4.17 with optimizations enabled. In all cases, we quantify computational cost of a transition as the gas cost of an Ethereum transaction that invokes the function implementing the transition.[12] The cost of deploying our smart contract was 504,672 gas without

[12] Gas measures the cost of executing computation on the Ethereum platform.

any plugins, 577,514 gas with locking plugin, 562,800 gas with transition counter plugin, and 637,518 gas with both plugins.[13]

Figure 3 shows the gas cost of each transition for all four combinations of the two plugins. We make two key observations. First, *computational overhead is almost constant* for both plugins and also for their combination. For example, the computational overhead introduced by locking varies between 10,668 and 10,686 gas. For the simplest transition, `unbid`, this constitutes a 54% increase in computational cost, while for the most complex transition, `reveal`, the increase is 16%. Second, the *computational overhead of the two plugins is additive*. The increase in computational cost for enabling locking, transition counter, and both are around 10,672 gas, 5,648 gas, and 16,319 gas, respectively.

7 Conclusion and Future Work

Distributed computing platforms with smart-contract functionality are envisioned to have a significant technological and economic impact in the future. However, if we are to avoid an equally significant risk of security incidents, we must ensure that smart contracts are secure. While previous research efforts focused on identifying vulnerabilities in existing contracts, we explored a different avenue by proposing and implementing a novel framework for creating secure smart contracts. We introduced a formal, FSM based model of smart contracts. Based on this model, we implemented a graphical editor for designing contracts as FSMs and an automatic code generator. We also provided a set of plugins that developers can add to their contracts. Two of these plugins, *locking* and *transition counter*, implement security features for preventing common vulnerabilities (i.e., reentrancy and unpredictable state). The other two plugins, *automatic timed transitions* and *access control*, implement common design patterns to facilitate the development of correct contracts with complex functionality.

We plan to extend our framework in multiple directions. First, we will introduce a number of plugins, implementing various security features and design patterns. We will provide security plugins for all the vulnerability types identified in [13] that can be addressed on the level of Solidity code. We will also provide plugins implementing the most popular design patterns surveyed in [14].

Second, we will integrate verification tools [11,12] and correctness-by-design techniques [22] into our framework. This will enable developers to easily verify the security and safety properties of their contracts. For example, developers will be able to verify if a malicious user could lead a contract into a deadlocked state. Recall that deadlocks present a serious issue since it may be impossible to recover the functionality or assets of a deadlocked contract.

Third, we will enable developers to model and verify multiple interacting contracts as a set of interacting FSMs. By verifying multiple contracts together, developers will be able to identify a wider range of issues. For example, a set of interacting contracts may get stuck in a deadlock even if the individual contracts are deadlock free.

[13] At the time of writing, this cost of deployment was well below $1 (if the deployment does not need to be prioritized).

Acknowledgements. We thank the anonymous reviewers for their invaluable suggestions and feedback.

A Event Plugin

In this section, we introduce an additional plugin, which developers can use to notify users of transition executions. The *event plugin* uses the `event` feature of Solidity, which provides a convenient interface to the Ethereum logging facilities. If this plugin is enabled, transitions tagged with *event* emit a Solidity event after they are executed. Ethereum clients can listen to these events, allowing them to be notified when a tagged transition is executed on the platform.

Implementation. If the event plugin is enabled, then

$$\begin{aligned} \mathit{Plugins} \mathrel{+}= \; & \mathit{TransitionEvent}(t_1) \\ & \mathit{TransitionEvent}(t_2) \\ & \ldots \end{aligned}$$

where $\{t_1, t_2, \ldots\}$ is the set of transitions with the tag *event*.

$$\begin{aligned} \mathit{TransitionEvent}(t) ::= \; & \texttt{event Event}t^{name}\texttt{;} \\ & \texttt{modifier event}t^{name}\ \texttt{\{} \\ & \quad \texttt{\_;} \\ & \quad \texttt{Event}t^{name}\texttt{();} \\ & \texttt{\}} \end{aligned}$$

For every transition t such that $\mathit{event} \in t^{tags}$ (i.e., transitions that are tagged to emit an event),

$$\mathit{TransitionPlugins}(t) \mathrel{+}= \texttt{event}t^{name}$$

References

1. Underwood, S.: Blockchain beyond Bitcoin. Commun. ACM **59**(11), 15–17 (2016)
2. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Technical Report EIP-150, Ethereum Project - Yellow Paper, April 2014
3. Clack, C.D., Bakshi, V.A., Braine, L.: Smart contract templates: foundations, design landscape and research directions. arXiv preprint arXiv:1608.00771 (2016)
4. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the Internet of Things. IEEE Access **4**, 2292–2303 (2016)
5. Vukolić, M.: Rethinking permissioned blockchains. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, pp. 3–7. ACM (2017)
6. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 254–269. ACM, October 2016

7. Finley, K.: A $50 million hack just showed that the DAO was all too human. Wired, June 2016. https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/
8. Qureshi, H.: A hacker stole $31m of ether - how it happened, and what it means for Ethereum. freeCodeCamp, July 2017. https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce
9. Bhargavan, K., et al.: Short paper: formal verification of smart contracts. In: Proceedings of the 11th ACM Workshop on Programming Languages and Analysis for Security (PLAS), in Conjunction with ACM CCS 2016, pp. 91–96, October 2016
10. Leising, M.: The Ether thief. Bloomberg Markets, June 2017. https://www.bloomberg.com/features/2017-the-ether-thief/
11. Bensalem, S., Bozga, M., Nguyen, T.-H., Sifakis, J.: D-finder: a tool for compositional deadlock detection and verification. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 614–619. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_45
12. Cavada, R., et al.: The NUXMV symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 334–342. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_22
13. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8
14. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 494–509. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_31
15. Hirai, Y.: Formal verification of deed contract in Ethereum name service, November 2016. https://yoichihirai.com/deed.pdf
16. Hirai, Y.: Defining the Ethereum virtual machine for interactive theorem provers. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 520–535. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_33
17. Fröwis, M., Böhme, R.: In code we trust? In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) ESORICS/DPM/CBT-2017. LNCS, vol. 10436, pp. 357–372. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67816-0_20
18. Solidity by Example: Blind auction. http://solidity.readthedocs.io/en/develop/solidity-by-example.html. Accessed 9 May 2017
19. Solidity Documentation: Common patterns. http://solidity.readthedocs.io/en/develop/common-patterns.html#state-machine. Accessed 9 May 2017
20. Mavridou, A., Laszka, A.: Designing secure Ethereum smart contracts: a finite state machine based approach. arXiv preprint arXiv:1711.09327 (2017)
21. Maróti, M., et al.: Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. In: Proceedings of the MPM@ MoDELS, pp. 41–60 (2014)
22. Mavridou, A., Stachtiari, E., Bliudze, S., Ivanov, A., Katsaros, P., Sifakis, J.: Architecture-based design: a satellite on-board software case study. In: Kouchnarenko, O., Khosravi, R. (eds.) FACS 2016. LNCS, vol. 10231, pp. 260–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57666-4_16

A Formal Model of Bitcoin Transactions

Nicola Atzei[1], Massimo Bartoletti[1(✉)], Stefano Lande[1], and Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
bart@unica.it
[2] Università degli Studi di Trento, Trento, Italy

Abstract. We propose a formal model of Bitcoin transactions, which is sufficiently abstract to enable formal reasoning, and at the same time is concrete enough to serve as an alternative documentation to Bitcoin. We use our model to formally prove some well-formedness properties of the Bitcoin blockchain, for instance that each transaction can only be spent once. We release an open-source tool through which programmers can write transactions in our abstract model, and compile them into standard Bitcoin transactions.

1 Introduction

In recent years we have observed a growing interest around *cryptocurrencies*. Bitcoin [13], the first decentralized cryptocurrency, was introduced in 2009, and through the years it has consolidated its position as the most popular one. Bitcoin and other cryptocurrencies have pushed forward the concept of decentralization, providing means for reliable interactions between mutually distrusting parties on an open network.

The nodes of the Bitcoin network maintain a public and immutable data structure, called *blockchain*. The blockchain stores the historical record of all transfers of bitcoins, which are referred to as *transactions*. When a node updates the blockchain, the other nodes verify if the appended transactions are valid, e.g. by checking if the conditions specified in *scripts* are satisfied. Scripts are programmable boolean functions: in their standard (and mostly used) form they verify a digital signature against a public key. Since the blockchain is immutable, tampering with a stored transaction would result in the invalidation of all the subsequent ones. Updating the state of the blockchain, i.e. appending new transactions, requires solving a moderately difficult cryptographic puzzle. In case of conflicting updates, the chain that required the largest computational effort is considered the valid one. Hence, the immutability and the consistency of the blockchain is bounded by the total computational power of honest nodes. An adversary with enough resources can append invalid transactions, e.g. with incorrect digital signatures, or rewrite a part of the blockchain, e.g. to perform a *double-spending attack*. The attack consists in paying someone by publishing a transaction on the blockchain, and then removing it (making the funds unspent).

Besides the intended monetary application, the Bitcoin blockchain can be seen as a way to consistently maintain the state of a system over a peer-to-peer

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 541–560, 2018.
https://doi.org/10.1007/978-3-662-58387-6_29

network, without the need of a trusted authority. If the system is a currency, its state is the amount of funds in each account. This concept can be generalised to the case where the system is a *smart contract* [15], namely an executable computer protocol which can also handle transfers of currency. The idea of exploiting the Bitcoin blockchain to build smart contracts has recently been explored by several works. Lotteries [2,5,6,11], gambling games [10], contingent payments [4], covenants [12,14], and other kinds of fair computations [1,9] are some examples of the capabilities of Bitcoin as a platform for smart contracts.

Smart contracts often rely on features of Bitcoin that go beyond the standard transfers of currency. For instance, while the vast majority of Bitcoin transactions uses scripts only to verify signatures, smart contracts like the above-mentioned ones exploit more complex scripts, e.g. to determine the winner of a lottery, or to check if a secret has been revealed. Smart contracts may also exploit other (infrequently used) features of Bitcoin, e.g. various signature modifiers, and temporal constraints on transactions.

