Shlomi Dolev
Jonathan Katz
Amnon Meisels (Eds.)

LNCS 13301

Cyber Security, Cryptology, and Machine Learning

6th International Symposium, CSCML 2022
Be'er Sheva, Israel, June 30 – July 1, 2022
Proceedings

Springer

Lecture Notes in Computer Science 13301

Shlomi Dolev · Jonathan Katz ·
Amnon Meisels (Eds.)

Cyber Security, Cryptology, and Machine Learning

6th International Symposium, CSCML 2022
Be'er Sheva, Israel, June 30 – July 1, 2022
Proceedings

Springer

Editors
Shlomi Dolev
Ben-Gurion University of the Negev
Be'er Sheva, Israel

Jonathan Katz
University of Maryland
College Park, MD, USA

Amnon Meisels
Ben-Gurion University of the Negev
Be'er Sheva, Israel

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-031-07688-6 ISBN 978-3-031-07689-3 (eBook)
https://doi.org/10.1007/978-3-031-07689-3

Preface

CSCML, the International Symposium on Cyber Security, Cryptography, and Machine Learning, is an international forum for researchers, entrepreneurs, and practitioners in the theory, design, analysis, implementation, or application of cyber security, cryptography, and machine learning systems and networks and, in particular, of conceptually innovative topics in these research areas. Information technology has become crucial to our everyday lives, an indispensable infrastructure of our society, and therefore a target for attacks by malicious parties. Cyber security is one of the most important fields of research these days because of these developments. Two of the (sometimes competing) fields of research, cryptography and machine learning are the most important building blocks of cyber security.

Topics of interest for CSCML include: cyber security design; secure software development methodologies; formal methods, semantics, and verification of secure systems; fault tolerance, reliability, and availability of distributed secure systems; game-theoretic approaches to secure computing; automatic recovery self-stabilizing and self-organizing systems; communication, authentication, and identification security; cyber security for mobile systems and the Internet of Things; cyber security of corporations; security and privacy for cloud, edge, and fog computing; cryptocurrency; blockchain; cryptography; cryptographic implementation analysis and construction; secure multi-party computation; privacy enhancing technologies and anonymity; post-quantum cryptology and security; machine learning and big data; anomaly detection and malware identification; business intelligence and security; digital forensics, digital rights management; trust management and reputation systems; and information retrieval, risk analysis, and DoS.

The 6th CSCML took place during June 30–July 1, 2022, in Beer-Sheva, Israel. The keynote speakers were Michal Braverman-Blumenstyk, Microsoft Corporate Vice President, Cloud and AI Division CTO, and Israel R&D Center General Manager; Dr. Burt Kaliski, Jr., SVP and Chief Technology Officer at Verisign; and Shlomo Dovrat, Co-founder and General Partner at Viola Ventures. The conference was organized in cooperation with the International Association for Cryptologic Research (IACR), and selected papers will appear in a dedicated special issue of the Journal of Computer and System Sciences.

This volume contains 24 contributions selected by the Program Committee from 51 submissions, and also includes 11 short papers. All submitted papers were read and evaluated by members of the Program Committee assisted by external reviewers. We thank the members of the Program Committee for all their hard work.

We are grateful to the EasyChair system that was used for the reviewing process. We also gratefully acknowledge the support of IBM and Ben-Gurion University of the Negev (BGU), in particular BGU-NHSA, the BGU Lynne and William Frankel Center

for Computer Science, the BGU Cyber Security Research Center, and the Department of Computer Science.

March 2022

Shlomi Dolev
Jonathan Katz
Amnon Meisels

Organization

CSCML, the International Symposium on Cyber Security, Cryptography, and Machine Learning, is an international forum for researchers, entrepreneurs, and practitioners in the theory, design, analysis, implementation, and application of cyber security, cryptography, or machine-learning systems.