As a matter of fact, using these advanced features to design a new smart contract is not a trivial matter, for two reasons. First, while the overall behaviour of Bitcoin is clear, the details of many of its crucial aspects are poorly documented. To understand the details of how a mechanism actually works, one has to explore various web pages (often inaccurate, or inconsistent, or overly technical), and eventually resort to the source code of the Bitcoin client[1] to have the correct answer. Second, the description of advanced features is often too concrete to be effectively used in the design and analysis of a smart contract (indeed, in many cases the only available description coincides with the implementation).

Contributions. We propose a formal model of Bitcoin transactions. This model is abstract enough to allow for formal reasoning on the behaviour of Bitcoin transactions. For instance, we use our model to formally prove some properties of the Bitcoin blockchain, e.g. that transactions cannot be spent twice (Theorem 1), and that the overall value contained in the blockchains (excluding the coinbase transactions) is decreasing (Theorem 2).

Our model formally specifies some poorly documented features of Bitcoin, e.g. transaction signatures and signature modifiers (Definition 4), output scripts (Definitions 1 and 7), multi-signature verification (Definition 6), Segregated Witnesses (Definitions 2 and 9), paving the way towards automatic verification.

We make available an open-source tool[2] which translates transactions specified in our model to *standard* Bitcoin transactions.

Structure of the Paper. Section. 2 briefly recaps Bitcoin transactions, which we formalise in Sect. 3. Besides transactions, we also provide an high-level model of the blockchain, and we study its basic properties. In Sect. 4 we illustrate, through a basic case study, the impact of the Segregated Witness feature on the expressiveness of Bitcoin smart contracts. In Sect. 5 we show how to translate

[1] https://github.com/bitcoin/bitcoin.
[2] https://github.com/bitcoin-transaction-model/bitcoin-transaction-model.

transactions from our model to standard Bitcoin transactions. We discuss the differences between our model and the actual Bitcoin in Sect. 6.

2 Bitcoin Transactions in a Nutshell

We now give a minimalistic introduction to the behaviour of Bitcoin transactions (see [7] for a general survey on the other aspects of Bitcoin).

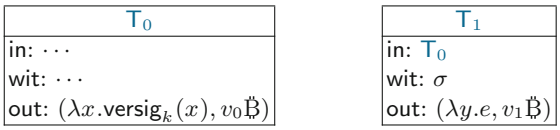


Fig. 1. Two Bitcoin transactions.

Users interact with Bitcoin through *addresses*, which they can freely generate. *Transactions* describe transfers of bitcoins (Ƀ) between addresses. The log of all transactions is recorded on a public, immutable and decentralised data structure called *blockchain*. To explain how the blockchain works, consider the transactions T_0 and T_1 displayed in Fig. 1. The transaction T_0 contains v_0 Ƀ, which can be *redeemed* by putting on the blockchain a transaction (e.g., T_1), whose in field is a reference to T_0. To redeem T_0, the *witness* of the redeeming transaction (the value in its wit field) must make the *output script* of T_0 (the first element of the pair in the out field) evaluate to true. When this happens, the value of T_0 is transferred to the new transaction, and T_0 is no longer redeemable.

In the example displayed before, the output script of T_0 evaluates to true when receiving a digital signature on the redeeming transaction T_1, with a given key pair k. We denote with $\mathsf{versig}_k(x)$ the verification of the signature x on the redeeming transaction: of course, since the signature must be included in the witness of the redeeming transaction, it will consider all the parts of that transaction *except* its wit field. We assume that σ is the signature of T_1.

Now, assume that the blockchain contains T_0, not yet redeemed, and someone tries to append T_1. To validate this operation, the nodes of the Bitcoin network check that $v_1 \leq v_0$, and then they evaluate the output script of T_0, by instantiating its formal parameter x to the signature σ in the witness of T_1. The function $\mathsf{versig}_k(\sigma)$ verifies that σ is actually the signature of T_1: therefore, the output script succeeds, and T_1 redeems T_0. Subsequently, a new transaction can redeem T_1 by satisfying its output script $\lambda y.e$ (not specified in the figure).

Bitcoin transactions may be more general than the ones illustrated by the previous example. First, there can be multiple inputs and outputs. Each output has an associated output script and value, and can be redeemed independently from the others. Consequently, in fields must specify which output they are redeeming. A transaction with multiple inputs associates a witness to each of them. The sum of the values of all the inputs must be greater or equal to the sum of the values

of all the outputs, otherwise the transaction is considered invalid. In its general form, the output script is a program in a (non Turing-complete) scripting language, featuring a limited set of logic, arithmetic, and cryptographic operators. Finally, a transaction can specify time constraints (absolute, or relative to its input transactions) about when it can appear on the blockchain.

3 A Formal Model of Bitcoin Transactions

In this section we introduce a formal model of Bitcoin transactions. We start in Sect. 3.1 by defining the scripts that can be used in transaction outputs. Then, in Sect. 3.2 we formalise transactions, and in Sect. 3.3 we define a signature scheme for them. Sections 3.4 and 3.5 give semantics, respectively, to scripts and transactions. In Sect. 3.6 we model the Bitcoin blockchain, and in particular we define the crucial notion of *consistency*, which corresponds to the one enforced by the Bitcoin consensus protocol. We then state a few results about consistent blockchains (their proofs are in Appendix A).

We start by introducing some auxiliary notation. We assume several sets, ranged over by meta-variables as shown in the left column of Table 1. We use the bold notation to denote finite sequences of elements. We denote with $\boldsymbol{x}_i$ the i-th element of a sequence $\boldsymbol{x}$, i.e. $\boldsymbol{x}_i = x_i$ if $\boldsymbol{x} = x_1 \dots x_n$, and with $\boldsymbol{x}_{i..j}$ the subsequence of $\boldsymbol{x}$ starting from the i-th element and ending to the j-th element. We denote with $|\boldsymbol{x}|$ the number of elements of $\boldsymbol{x}$, and with $[]$ the empty sequence. We denote with $f : A \rightharpoonup B$ a *partial* function f from A to B, with $\mathrm{dom}\, f$ the *domain* of f, i.e. the subset of A where f is defined, and with $\mathrm{ran}\, f$ the *range* of f, i.e. $\mathrm{ran}\, f = \{f(x) \mid x \in \mathrm{dom}\, f\}$. We use $\bot$ to represent an "undefined" element; in particular, when the element is a partial function, $\bot$ denotes the function with empty domain. For a pair (x, y), we define $\mathit{fst}(x, y) = x$ and $\mathit{snd}(x, y) = y$.

Table 1. Summary of notation.

$\mathsf{A}, \mathsf{B}, \dots \in \mathsf{Part}$	Participants	$e, e', \dots \in \mathsf{Exp}$	Script expressions
$x, y, \dots \in \mathsf{Var}$	Variables	$\mathsf{T}, \mathsf{T}', \dots \in \mathsf{Tx}$	Transactions
$\nu, \nu', \dots \in \mathsf{Den}$	Denotations, i.e.:	μ, μ'	Signature modifier
$k, k' \dots \in \mathbb{Z}$	Constants	$\mathsf{sig}_k^{\mu,i}(\mathsf{T})$	Transaction signature
$t, t' \dots \in \mathbb{N}$	Time	$\mathsf{ver}_k(\sigma, \mathsf{T}, i)$	Signature verification
$v, v' \dots \in \mathbb{N}$	Currency values	$\mathsf{T}, i \models \lambda \boldsymbol{x}.e$	Script verification
$\sigma, \sigma', \dots \in \mathbb{Z}$	Signatures	$(\mathsf{T}, i, t) \overset{v}{\rightsquigarrow} (\mathsf{T}', j, t')$	Transaction redeem
$\mathit{true}, \mathit{false}$	Boolean values	$\mathbf{B} = (\mathsf{T}_1, t_1) \cdots$	Blockchains
$\bot$	Undefined	$\mathbf{B} \rhd (\mathsf{T}, t)$	Consistent update

3.1 Scripts

Each output in a Bitcoin transaction contains a script, which is used to establish when the output can be redeemed by another transaction. Intuitively, a script

is a first-order function (written in a non Turing-equivalent language), which is applied to the witness provided by the redeeming transaction. The output can be redeemed only if such function application evaluates to true.

In our model, we abstract from the actual stack-based scripting language implemented in Bitcoin[3], by using instead a minimalistic language of expressions.

Definition 1 (Scripts). *We define the set* Exp *of* script expressions *(ranged over by* $e, e', \dots$*) as follows:*

$$e ::= x \mid k \mid e + e \mid e - e \mid e = e \mid e < e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid |e| \mid \mathsf{H}(e) \mid \mathsf{versig}_k(\boldsymbol{e}) \mid \mathsf{absAfter}\ t : e \mid \mathsf{relAfter}\ t : e$$

We denote with Script *the set of terms of the form* $\lambda \boldsymbol{z}.e$ *such that all the variables in* e *occur in* $\boldsymbol{z}$*.*

Besides some basic arithmetic and logical operators, script expressions include a few operators inspired from the actual Bitcoin scripting language. The expression $|e|$ denotes the size, in bytes, of the evaluation of e. The expression $\mathsf{H}(e)$ evaluates to the hash of e. The expression $\mathsf{versig}_k(\boldsymbol{e})$ takes as arguments a sequence of m script expressions, representing signatures of the enclosing transactions, and a sequence of n public keys. Intuitively, it evaluates to true whenever the provided signatures are verified by using m out of the n provided keys. The expressions $\mathsf{absAfter}\ t : e$ and $\mathsf{relAfter}\ t : e$ define temporal constraints (see Sect. 3.4). They evaluate as e if the constraints are satisfied, otherwise they fail.

Notation 1. We use the following syntactic sugar for expressions: (i) *false* to denote $1 = 0$ (ii) *true* to denote $1 = 1$ (iii) $e \wedge e'$ to denote $\mathsf{if}\ e\ \mathsf{then}\ e'\ \mathsf{else}\ \mathit{false}$ (iv) $e \vee e'$ to denote $\mathsf{if}\ e\ \mathsf{then}\ \mathit{true}\ \mathsf{else}\ e'$ (v) $\mathsf{not}\ e$ to denote $\mathsf{if}\ e\ \mathsf{then}\ \mathit{false}\ \mathsf{else}\ \mathit{true}$.

3.2 Transactions

The following definition formalises Bitcoin transactions.

Definition 2 (Transactions). *We inductively define the set* Tx *of transactions as follows. A transaction* T *is a tuple* $(\mathsf{in}, \mathsf{wit}, \mathsf{out}, \mathsf{absLock}, \mathsf{relLock})$*, where:*

- $\mathsf{in} : \mathbb{N} \rightharpoonup \mathsf{Tx} \times \mathbb{N}$
- $\mathsf{wit} : \mathbb{N} \rightharpoonup \mathbb{Z}^*$*, where* $\mathrm{dom}\,\mathsf{wit} = \mathrm{dom}\,\mathsf{in}$
- $\mathsf{out} : \mathbb{N} \rightharpoonup \mathsf{Script} \times \mathbb{N}$
- $\mathsf{absLock} : \mathbb{N}$
- $\mathsf{relLock} : \mathbb{N} \rightharpoonup \mathbb{N}$*, where* $\mathrm{dom}\,\mathsf{relLock} = \mathrm{dom}\,\mathsf{in}$

where, for all $i, j \in \mathrm{dom}\,\mathsf{in}$*,* $\mathit{fst}(\mathsf{in}(i)).\mathsf{wit} = \bot$ *and* $i \neq j \implies \mathsf{in}(i) \neq \mathsf{in}(j)$.
We denote with $\mathsf{T}.\mathsf{f}$ *the value of field* f *of* T*, for* $\mathsf{f} \in \{\mathsf{in}, \mathsf{wit}, \mathsf{out}, \mathsf{absLock}, \mathsf{relLock}\}$.
We say that T *is* initial *when* $\mathsf{T}.\mathsf{in} = \mathsf{T}.\mathsf{relLock} = \bot$ *and* $\mathsf{T}.\mathsf{absLock} = 0$.

[3] https://en.bitcoin.it/wiki/Script.

The fields in and out represent, respectively, the inputs and the outputs of a transaction. There is an input for each $i \in \mathrm{dom}\,\mathsf{in}$, and an output for each $j \in \mathrm{dom}\,\mathsf{out}$. When $\mathsf{T}.\mathsf{in}(i) = (\mathsf{T}', j)$, it means that the i-th input of T wants to redeem the j-th output of T'. The side condition $i \neq j \Rightarrow \mathsf{in}(i) \neq \mathsf{in}(j)$ ensures that inputs are pairwise distinct. The side condition $\mathit{fst}(\mathsf{in}(i)).\mathsf{wit} = \bot$ is related to the *Segregated Witness (SegWit)* feature[4], and it requires that the witness of the input transaction is left unspecified. The output $\mathsf{T}'.\mathsf{out}(j)$ is a pair $(\lambda \boldsymbol{z}.e, v)$, meaning that v Satoshis ($1\ \text{฿} = 10^8$ Satoshis) can be redeemed by whoever can provide a witness which satisfies $\lambda \boldsymbol{z}.e$. Such witness is defined by $\mathsf{T}.\mathsf{wit}(i)$. The fields $\mathsf{T}.\mathsf{absLock}$ and $\mathsf{T}.\mathsf{relLock}(i)$ specify a constraint on when T can be put on the blockchain: the first in absolute terms, whereas the second is relative to the transaction in the input $\mathsf{T}.\mathsf{in}(i)$. More specifically, $\mathsf{T}.\mathsf{absLock} = t$ means that T can appear on the blockchain only after time t. If $\mathsf{T}.\mathsf{relLock}(i) = t$, then T can appear only after time t since the transaction in $\mathsf{T}.\mathsf{in}(i)$ appeared.

To improve readability, we use the following conventions: (i) if T has exactly one input, we denote it by $\mathsf{T}.\mathsf{in}$ (omitting the index, which we assume to be 1); We act similarly for $\mathsf{T}.\mathsf{wit}$, $\mathsf{T}.\mathsf{out}$, and $\mathsf{T}.\mathsf{relLock}$; (ii) if $\mathsf{T}.\mathsf{absLock} = 0$, we omit it (similarly for $\mathsf{T}.\mathsf{relLock}$ when it is $\bot$); (iii) we denote with $\mathit{script}(\mathsf{T}.\mathsf{out}(i))$ and $\mathit{val}(\mathsf{T}.\mathsf{out}(i))$, respectively, the first and the second element of the pair $\mathsf{T}.\mathsf{out}(i)$.

3.3 Transaction Signatures

We extend to transactions the signing and verification functions of the signature schemes, denoted respectively as $\mathit{sig}_k(\cdot)$ and $\mathit{ver}_k(\cdot,\cdot)$. For simplicity, although we will always use $k = (k_p, k_s)$ for key pairs, we implicitly assume that $\mathit{sig}_k(\cdot)$ only uses the private part k_s, while $\mathit{ver}_k(\cdot,\cdot)$ only uses the public part k_p.

In Bitcoin, transaction signatures never apply to the whole transaction: users can specify which parts of a transaction are signed (with the exception of the wit field, which is never signed). However, not all possible combinations of transaction parts are possible; the legit ones are listed in Definition 4. In order to specify which parts of a transaction are signed, we first introduce the auxiliary notion of *transaction substitution*.

Definition 3 (Transaction substitutions). *A* transaction substitution Σ *is a function from* Tx *to* Tx. *For a transaction field* f*, we denote with* $\{\mathsf{f} \mapsto d\}$ *the substitution which replaces the value of* f *with* d. *For* $\mathsf{f} \neq \mathsf{absLock}$ *and* $i \in \mathbb{N}$*, we denote with* $\{\mathsf{f}(i) \mapsto d\}$ *the substitution which replaces* $\mathsf{f}(i)$ *with* d. *Further, for* $\circ \in \{<, >, \neq\}$*, we denote with* $\{\mathsf{f}(\circ\, i) \mapsto d\}$ *the substitution which replaces* $\mathsf{f}(j)$ *with* d*, for all* $j \circ i \in \mathrm{dom}\,\mathsf{f}$.

Definition 4 (Signature modifiers). *We define signature modifiers* μ_i *(with* $i \in \mathbb{N}$*) in Fig. 2. We associate to each modifier a substitution, and we denote with* $\mu_i(\mathsf{T})$ *the result of applying it to the transaction* T.

[4] This feature, specified in the BIP 141 and activated on August 24th 2017, implies that witnesses are not used in the computation of transaction hashes.

$$
\begin{aligned}
aa_i(\mathsf{T}) &= \mathsf{T}\{\mathsf{wit}(1) \mapsto i\}\{\mathsf{wit}(\neq 1) \mapsto \bot\} \\
an_i(\mathsf{T}) &= aa_i(\mathsf{T}\{\mathsf{out} \mapsto \bot\}) \\
as_i(\mathsf{T}) &= aa_i(\mathsf{T}\{\mathsf{out}(< i) \mapsto (\mathit{false}, 0)\}\{\mathsf{out}(> i) \mapsto \bot\}) \\
sa_i(\mathsf{T}) &= aa_1(\mathsf{T}\{\mathsf{in}(1) \mapsto \mathsf{T}.\mathsf{in}(i)\}\{\mathsf{in}(\neq 1) \mapsto \bot\} \\
&\qquad \{\mathsf{relLock}(1) \mapsto \mathsf{T}.\mathsf{relLock}(i)\}\{\mathsf{relLock}(\neq 1) \mapsto \bot\}) \\
sn_i(\mathsf{T}) &= sa_i(an_i(\mathsf{T})) \\
ss_i(\mathsf{T}) &= sa_i(as_i(\mathsf{T}))
\end{aligned}
$$

Fig. 2. Signature modifiers.

Each modifier is represented by a pair of symbols, describing, respectively, the set of inputs and of outputs being signed ($a =$ all, $s =$ single, $n =$ none), and an index $i \in \mathbb{N}$. The index has different meanings, depending on the modifier. Regarding the first symbol of the modifier, if it is a, then i is the index of the witness where the signature will be included, so to ensure that a signature computed for being included in the witness at index i can not be used in any witness with index $j \neq i$ (see Example 4). If the first symbol of the modifier is s, then only the i-th input is signed, while all the other inputs are removed from the transaction. With respect to the second symbol of the modifier, if it is s, then i is the index of the signed output; otherwise, i has no effect on the outputs to be signed. Note that a single index is used for both inputs and outputs: in any case, the index refers to the witness where the signature will be included.

Definition 5 (Transaction signatures). *We define the transaction signature (under modifier μ and index i) and verification functions as follows:*

$$\mathsf{sig}_k^{\mu,i}(\mathsf{T}) = (sig_k(\mu_i(\mathsf{T}), \mu), \mu) \quad \mathsf{ver}_k(\sigma, \mathsf{T}, i) = ver_k(w, (\mu_i(\mathsf{T}), \mu)) \;\; \textit{if } \sigma = (w, \mu)$$

Hereafter, we use $\sigma, \sigma', \ldots$ to range over transaction signatures.

Note that a signature $\sigma = (sig_k((\mu_i(\mathsf{T}), \mu)), \mu)$ does not contain the index i. Consequently, the verification function requires i to be passed as parameter, i.e. we write $\mathsf{ver}_k(\sigma, \mathsf{T}, i)$. The parameter i will be instantiated by the script verification function (see Definition 8). Besides the modified transaction $\mu_i(\mathsf{T})$, the signature also applies to the modifier μ. In this way, signing a single-input transaction T with modifier aa_1 and with modifier sa_1 results in two different signatures, even though $aa_1(\mathsf{T}) = sa_1(\mathsf{T})$.