Founding Steering Committee

Orna Berry	Google Cloud, Israel
Shlomi Dolev (Chair)	Ben-Gurion University of the Negev, Israel
Yuval Elovici	Ben-Gurion University of the Negev, Israel
Bezalel Gavish	Southern Methodist University, USA
Ehud Gudes	Ben-Gurion University of the Negev, Israel
Jonathan Katz	University of Maryland, USA
Rafail Ostrovsky	University of California, Los Angeles, USA
Jeffrey D. Ullman	Stanford University, USA
Kalyan Veeramachaneni	MIT, USA
Yaron Wolfsthal	IBM, Israel
Moti Yung	Columbia University and Google, USA

Organizing Committee

General Chair

Shlomi Dolev	Ben-Gurion University of the Negev, Israel

Program Chairs

Jonathan Katz	University of Maryland, USA
Amnon Meisels	Ben-Gurion University of the Negev, Israel

Organizing Chair

Rosemary Franklin	Ben-Gurion University of the Negev, Israel

Program Committee

Gilad Asharov	Bar-Ilan University, Israel
Manuel Barbosa	HASLAB-INESC TEC and FCUP, Portugal
Don Beaver	Meta, Novi Research, USA
Alex Biryukov	University of Luxembourg, Luxembourg
Dor Bitan	University of California, Berkeley, USA
Carlo Blundo	Università degli Studi di Salerno, Italy
Harry Buhrman	CWI, University of Amsterdam, and QuSoft, The Netherlands
Ashish Choudhury	IIIT Bangalore, India
Hadassa Daltrophe	Sami Shamoon College of Engineering, Israel
Stefan Dziembowski	University of Warsaw, Poland
Oren Freifeld	Ben-Gurion University of the Negev, Israel
Felix Freiling	FAU, Germany
Benjamin Fuller	University of Connecticut, USA
Juan A. Garay	Texas A&M University, USA
Craig Gentry	Algorand Foundation, USA
Niv Gilboa	Ben-Gurion University of the Negev, Israel
Ehud Gudes	Ben-Gurion University of the Negev, Israel
Shay Gueron	University of Haifa and Amazon, Israel
David Heath	Georgia Institute of Technology, USA
Gene Itkis	MIT Lincoln Lab and US Military Academy, West Point, USA
Bhavana Kanukurthi	Indian Institute of Science, India
Çetin Kaya Koç	University of California, Santa Barbara, USA
Vladimir Kolesnikov	Georgia Institute of Technology, USA
Benjamin Kreuter	University of Virginia and Google, USA
Ranjit Kumaresan	University of Maryland, USA
Daniel Masny	Meta, USA
Thomas Peyrin	Nanyang Technological University, Singapore
Rami Puzis	Ben-Gurion University of the Negev, Israel
Eyal Ronen	Tel Aviv University, Israel
Alexander Russell	University of Connecticut, USA
Alessandra Scafuro	North Carolina State University, USA
Berry Schoenmakers	Eindhoven University of Technology, The Netherlands
Gil Segev	Hebrew University of Jerusalem, Israel
Qiang Tang	University of Sydney, Australia
Tamir Tassa	The Open University of Israel, Israel
Nikos Triandopoulos	Stevens Institute of Technology, USA
Ni Trieu	Arizona State University, USA
Eran Tromer	Tel Aviv University, Israel

Sponsors

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

In cooperation with

Contents

Blind Rotation in Fully Homomorphic Encryption with Extended Keys

Marc Joye(✉) and Pascal Paillier

Zama, Paris, France
marc.joye@zama.ai

Abstract. Most solutions for fully homomorphic encryption rely on hard lattice problems. Accordingly, the resulting ciphertexts must contain a certain level of noise to guarantee the security of the encryption. Running homomorphic operations on these noisy ciphertexts in turn further increases the noise level in the resulting ciphertexts. If the noise exceeds a given threshold, the ciphertexts are no longer decryptable. Bootstrapping enables to deal with this issue by resetting the noise present in a ciphertext to a nominal level.