Notation 2. Note that $\mathsf{sig}_k^{\mu,i}(\mathsf{T})$ can meaningfully appear within $\mathsf{T}.\mathsf{wit}(i)$, since such signature does not depend on the wit field of transactions (as all signature modifiers overwrite all the witnesses). When a signature of T appears within $\mathsf{T}.\mathsf{wit}(i)$, as a shorthand we denote it with sig_k^{μ} (so, neglecting the enclosing transaction T and the index i), or just sig_k when $\mu = aa$.

We now extend the signature verification $\mathsf{ver}_k(\sigma, \mathsf{T}, i)$ to the case where, instead of providing a single key k and a single signature σ, one has many keys

and signatures, i.e. $\mathsf{ver}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i)$. Intuitively, if $|\boldsymbol{\sigma}| = m$ and $|\boldsymbol{k}| = n$, the function $\mathsf{ver}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i)$ implements a m-of-n multi-signature scheme, i.e. it evaluates to true if all the m signatures match (some of) the keys in $\boldsymbol{k}$. The actual definition is a bit more complex, to be coherent with the one implemented in Bitcoin.

Definition 6 (Multi-signature verification). *Let $\boldsymbol{k}$ and $\boldsymbol{\sigma}$ be sequences of (public) keys and signatures such that $|\boldsymbol{k}| \geq |\boldsymbol{\sigma}|$, and let $i \in \mathbb{N}$. For all $m, n \in \mathbb{N}$, we define the function:*

$$\mathsf{ver}^{n,m}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i) \equiv \begin{cases} \mathit{true} & \mathit{if}\ m = 0 \\ \mathit{false} & \mathit{if}\ m \neq 0\ \mathit{and}\ n = 0 \\ \mathsf{ver}^{n-1,m-1}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i) & \mathit{if}\ m, n \neq 0\ \mathit{and}\ \mathsf{ver}_{\boldsymbol{k}_n}(\boldsymbol{\sigma}_m, \mathsf{T}, i) \\ \mathsf{ver}^{n-1,m}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i) & \mathit{otherwise} \end{cases}$$

Then, we define $\mathsf{ver}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i) = \mathsf{ver}^{|\boldsymbol{k}|,|\boldsymbol{\sigma}|}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, i)$.

Our formalisation of multi-signature verification (Definition 6) follows closely the implementation of Bitcoin, whose stack-based scripting language imposes that the sequence $\boldsymbol{\sigma}$ is read in reverse order. Accordingly, the function ver tries to verify the last signature in $\boldsymbol{\sigma}$ with the last key in $\boldsymbol{k}$. If they match, the function ver proceeds to verify the previous signature in the sequence, otherwise it tries to verify the signature with the previous key.

Example 1 (2-of-3 multi-signature). Let $\boldsymbol{k} = k_a k_b k_c$, and let $\boldsymbol{\sigma} = \sigma_p \sigma_q$ be such that $\mathsf{ver}_{k_a}(\sigma_p, \mathsf{T}, 1) = \mathsf{ver}_{k_b}(\sigma_q, \mathsf{T}, 1) = \mathit{true}$, and false otherwise. We have that:

$$\begin{aligned} \mathsf{ver}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, 1) &= \mathsf{ver}^{3,2}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, 1) && (\text{as } |\boldsymbol{k}| = 3 \text{ and } |\boldsymbol{\sigma}| = 2) \\ &= \mathsf{ver}^{2,2}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_c}(\sigma_q, \mathsf{T}, 1) = \mathit{false}) \\ &= \mathsf{ver}^{1,1}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_b}(\sigma_q, \mathsf{T}, 1) = \mathit{true}) \\ &= \mathsf{ver}^{0,0}_{\boldsymbol{k}}(\boldsymbol{\sigma}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_a}(\sigma_p, \mathsf{T}, 1) = \mathit{true}) \\ &= \mathit{true} && (\text{as } m = 0) \end{aligned}$$

Note that, if we let $\boldsymbol{\sigma'} = \sigma_q \sigma_p$, the resulting evaluation will be:

$$\begin{aligned} \mathsf{ver}_{\boldsymbol{k}}(\boldsymbol{\sigma'}, \mathsf{T}, 1) &= \mathsf{ver}^{3,2}_{\boldsymbol{k}}(\boldsymbol{\sigma'}, \mathsf{T}, 1) && (\text{as } |\boldsymbol{k}| = 3 \text{ and } |\boldsymbol{\sigma'}| = 2) \\ &= \mathsf{ver}^{2,2}_{\boldsymbol{k}}(\boldsymbol{\sigma'}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_c}(\sigma_p, \mathsf{T}, 1) = \mathit{false}) \\ &= \mathsf{ver}^{1,2}_{\boldsymbol{k}}(\boldsymbol{\sigma'}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_b}(\sigma_p, \mathsf{T}, 1) = \mathit{false}) \\ &= \mathsf{ver}^{0,1}_{\boldsymbol{k}}(\boldsymbol{\sigma'}, \mathsf{T}, 1) && (\text{as } \mathsf{ver}_{k_a}(\sigma_p, \mathsf{T}, 1) = \mathit{true}) \\ &= \mathit{false} && (\text{as } m \neq 0 \text{ and } n = 0) \end{aligned}$$

□

3.4 Semantics of Scripts

Definition 7 gives the semantics of script expressions. This semantics will be used in Sect. 3.5 to define when a transaction can redeem another one. We use

an environment $\rho : \mathsf{Var} \rightharpoonup \mathbb{Z}$ which associates a denotation to each variable occurring in it. Further, we use a transaction $\mathsf{T} \in \mathsf{Tx}$ and an index $i \in \mathbb{N}$ to indicate the witness redeeming the script, both used to evaluate the timelock expressions. We use the denotation $\bot$ to represent "failure" of the evaluation. This is the case e.g. of timelock expressions, when the temporal constraint is not satisfied. All the semantic operators used in Definition 7 are *strict*, i.e. they evaluate to $\bot$ if some of their operands is $\bot$.

$$
\begin{aligned}
[\![x]\!]_{\mathsf{T},i,\rho} &= \rho(x) \\
[\![k]\!]_{\mathsf{T},i,\rho} &= k \\
[\![\mathsf{versig}_{\boldsymbol{k}}(\boldsymbol{e})]\!]_{\mathsf{T},i,\rho} &= \mathsf{ver}_{\boldsymbol{k}}([\![\boldsymbol{e}]\!]_{\mathsf{T},i,\rho}, \mathsf{T}, i) \\
[\![\mathsf{H}(e)]\!]_{\mathsf{T},i,\rho} &= H([\![e]\!]_{\mathsf{T},i,\rho}) \qquad (H \text{ is a public hash function}) \\
[\![\mathsf{absAfter}\ t : e]\!]_{\mathsf{T},i,\rho} &= \mathit{if}\ \mathsf{T}.\mathsf{absLock} \geq t\ \mathit{then}\ [\![e]\!]_{\mathsf{T},i,\rho}\ \mathit{else}\ \bot \\
[\![\mathsf{relAfter}\ t : e]\!]_{\mathsf{T},i,\rho} &= \mathit{if}\ \mathsf{T}.\mathsf{relLock}(i) \geq t\ \mathit{then}\ [\![e]\!]_{\mathsf{T},i,\rho}\ \mathit{else}\ \bot \\
[\![e \circ e']\!]_{\mathsf{T},i,\rho} &= [\![e]\!]_{\mathsf{T},i,\rho} \circ_\bot [\![e']\!]_{\mathsf{T},i,\rho} \qquad (\circ \in \{+, -, =, <\}) \\
[\![|e|]\!]_{\mathsf{T},i,\rho} &= \mathit{size}([\![e]\!]_{\mathsf{T},i,\rho}) \\
[\![\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]_{\mathsf{T},i,\rho} &= \mathit{if}\ [\![e_0]\!]_{\mathsf{T},i,\rho}\ \mathit{then}\ [\![e_1]\!]_{\mathsf{T},i,\rho}\ \mathit{else}\ [\![e_2]\!]_{\mathsf{T},i,\rho}
\end{aligned}
$$

Fig. 3. Semantics of script expressions.

Definition 7 (Expression evaluation). *Let $\rho : \mathsf{Var} \rightharpoonup \mathbb{Z}$, let $\mathsf{T} \in \mathsf{Tx}$ and $i \in \mathbb{N}$. We define the function $[\![\cdot]\!]_{\mathsf{T},i,\rho} : \mathsf{Exp} \to \mathsf{Den}$ in Fig. 3, where we use the following operators on denotations:*

$$
\mathit{if}\ \nu_0\ \mathit{then}\ \nu_1\ \mathit{else}\ \nu_2 \equiv \begin{cases} \nu_1 & \mathit{if}\ \nu_0 = \mathit{true} \\ \nu_2 & \mathit{if}\ \nu_0 = \mathit{false} \\ \bot & \mathit{otherwise} \end{cases} \qquad \mathit{size}(\nu) \equiv \begin{cases} \bot & \mathit{if}\ \nu \notin \mathbb{Z} \\ 0 & \mathit{if}\ \nu = 0 \\ \left\lceil \frac{\log_2 |\nu|}{7} \right\rceil & \mathit{otherwise} \end{cases}
$$

$$
\nu_0 \circ_\bot \nu_1 \equiv \mathit{if}\ \nu_0, \nu_1 \in \mathbb{Z}\ \mathit{then}\ \nu_0 \circ \nu_1\ \mathit{else}\ \bot \qquad (\circ \in \{+, -, =, <\})
$$

Definition 8 (Script verification). *We say that the input i of T* verifies *$\lambda \boldsymbol{x}.e$ (in symbols: $\mathsf{T}, i \models \lambda \boldsymbol{x}.e$) when $\boldsymbol{x} = x_1 \ldots x_n$, $\mathsf{T}.\mathsf{wit}(i) = k_1 \ldots k_n$, and:*

$$
[\![e]\!]_{\mathsf{T},i,\{x_j \mapsto k_j \mid j \in 1 \ldots n\}} = \mathit{true}
$$

Example 2. Let H be a hash function, let $s, h \in \mathbb{Z}$ be such that $h = H(s)$, and let T be such that $\mathsf{T}.\mathsf{wit}(1) = (\sigma, s)$, with $\sigma = \mathsf{sig}_k^{aa}(\mathsf{T})$. We prove that:

$$
\mathsf{T}, 1 \models \lambda(\varsigma, x).\big(\mathsf{versig}_k(\varsigma)\ \mathsf{and}\ \mathsf{H}(x) = h\big)
$$

To do this, let $\rho = \{\varsigma \mapsto \sigma, x \mapsto s\}$. We have that:

$$
\begin{aligned}
&[\![\mathsf{versig}_k(\varsigma)\ \mathsf{and}\ \mathsf{H}(x) = h]\!]_{\mathsf{T},1,\rho} = [\![\mathsf{versig}_k(\varsigma)]\!]_{\mathsf{T},1,\rho}\ \mathit{and}\ [\![\mathsf{H}(x) = h]\!]_{\mathsf{T},1,\rho} \\
&= \mathsf{ver}_k([\![\varsigma]\!]_{\mathsf{T},1,\rho}, \mathsf{T}, 1)\ \mathit{and}\ ([\![\mathsf{H}(x)]\!]_{\mathsf{T},1,\rho} =_\bot [\![h]\!]_{\mathsf{T},1,\rho}) \\
&= \mathsf{ver}_k(\rho(\varsigma), \mathsf{T}, 1)\ \mathit{and}\ (H([\![x]\!]_{\mathsf{T},1,\rho}) =_\bot h) = \mathsf{ver}_k(\sigma, \mathsf{T}, 1)\ \mathit{and}\ (H(\rho(x)) =_\bot h) \\
&= \mathit{true}
\end{aligned}
$$

□

3.5 Semantics of Transactions

Definition 9 describes when the j-th input of a transaction T' (put on the blockchain at time t') can redeem v Satoshis from the i-th output of the transaction T (put on the blockchain at time t). We denote this by $(\mathsf{T}, i, t) \overset{v}{\rightsquigarrow} (\mathsf{T}', j, t')$.

Definition 9 (Output redeeming). *We write* $(\mathsf{T}, i, t) \overset{v}{\rightsquigarrow} (\mathsf{T}', j, t')$ *iff all the following conditions hold:*

(a) $\mathsf{T}'.\mathsf{in}(j) = (\mathsf{T}\{\mathsf{wit} \mapsto \bot\}, i)$
(b) $\mathsf{T}', j \models \mathit{script}(\mathsf{T}.\mathsf{out}(i))$
(c) $v = \mathit{val}(\mathsf{T}.\mathsf{out}(i))$
(d) $t' \geq \mathsf{T}'.\mathsf{absLock}$
(e) $t' - t \geq \mathsf{T}'.\mathsf{relLock}(j)$

We write $(\mathsf{T}, i, t) \not\rightsquigarrow (\mathsf{T}', j, t')$ *when for no* v *it holds that* $(\mathsf{T}, i, t) \overset{v}{\rightsquigarrow} (\mathsf{T}', j, t')$.

Item (a) links the j-th input of T' to the i-th output of T. Note that, since we are modelling SegWit, the witness in the transaction $\mathsf{T}'.\mathsf{in}(j)$ is left unspecified: this is why we set to $\bot$ also the witness of T. Item (b) requires that the j-th witness of T' verifies the i-th output script of T. Item (c) just defines v as the value in the i-th output of T. Items (d) and (e) check the absolute and relative timelocks, respectively. The first constraint states that T' cannot appear on the blockchain before $\mathsf{T}'.\mathsf{absLock}$; the second one states that T' cannot appear until at least $\mathsf{T}'.\mathsf{relLock}(j)$ time units have elapsed since T was put on the blockchain.

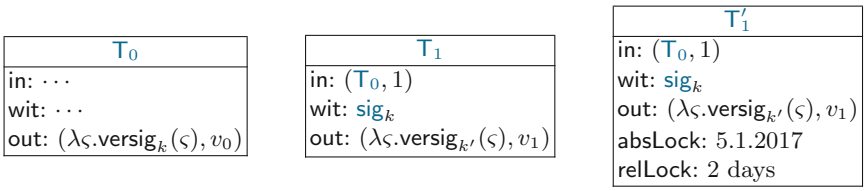


Fig. 4. Three transactions. For notational conciseness, when displaying transactions we omit the substitution $\{\mathsf{wit} \mapsto \bot\}$ for the transaction within the in field (e.g., we just write T_0 within $\mathsf{T}_1.\mathsf{in}$). Also, we use dates in time constraints.

Example 3. With the transactions in Fig. 4, we have $(\mathsf{T}_0, 1, t_0) \overset{v_0}{\rightsquigarrow} (\mathsf{T}_1, 1, t_1)$. Indeed, for item (a) we have that $\mathsf{T}_1.\mathsf{in}(1) = (\mathsf{T}_0\{\mathsf{wit} \mapsto \bot\}, 1)$; for item (b), $\mathsf{T}_1, 1 \models \lambda\varsigma.\mathsf{versig}_k(\varsigma)$; for item (c), $v_0 = \mathit{val}(\mathsf{T}_0.\mathsf{out}(1))$. The other two items trivially hold, as there are no time constraints. We also have $(\mathsf{T}_0, 1, 2.1.2017) \overset{v_0}{\rightsquigarrow} (\mathsf{T}'_1, 1, 6.1.2017)$. To show that, we have to check also items (d) and (e). For item (d), we have that $6.1.2017 \geq \mathsf{T}'_1.\mathsf{absLock} = 5.1.2017$. For item (e), we have that $6.1.2017 - 2.1.2017 \geq \mathsf{T}'_1.\mathsf{relLock}(1) = 2$ days. □

T'_1
in: ...
wit: ...
out: $(\lambda\varsigma.\mathsf{versig}_k(\varsigma), 1)$

T'_2
in: ...
wit: ...
out: $(\lambda\varsigma.\mathsf{versig}_k(\varsigma), 2)$

T'_3
in: $1 \mapsto (\mathsf{T}'_1, 1), 2 \mapsto (\mathsf{T}'_2, 1)$
wit: $1 \mapsto \mathsf{sig}_k, 2 \mapsto \mathsf{sig}_k$
out: $(\lambda\varsigma.\mathsf{versig}_{k_2}(\varsigma), 3)$

Fig. 5. Three transactions for Example 4. Note that, by Notation 2, the first witness of T'_3 is $\mathsf{sig}_k^{aa,1}(\mathsf{T}'_3)$, while the second is $\mathsf{sig}_k^{aa,2}(\mathsf{T}'_3)$.

Example 4. Consider the transactions in Fig. 5. The signature in $\mathsf{T}'_3.\mathsf{wit}(1)$ is computed as follows:

$$\begin{aligned}\mathsf{sig}_k^{aa,1}(\mathsf{T}'_3) &= (sig_k(aa_1(\mathsf{T}'_3, aa)), aa) && \text{by Definition 5}\\ &= (sig_k(\mathsf{T}'_3\{\mathsf{wit}(1) \mapsto 1\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}, aa), aa) && \text{by Definition 4}\end{aligned}$$

We prove that, when verifying $(\mathsf{T}'_1, 1, t) \overset{1}{\rightsquigarrow} (\mathsf{T}'_3, 1, t')$, item (b) of Definition 9 holds, i.e. $\mathsf{T}'_3, 1 \models script(\mathsf{T}'_1.\mathsf{out}(1))$. To this purpose, let $\rho = \{\varsigma \mapsto (w, aa)\}$, where $w = sig_k(\mathsf{T}'_3\{\mathsf{wit}(1) \mapsto 1\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}, aa)$. We have that:

$$\begin{aligned}[\![\mathsf{versig}_k(\varsigma)]\!]_{\mathsf{T}'_3,1,\rho} &= \mathsf{ver}_k([\![\varsigma]\!]_{\mathsf{T}'_3,1,\rho}, \mathsf{T}'_3, 1) && \text{by Def. 7}\\ &= \mathsf{ver}_k((w, aa), \mathsf{T}'_3, 1) && \rho(\varsigma) = (w, aa)\\ &= ver_k(w, (aa_1(\mathsf{T}'_3), aa)) && \text{by Def. 5}\\ &= ver_k(w, (\mathsf{T}'_3\{\mathsf{wit}(1) \mapsto 1\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}, aa)) && \text{by Def. 4}\\ &= true && \text{by Def. of } w\end{aligned}$$

We now show that w is *not* valid for the other witness, i.e. $(\mathsf{T}'_2, 1, t) \overset{2}{\not\rightsquigarrow} (\mathsf{T}''_3, 2, t')$, where $\mathsf{T}''_3 = \mathsf{T}'_3\{\mathsf{wit}(2) \mapsto \mathsf{sig}_k^{aa,1}(\mathsf{T}'_3)\}$. Let $\rho = \{\varsigma \mapsto (w, aa)\}$. Item (b) of Definition 9 does not hold:

$$\begin{aligned}[\![\mathsf{versig}_k(\varsigma)]\!]_{\mathsf{T}''_3,2,\rho} &= \mathsf{ver}_k((w, aa), \mathsf{T}''_3, 2) && \text{as above}\\ &= ver_k(w, (aa_2(\mathsf{T}''_3), aa)) && \text{by Def. 5}\\ &= ver_k(w, (\mathsf{T}''_3\{\mathsf{wit}(1) \mapsto 2\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}, aa)) && \text{by Def. 4}\\ &= false\end{aligned}$$

In the last equation, w is not a valid signature for $\mathsf{T}''_3\{\mathsf{wit}(1) \mapsto 2\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}$ because it is computed on $\mathsf{T}'_3\{\mathsf{wit}(1) \mapsto 1\}\{\mathsf{wit}(\neq 1) \mapsto \bot\}$, and the two transactions differ on $\mathsf{wit}(1)$. □

3.6 Blockchain and Consistency

In Definition 10 we model blockchains as sequences of *timed transactions* (T, t), where t represents the time when the transaction T has been added. Note that our definition is very permissive: for instance, it allows a blockchain to contain transactions which do not redeem any transactions, or double-spent transactions. We will rule out such *inconsistent* blockchains later on in Definition 13.