Certain fully homomorphic encryption schemes require the use of binary keys for the bootstrapping operation. This paper describes how to extend the underlying blind rotation so as to efficiently support a wider number of key formats. It also investigates a multi-digit approach wherein multiple key digits are processed concurrently. All in all, the proposed solutions offer more flexibility in the parameter selection and yield a variety of new trade-offs for better performance.

Keywords: Fully homomorphic encryption · FHEW · TFHE · Key generation · Private machine learning

1 Introduction

Fully homomorphic encryption (FHE) [16] is a very powerful cryptographic primitive. It allows performing arbitrary computation directly on encrypted data—without ever requiring intermediate decryption.

All known FHE instantiations follow the same blueprint, as originally devised in Gentry's seminal paper [11]. The idea is (i) to start with a "somewhat" homomorphic encryption scheme which supports a bounded number of homomorphic operations and (ii) to convert it into a *fully* homomorphic encryption scheme. The conversion step is referred to as the *bootstrapping*. Basically, this is accomplished by homomorphically evaluating the decryption function on the ciphertext. The resulting ciphertext encrypts the same plaintext but is somehow "refreshed". Homomorphic operations can therefore be further iterated again and again. This is particularly appealing in the case of private machine learning, where inference is performed homomorphically over user-encrypted data using substantially deep models with a number of layers *in the hundreds*.

The bootstrapping is however a relatively demanding operation and, despite of being intensively studied (e.g., [2,3,6,7,9,10,12,17]), remains the main bottleneck in current FHE implementations. Another approach is to avoid bootstrappings altogether and to increase the cryptographic parameters accordingly in order to accommodate the circuit being evaluated in a *leveled* way [5]. This, in turn, limits homomorphic inference to smaller models with at most a few tens of layers.

Our Contributions. The most efficient bootstrappings to date are achieved by GSW-derived cryptosystems [7,9] and their variants. The TFHE cryptosystem [7] displays a time of a few tens of milliseconds to perform a bootstrapping with typical parameters on a regular laptop. By design, TFHE (and its variants like [8]) requires binary keys in an essential way for the bootstrapping.

In a recent work, Micciancio and Polyakov [15] remark that a ternary key can be viewed as a difference of two binary keys. This allows them to evaluate a bootstrapping with ternary keys as a series of two original bootstrappings (*i.e.*, with binary keys). Moving from binary keys to ternary keys or more can be useful at it allows for trade-offs (speed/size of the bootstrapping keys/noise level) otherwise not necessarily applicable for a given security level.

Another important motivation to switch from binary to ternary (or more) is to easily fall back to the next best alternative, should *e.g.* the special case of binary keys reveal less secure than expected. Our extended settings allow one to carefully reevaluate the new best trade-offs available after any kind of security-impacting breakthrough.

Our main results are:

– The core operation for the bootstrapping in TFHE and the likes is a blind rotation. It consists of a succession of *external products* which comprise polynomial multiplications and integer recodings. If n denotes the number of digits of a ternary key, the method of Micciancio–Polyakov requires $2n$ external products. We rely on an additive representation and show how a bootstrapping with ternary keys can be done with only n external products. Furthermore, our method features the same memory requirements as in Micciancio–Polyakov's method for storing the corresponding bootstrapping keys.
– We demonstrate that our approach is generic and can be adapted to support arbitrary key formats. Specifically, we extend our ternary approach to keys expressed in a general radix. Somewhat surprisingly, for an encryption key represented with n digits, the number of external products required to complete a bootstrapping is equal to n. We note that the dimension n is a decreasing function of the radix.
– In [17], Zhou *et al.* point out that two bits of the secret key can be processed concurrently during the bootstrapping of the TFHE scheme. Their approach was later refined in [4]. We extend our additive splitting to higher radices and detail how the number of external products can be reduced to n/d by processing d digits per iteration. The number of bootstrapping keys however increases very quickly. In practice, d is likely restricted to small values like $d = 2$ or $d = 3$.