Definition 10 (Blockchain). *A blockchain* $\mathbf{B}$ *is a sequence* $(\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$*, where* T_1 *is the only transaction with* $\mathsf{in} = \bot$*, and* $t_i \leq t_j$ *for all* $1 \leq i \leq j \leq n$*.*

We denote with $trans_{\mathbf{B}}$ *the set of transactions occurring in* $\mathbf{B}$*, and with* $time_{\mathbf{B}}(\mathsf{T}_i)$ *the time* t_i *of transaction* T_i *in* $\mathbf{B}$*. Given a transaction* T*, we define* $match_{\mathbf{B}}(\mathsf{T})$ *as the set of transactions* T_i *such that* $\mathsf{T}\{\mathsf{wit} \mapsto \bot\} = \mathsf{T}_i\{\mathsf{wit} \mapsto \bot\}$*.*

Definition 11 (Unspent output). *Let* $\mathbf{B} = (\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ *be a blockchain. We say that the output* j *of transaction* T_i *is* unspent *in* $\mathbf{B}$ *whenever:*

$$\forall i' \leq n, j' \in \mathbb{N} \;\; : \;\; (\mathsf{T}_i, j, t_i) \not\rightsquigarrow (\mathsf{T}_{i'}, j', t_{i'})$$

Given a blockchain $\mathbf{B}$*, we define:*

- $UTXO_{\mathbf{B}}$*, the* Unspent Transaction Output *of* $\mathbf{B}$*, as the set of pairs* (T_i, j) *such that output* j *of* T_i *is unspent in* $\mathbf{B}$*.*
- $val(\mathbf{B})$*, the* value *of* $\mathbf{B}$*, as the sum of the values of all outputs in its UTXO.*

T_1	
in:	$\bot$
wit:	$\bot$
out:	$1 \mapsto (\lambda\varsigma.\mathsf{versig}_{k_1}(\varsigma), 3)$
	$2 \mapsto (\lambda\varsigma.\mathsf{versig}_{k_2}(\varsigma), 5)$
	$3 \mapsto (\lambda\varsigma.\mathsf{versig}_{k_3}(\varsigma), 7)$

T_2	
in:	$1 \mapsto (\mathsf{T}_1, 2), 2 \mapsto (\mathsf{T}_1, 3)$
wit:	$1 \mapsto \mathsf{sig}_{k_2}, 2 \mapsto \mathsf{sig}_{k_3}$
out:	$(\lambda\varsigma.\mathsf{versig}_{k_2}(\varsigma), 10)$

T_3	
in:	$(\mathsf{T}_1, 2)$
wit:	sig_{k_2}
out:	$(\lambda\varsigma.\mathsf{versig}_{k_2}(\varsigma), 5)$

Fig. 6. Three transactions for Examples 5 to 7.

Example 5. Consider the transactions in Fig. 6, and let $\mathbf{B} = (\mathsf{T}_1, 0)(\mathsf{T}_2, t_2)$. We have that $(\mathsf{T}_1, 2, 0) \overset{5}{\rightsquigarrow} (\mathsf{T}_2, 1, t_2)$ and $(\mathsf{T}_1, 3, 0) \overset{7}{\rightsquigarrow} (\mathsf{T}_2, 2, t_2)$, while the other outputs are unspent. Hence, the UTXO of $\mathbf{B}$ is $\{(\mathsf{T}_1, 1), (\mathsf{T}_2, 1)\}$. □

The following definition establishes when (T, t) is a *consistent update* of $\mathbf{B}$.

Definition 12 (Consistent update). *We write* $\mathbf{B} \rhd (\mathsf{T}, t)$ *iff either* $\mathbf{B} = []$*,* T *is initial and* $t = 0$*, or, given, for all* $i \in \mathrm{dom}\,(\mathsf{T}.\mathsf{in})$*:*

$$\begin{aligned}
&\{\mathsf{T}'_i\} = match_{\mathbf{B}}(fst(\mathsf{T}.\mathsf{in}(i))) && \textit{(redeemed transaction)}\\
&o_i = snd(\mathsf{T}.\mathsf{in}(i)) && \textit{(redeemed output index)}\\
&t'_i = time_{\mathbf{B}}(\mathsf{T}'_i) && \textit{(time when } \mathsf{T}'_i \textit{ was added to } \mathbf{B}\textit{)}\\
&v_i = val(\mathsf{T}'_i.\mathsf{out}(o_i)) && \textit{(value of the redeemed output)}
\end{aligned}$$

the following conditions hold:

(1) $\forall i \in \mathrm{dom}\,\mathsf{T}.\mathsf{in} : (\mathsf{T}'_i, o_i) \in UTXO_{\mathbf{B}}$
(2) $\forall i \in \mathrm{dom}\,\mathsf{T}.\mathsf{in} : (\mathsf{T}'_i, o_i, t'_i) \overset{v_i}{\rightsquigarrow} (\mathsf{T}, i, t)$
(3) $\sum\{v_i \mid i \in \mathrm{dom}\,\mathsf{T}.\mathsf{in}\} \geq \sum\{val(\mathsf{T}.\mathsf{out}(j)) \mid j \in \mathrm{dom}\,\mathsf{T}.\mathsf{out}\}$
(4) $\mathbf{B} = \mathbf{B}'(\mathsf{T}', t') \implies t \geq t'$

Firstly, for each $\mathsf{T}.\mathsf{in}(i)$ we obtain the singleton $\{\mathsf{T}'_i\}$ from the blockchain, using $match_{\mathbf{B}}$, such that $fst(\mathsf{T}.\mathsf{in}(i))\{\mathsf{wit} \mapsto \bot\} = \mathsf{T}'_i\{\mathsf{wit} \mapsto \bot\}$. The update is inconsistent if $match_{\mathbf{B}}(fst(\mathsf{T}.\mathsf{in}(i)))$ is not a singleton for some i. Condition (1) requires that the redeemed outputs are currently unspent in $\mathbf{B}$. Condition (2) asks that each input of T redeems an output of a transaction in $\mathbf{B}$. Condition (3) requires that the sum of the values of the outputs of T is not greater than the total value it redeems. Finally, (4) requires that the time of T is greater than or equal to the time of the last transaction in $\mathbf{B}$.

Example 6. Consider again the transactions in Fig. 6, and let $\mathbf{B} = (\mathsf{T}_1, 0)$. We prove that $\mathbf{B} \rhd (\mathsf{T}_2, t_2)$. Let $o_1 = 2$, $o_2 = 3$, $t'_1 = t'_2 = 0$, $v_1 = 5$, $v_2 = 7$. We now prove that the conditions of Definition 12 are satisfied. For condition (1), note that both $(\mathsf{T}_1, 2)$ and $(\mathsf{T}_1, 3)$ are unspent, according to Definition 11. For condition (2), note that:

$$(\mathsf{T}_1, 2, 0) \overset{v_1}{\rightsquigarrow} (\mathsf{T}_2, 1, t_2) \qquad (\mathsf{T}_1, 3, 0) \overset{v_2}{\rightsquigarrow} (\mathsf{T}_2, 2, t_2)$$

hold, according to Definition 9. Finally, for condition (3), we have that:

$$\sum\{v_i \mid i \in \{1, 2\}\} = 5 + 7 \;\geq\; \sum\{val(\mathsf{T}_2.\mathsf{out}(j)) \mid j \in \mathrm{dom}\,\mathsf{T}_2.\mathsf{out}\} = 10$$

Therefore, (T_2, t_2) is a consistent update of $\mathbf{B}$. □

Example 7 (Double spending). Consider again the transactions in Fig. 6, and let $\mathbf{B} = (\mathsf{T}_1, 0)(\mathsf{T}_2, t_2)$.

We prove that (T_3, t_3) is *not* a consistent update of $\mathbf{B}$. Although condition (2) of Definition 12 holds:

$$(\mathsf{T}_1, 2, 0) \overset{5}{\rightsquigarrow} (\mathsf{T}_3, 1, t_3)$$

we have that condition (1) is *not* satisfied. In fact, according to Definition 11, $(\mathsf{T}_1, 2)$ is already spent in $\mathbf{B}$ because

$$(\mathsf{T}_1, 2, 0) \overset{5}{\rightsquigarrow} (\mathsf{T}_2, 1, t_2)$$

holds and both T_1 and T_2 are in $\mathbf{B}$. Since T_3 is trying to spend an output already spent, this transaction should not be appended to $\mathbf{B}$. □

We now define when a blockchain is consistent. Intuitively, consistency holds when the blockchain has been constructed, started from the empty one, by appending consistent updates, only. The actual definition is given by induction.

Definition 13 (Consistency). *We say that a blockchain* $\mathbf{B}$ *is consistent if either* $\mathbf{B} = []$*, or* $\mathbf{B} = \mathbf{B}'(\mathsf{T}, t)$ *with* $\mathbf{B}'$ *consistent and* $\mathbf{B}' \rhd (\mathsf{T}, t)$*.*

Note that the empty blockchain is consistent; the blockchain with a single transaction (T_1, t_1) is consistent iff T_1 is initial and $t_1 = 0$. The transaction T_1 models the first transaction in the *genesis block* (as discussed in Sect. 6, we are abstracting away the *coinbase* transactions, which forge new bitcoins).

We now establish some basic properties of consistent blockchains. Lemma 1 states that, in a consistent blockchain, the inputs of a transaction point backwards to the output of some transaction in the blockchain.

Lemma 1. *If* $(\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ *is consistent, then:*

$$\forall i \in 2 \ldots n : \forall (\mathsf{T}, h) \in \mathrm{ran}\,(\mathsf{T}_i.\mathsf{in}) : \exists j < i : \mathsf{T}_j\{\mathsf{wit} \mapsto \bot\} = \mathsf{T} \ \wedge \ h \in \mathrm{dom}\,(\mathsf{T}_j.\mathsf{out})$$

The following theorem establishes that a transaction output cannot be redeemed twice in a consistent blockchain.

Theorem 1 (No double spending). *If* $(\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ *is consistent, then:*

$$\forall i \neq j \in 1 \ldots n \ : \ \mathrm{ran}\,(\mathsf{T}_i.\mathsf{in}) \cap \mathrm{ran}\,(\mathsf{T}_j.\mathsf{in}) = \emptyset$$

The following lemma states that there can be at most a single match of an arbitrary transaction within a consistent blockchain. This implies that the in field of an arbitrary transaction points at most to one transaction output within the blockchain.

Lemma 2. *If* $\mathbf{B}$ *is consistent, then for all transactions* T*,* $match_{\mathbf{B}}(\mathsf{T})$ *contains at most one element.*

Lemma 3 ensures that all the transactions on a consistent blockchain are pairwise distinct, even when neglecting their witnesses.

Lemma 3. *If* $(\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ *is consistent, then:*

$$\forall i \neq j \in 1 \ldots n \ : \ \mathsf{T}_i\{\mathsf{wit} \mapsto \bot\} \neq \mathsf{T}_j\{\mathsf{wit} \mapsto \bot\}$$

The following theorem states that the overall value of a blockchain decreases as the blockchain grows. This is because our model does not keep track of the *coinbase transactions*, which in Bitcoin allow miners to collect transaction fees (the difference between inputs and outputs of a transaction), and block rewards.

Theorem 2 (Non-increasing value). *Let* $\mathbf{B}$ *be a consistent blockchain, and let* $\mathbf{B}'$ *be a non-empty prefix of* $\mathbf{B}$*. Then,* $val(\mathbf{B}') \geq val(\mathbf{B})$*.*

Note that the scripting language and its semantics are immaterial in all the statements above. Actually, proving these results never involves checking condition (b) of Definition 9. Of course, the choice of the scripting language affects the expressiveness of the smart contracts built upon Bitcoin.

4 Example: Static Chains of Transactions

We now formally specify in our model a simple smart contract[5], which illustrates the impact of SegWit on the expressiveness of Bitcoin contracts.

A participant A wants to send an indirect payment of 1 ₿ to C, routing it through B. To authorize the payment, B wants to keep a fee of 0.1 ₿. However, A is afraid that B will keep all the money for himself, so she exploits the following contract. She creates a *chain* of transactions, as shown in Fig. 7. The transaction T_{AB} transfers 1.1 ₿ from A to B (but it is not signed by A, yet), while T_{BC} transfers 1 ₿ from B to C. We assume that $(\mathsf{T}_{\mathsf{A}}, 1)$ is a transaction output redeemable by A through her key k_{A}, and that k_{B} is the key of B.

T_{AB}
in: $(\mathsf{T}_{\mathsf{A}}, 1)$
wit: $\bot$
out: $(\lambda\varsigma_{\mathsf{A}}\varsigma_{\mathsf{B}}.\mathsf{versig}_{k_{\mathsf{A}}k_{\mathsf{B}}}(\varsigma_{\mathsf{A}}\varsigma_{\mathsf{B}}), 1.1₿)$

T_{BC}	
in:	$(\mathsf{T}_{\mathsf{AB}}, 1)$
wit:	$\bot$
out:	$1 \mapsto (\lambda\varsigma_{\mathsf{B}}.\mathsf{versig}_{k_{\mathsf{B}}}(\varsigma_{\mathsf{B}}), 0.1₿)$
	$2 \mapsto (\lambda\varsigma_{\mathsf{C}}.\mathsf{versig}_{k_{\mathsf{C}}}(\varsigma_{\mathsf{C}}), 1₿)$

Fig. 7. Transactions of the chain contract.

The protocol of A is the following: A starts by asking B for his signature on T_{BC}, ensuring that C will be paid. After receiving and verifying the signature, A puts T_{AB} on the blockchain, adding her signature on the wit field. Then, she also appends T_{BC}, replacing the wit field with her signature and B's one. Since A takes care of publishing the transactions, the behaviour of B consists just in sending his signature on T_{BC}.

Remarkably, this contract relies on the SegWit feature: indeed, without SegWit it no longer works. We can disable SegWit by changing our model as follows:

- in Definition 2, we no longer require that $\forall i \in \mathrm{dom}\,\mathsf{in} : \mathit{fst}(\mathsf{in}(i)).\mathsf{wit} = \bot$
- in Definition 9, we replace item (a) with the condition: $\mathsf{T}'.\mathsf{in}(j) = (\mathsf{T}, i)$
- in Definition 10, we let $\mathit{match}_{\mathbf{B}}(\mathsf{T}) = \{\mathsf{T}\}$ if T occurs in $\mathbf{B}$, empty otherwise.

To see why disabling SegWit breaks the contract, assume that the transaction $\mathsf{T} = \mathsf{T}_{\mathsf{AB}}\{\mathsf{wit} \mapsto \mathsf{sig}^{aa}_{k_{\mathsf{A}}}(\mathsf{T}_{\mathsf{AB}})\}$ is unspent on the blockchain, when participant A attempts to append also $\mathsf{T}' = \mathsf{T}_{\mathsf{BC}}\{\mathsf{wit} \mapsto \mathsf{sig}^{aa}_{k_{\mathsf{A}}}(\mathsf{T}_{\mathsf{BC}})\,\mathsf{sig}^{aa}_{k_{\mathsf{B}}}(\mathsf{T}_{\mathsf{BC}})\}$. To be a consistent update, by item (2) of Definition 12 we must have (for some $t_1 \leq t_2$):

$$(\mathsf{T}, 1, t_1) \overset{1\,₿}{\rightsquigarrow} (\mathsf{T}', 1, t_2) \tag{1}$$

For this, all the conditions in Definition 9 must hold. However, since we have disabled SegWit, for item (a) we no longer check that:

$$\mathsf{T}'.\mathsf{in}(1) = (\mathsf{T}\{\mathsf{wit} \mapsto \bot\}, 1)$$

[5] https://www.bitcoinhk.org/media/presentations/2016-03-16/2016-03-16-Segregated_Witness.pdf.

but instead we need to check the condition:

$$\tilde{\mathsf{T}}'.\mathsf{in}(1) \;=\; (\tilde{\mathsf{T}}, 1) \tag{2}$$

where the transactions $\tilde{\mathsf{T}}, \tilde{\mathsf{T}}'$ correspond to the non-SegWit versions of T, T', i.e. their in fields point to their actual parents, according to the new Definition 2.

Hence, condition (2) checks the equality between $\tilde{\mathsf{T}}_{\mathsf{AB}}$ (the transaction in the input of $\tilde{\mathsf{T}}'$) and $\tilde{\mathsf{T}}_{\mathsf{AB}}\{\mathsf{wit} \mapsto \mathsf{sig}^{aa}_{k_{\mathsf{A}}}(\tilde{\mathsf{T}}_{\mathsf{AB}})\}$ (the transaction $\tilde{\mathsf{T}}$). Note that all the fields of the second transaction—but the wit field—are equal to those of the first transaction. Instead, the witness of $\tilde{\mathsf{T}}_{\mathsf{AB}}$ is $\bot$, while the one of $\tilde{\mathsf{T}}$ contains the signature of A. This difference in the wit field is ignored with the SegWit semantics, while it is discriminating for the older version of Bitcoin.

A naïve attempt to amend the contract would be to set the input field of $\tilde{\mathsf{T}}'$ to $\tilde{\mathsf{T}}$. However, this would invalidate the signature of A on $\tilde{\mathsf{T}}'$.

5 Compiling to Standard Bitcoin Transactions

We now sketch how to compile the transactions of our abstract model into concrete Bitcoin transactions. In particular, we aim at producing *standard* Bitcoin transactions, which respect further constraints on their fields[6]. This is crucial, because non-standard transactions are mostly discarded by the Bitcoin network.

Our compiler produces output scripts of the following kinds, which are all allowed in standard transactions:

Pay to Public Key Hash (P2PKH) takes as parameters a public key and a signature, and checks that (i) the hash of the public key matches the hash hardcoded in the script; (ii) the signature is verified against the public key.

Pay to Script Hash (P2SH) contains only a hash (say, h). The actual script $\lambda \boldsymbol{x}.e$—which is *not* required to be standard—is contained instead in the wit field of the redeeming transaction, alongside with the actual parameters $\boldsymbol{k}$. The evaluation succeeds if $H(\lambda \boldsymbol{x}.e) = h$ and $(\lambda \boldsymbol{x}.e)\boldsymbol{k}$ evaluates to *true*. The only constraint imposed by P2SH is on the size of the script, which is limited to the size of a stack element (520 bytes).

OP_RETURN allows to put up to 80 bytes of data in an output script, making the output unredeemable.

We compile the scripts of the form $\lambda\varsigma.\mathsf{versig}_k(\varsigma)$ to P2PKH, and those of the form $\lambda.k$ to OP_RETURN. All other scripts are compiled to P2SH when they comply with the size constraint, otherwise compilation fails. In this way, our compiler always produces standard transactions.

Our compiler exploits the *alternative stack* as temporary storage of the variable values. In this way we cope with the stack-based nature of the Bitcoin scripting language. For instance, for the script $\lambda x.\mathsf{H}(x) = \mathsf{H}(x+1)$, the variable x is pushed on the alternative stack beforehand, then duplicated and copied in the main stack before each operation involving x.

[6] https://bitcoin.org/en/developer-guide#standard-transactions.

6 Conclusions

We have proposed a formal model for Bitcoin transactions. Our model abstractly describes their essential aspects, at the same time enabling formal reasoning, and providing a formal specification to some of Bitcoin's less documented features.

An alternative model of transactions in blockchain systems has been proposed in [8]. Roughly, blockchains are represented as directed acyclic graphs, where edges denote transfers of assets. This model is quite abstract, so that it can be instantiated to different blockchains (e.g., Bitcoin, Ethereum, and Hyperledger Fabric). Differently from ours, the model in [8] does not capture some peculiar features of Bitcoin, like e.g. transaction signatures and signature modifiers, output scripts, multi-signature verification, and Segregated Witnesses.

Our work provides the theoretical foundations to model Bitcoin smart contracts, reducing the gap between cryptography and programming languages communities. A formal description of smart contracts enables their automated verification and analysis, which are of crucial importance in a context where design flaws may result in loss of money. For instance, our model has been exploited in [3] to present a comprehensive survey of Bitcoin smart contracts.

Differences Between Our Model and Bitcoin. There are some differences between our model and the actual Bitcoin, which we outline below.

In Definition 2, we stipulate that the in field of a transaction points to another transaction. Instead, in Bitcoin the in field contains the identifier of the input transaction. More specifically, this identifier is defined as $H(\mu(\mathsf{T}))$, where: (i) $\mu = \{\mathsf{wit} \mapsto \bot\}$ since the activation of the SegWit feature; (ii) $\mu = \bot$, beforehand. Consequently, the condition $(\mathsf{T}, i, t) \overset{v}{\rightsquigarrow} (\mathsf{T}', j, t')$ item (a) of Definition 9 would be translated in Bitcoin as: $\mathsf{T}'.\mathsf{in}(j) = (\mathsf{H}(\mu(\mathsf{T}'')), i)$, where $\mathsf{H}(\mu(\mathsf{T}'')) = \mathsf{H}(\mu(\mathsf{T}))$. Intuitively, the in field specifies the transaction (and the output index) to redeem. Since the activation of SegWit, the computation of the transaction identifier does not take in account the wit field.

The scripting language in Definition 1 is a bit more expressive than Bitcoin's. For instance, the script $\lambda x.\mathsf{H}(x) < k$ is admissible in our model, while it is not in Bitcoin. Indeed, the Bitcoin scripting language only admits the comparison (via the `OP_LESSTHANOREQUAL` opcode) on 32-bit integers, while two arbitrary values can only be tested for equality (via the `OP_EQUAL` opcode). Similar restrictions apply to arithmetic operations. It is straightforward to adapt our model to apply the same restrictions on Bitcoin scripts. Indeed, our compiler already implements a simple type system which rules away scripts not admissible in Bitcoin.

Definition 10 models blockchains as sequences of transactions, while in Bitcoin they are sequences of *blocks* of transactions. In this way, we are abstracting both from the cryptographic puzzle that miners have to solve to append new blocks to the blockchain, and from the *coinbase transactions*, which (like our initial transaction) do not redeem other transactions, and mint new bitcoins (the block rewards). Coinbase transactions are also used in Bitcoin to collect transaction fees, which are just discarded in our model. Extending our model with coinbase transactions would falsify Theorem 2, since the overall value in the

blockchain would no longer be decreasing. Definition 10 requires the timestamp of each transaction to increase monotonically. Instead, in Bitcoin a timestamp is valid if it is greater than the median timestamp of previous 11 blocks.

In Definitions 2 and 9, the absLock and relLock fields specify the time when a transaction can be appended to the blockchain. In Bitcoin transactions, besides the time we can also use the *block height*, i.e. the distance between any given block and the genesis block. Setting the block height to h implies that the transaction can be mined from the block h onward.

Acknowledgments. The authors thank the anonymous reviewers of Financial Cryptography 2018 and A. S. Podda for their insightful comments. This work is partially supported by Aut. Reg. Sardinia project P.I.A. 2013 "NOMAD". Stefano Lande gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Aut. Reg. Sardinia, European Social Fund 2014–2020).

A Proofs

Proof of Lemma 1
By Definition 13, (T_i, t_i) is a consistent update of $(\mathsf{T}_1, t_1) \cdots (\mathsf{T}_{i-1}, t_{i-1})$. The thesis follows from condition (2) of Definition 12. □

Proof of Theorem 1
Let $\mathbf{B} = (\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ be consistent. By contradiction, assume that there exist $i < j$ and i', j' such that $\mathsf{T}_i.\mathsf{in}(i') = \mathsf{T}_j.\mathsf{in}(j')$. By consistency, there exist h, h' such that $(\mathsf{T}_h\{\mathsf{wit} \mapsto \bot\}, h') = \mathsf{T}_i.\mathsf{in}(i')$. Since $\mathbf{B}_{1..i-1} \rhd (\mathsf{T}_i, t_i)$, then by item (2) of Definition 12 it must be $(\mathsf{T}_h, h', t_h) \rightsquigarrow (\mathsf{T}_i, i', t_i)$. Hence, by Definition 11 it follows that (T_h, h') is already *spent* in $\mathbf{B}$. Since $\mathbf{B}_{1..j-1} \rhd (\mathsf{T}_j, t_j)$, by item (1) of Definition 12, (T_h, h') must be *unspent*—contradiction. □

Proof of Lemma 2
Let $\mathbf{B} = (\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$ be consistent. By contradiction, assume that $\mathsf{T}_i, \mathsf{T}_j \in \mathit{match}_{\mathbf{B}}(\mathsf{T})$, with $\mathsf{T}_i \neq \mathsf{T}_j$ (and so, $i \neq j$). By Definition 10 it must be $\mathsf{T}_i\{\mathsf{wit} \mapsto \bot\} = \mathsf{T}\{\mathsf{wit} \mapsto \bot\} = \mathsf{T}_j\{\mathsf{wit} \mapsto \bot\}$, hence in particular $\mathsf{T}_i.\mathsf{in} = \mathsf{T}_j.\mathsf{in}$. There are two cases. If $\mathsf{T}_i.\mathsf{in} = \mathsf{T}_j.\mathsf{in} = \bot$, then by Definition 10 $\mathbf{B}$ is not a blockchain, since $i \neq j$. Hence, $\mathrm{ran}\,(\mathsf{T}_i.\mathsf{in}) \cap \mathrm{ran}\,(\mathsf{T}_j.\mathsf{in}) = \mathrm{ran}\,(\mathsf{T}_i.\mathsf{in}) \neq \emptyset$. By Theorem 1, this cannot happen because $\mathbf{B}$ is consistent—contradiction. □

Proof of Lemma 3
Straightforward from Lemma 2, taking $\mathsf{T} = \mathsf{T}_j$. □

Proof of Theorem 2
Let $\mathbf{B} = (\mathsf{T}_1, t_1) \cdots (\mathsf{T}_n, t_n)$. By contradiction, there exists some $i < n$ such that, given $\mathbf{B}_i = (\mathsf{T}_1, t_1) \cdots (\mathsf{T}_i, t_i)$:

$$\mathit{val}(\mathbf{B}_i) \;<\; \mathit{val}(\mathbf{B}_i(\mathsf{T}_{i+1}, t_{i+1}))$$

Let U_i and U_{i+1} be the UTXOs of $\mathbf{B}_i$ and of $\mathbf{B}_i(\mathsf{T}_{i+1}, t_{i+1})$, respectively, and let $U = U_i \cap U_{i+1}$. Since $\mathit{val}(U_i) < \mathit{val}(U_{i+1})$, then it must be $\mathit{val}(U_i \setminus U) < \mathit{val}(U_{i+1} \setminus U)$. The set $U_i \setminus U$ contains the outputs redeemed by T_{i+1}, while the

set $U_{i+1} \setminus U$ contains exactly the outputs in T_{i+1}. Since $\mathbf{B}$ is consistent, then $\mathbf{B}_i \rhd (\mathsf{T}_{i+1}, t_{i+1})$. Then, by Definition 12, for each $k \in \mathrm{dom}\, \mathsf{T}_{i+1}.\mathsf{in}$, there exists a unique $j \leq i$ such that, given $o_k = snd(\mathsf{T}_{i+1}.\mathsf{in}(k))$ and $v_k = val(\mathsf{T}_j.\mathsf{out}(o_k))$:

$$(\mathsf{T}_j, o_k, t_j) \overset{v_k}{\rightsquigarrow} (\mathsf{T}_{i+1}, k, t_{i+1})$$

Then, by item (3) of Definition 12:

$$\begin{aligned} val(U_i \setminus U) &= \sum \{v_k \mid k \in \mathrm{dom}\, \mathsf{T}_{i+1}.\mathsf{in}\} \\ &\geq \sum \{val(\mathsf{T}_{i+1}.\mathsf{out}(h)) \mid h \in \mathrm{dom}\, \mathsf{T}_{i+1}.\mathsf{out}\} = val(U_{i+1} \setminus U) \end{aligned}$$

while we assumed $val(U_i \setminus U) < val(U_{i+1} \setminus U)$—contradiction. □

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: Fair two-party computations via Bitcoin deposits. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014. LNCS, vol. 8438, pp. 105–121. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44774-1_8
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multi-party computations on Bitcoin. In: IEEE Symposium on Security and Privacy, pp. 443–458 (2014)
3. Atzei, N., Bartoletti, M., Cimoli, T., Lande, S., Zunino, R.: SoK: unraveling Bitcoin smart contracts. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 217–242. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89722-6_9
4. Banasik, W., Dziembowski, S., Malinowski, D.: Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 261–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_14
5. Bartoletti, M., Zunino, R.: Constant-deposit multiparty lotteries on Bitcoin. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 231–247. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_15
6. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
7. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE S & P, pp. 104–121 (2015)
8. Cachin, C., Caro, A.D., Moreno-Sanchez, P., Tackmann, B., Vukolić, M.: The transaction graph for modeling blockchain semantics. Cryptology ePrint Archive, Report 2017/1070 (2017). https://eprint.iacr.org/2017/1070
9. Kumaresan, R., Bentov, I.: How to use Bitcoin to incentivize correct computations. In: ACM CCS, pp. 30–41 (2014)
10. Kumaresan, R., Moran, T., Bentov, I.: How to use Bitcoin to play decentralized poker. In: ACM CCS, pp. 195–206 (2015)
11. Miller, A., Bentov, I.: Zero-collateral lotteries in Bitcoin and Ethereum. In: EuroS&P Workshops, pp. 4–13 (2017)

12. Möser, M., Eyal, I., Gün Sirer, E.: Bitcoin covenants. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 126–141. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_9
13. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
14. O'Connor, R., Piekarska, M.: Enhancing bitcoin transactions with covenants. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 191–198. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_12
15. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997). http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548

Author Index