Outline of the Paper. The rest of the paper is organized as follows. In the next section, we review the bootstrapping behind TFHE, including its extension to programmable bootstrapping. Section 3 introduces our main technique. We explain how the multiplicative splitting in Micciancio–Polyakov's approach can be turned into an additive one. This gives rise to a more efficient bootstrapping for ternary keys. We then generalize our new approach to higher-radix representations in Sect. 4. We also cover the case of multiple digits processed concurrently. Section 5 provides a performance analysis according to the parameter selection. Finally, we conclude the paper in Sect. 6.

2 Programmable Bootstrapping

The bootstrapping is an essential technique for FHE as it enables to control the noise growth and to refresh ciphertexts whenever the noise exceeds a given level. In this section, we review the bootstrapping for the TFHE family and its extension to *programmable* bootstrapping.

The (Discretized) TFHE Scheme. For our purposes, we consider the TFHE family. We follow the presentation of [8,14].[1] Define the polynomial ring $\mathbb{Z}_{N,q}[X] = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N+1)$ where q and N are powers of 2. Define also the binary set $\mathbb{B} = \{0,1\}$ and $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N+1)$. For a secret key $\boldsymbol{\mathfrak{s}} \xleftarrow{\$} \mathbb{B}_N[X]^k$, the GLWE encryption of a plaintext $\bar{\mathfrak{u}} \in \mathbb{Z}_{N,q}[X]$ is given by $\boldsymbol{\bar{\mathfrak{c}}} = (\bar{\mathfrak{a}}_1, \dots, \bar{\mathfrak{a}}_k, \bar{\mathfrak{b}}) \in \mathbb{Z}_{N,q}[X]^{k+1}$ where $\bar{\mathfrak{a}}_j \xleftarrow{\$} \mathbb{Z}_{N,q}[X]$ and $\bar{\mathfrak{b}} = \sum_{j=1}^{k} \mathfrak{s}_j\, \bar{\mathfrak{a}}_j + \bar{\mathfrak{u}}^*$ with $\bar{\mathfrak{u}}^* = \bar{\mathfrak{u}} + \bar{\mathfrak{e}}$ for some (small) random noise $\bar{\mathfrak{e}}$. We write $\boldsymbol{\bar{\mathfrak{c}}} \leftarrow \mathrm{GLWE}_{\boldsymbol{\mathfrak{s}}}(\bar{\mathfrak{u}})$. When $(N,k) = (1,n)$, it turns out that $\mathbb{Z}_{N,q}[X] = \mathbb{Z}/q\mathbb{Z}$ and the above procedure leads to an LWE ciphertext. We then write $\boldsymbol{\bar{c}} \leftarrow \mathrm{LWE}_{\boldsymbol{s}}(\bar{\mu}) = (\bar{a}_1, \dots, \bar{a}_n, \bar{b}) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ as the encryption of a plaintext $\bar{\mu} \in \mathbb{Z}/q\mathbb{Z}$ under the secret key $\boldsymbol{s} = (s_1, \dots, s_n) \in \mathbb{B}^n$, where $\bar{a}_j \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ and $\bar{b} = \sum_{j=1}^{n} s_j\, \bar{a}_j + \bar{\mu}^*$ with $\bar{\mu}^* = \bar{\mu} + \bar{e}$ for some (small) random noise $\bar{e}$.

Blind Rotation. The main step in the TFHE bootstrapping is the so-called blind rotation. It converts an LWE ciphertext $\boldsymbol{\bar{c}} \leftarrow \mathrm{LWE}_{\boldsymbol{s}}(\bar{\mu}) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ into a ciphertext $\boldsymbol{\bar{\mathfrak{c}}'} \leftarrow \mathrm{GLWE}_{\boldsymbol{\mathfrak{s}}'}(X^{-\tilde{\mu}^*} \cdot \bar{\mathfrak{v}}) \in \mathbb{Z}_{N,q}[X]^{k+1}$; namely, a GLWE encryption of $X^{-\tilde{\mu}^*} \cdot \bar{\mathfrak{v}}$ under key $\boldsymbol{\mathfrak{s}}' \in \mathbb{B}_N[X]^k$, where $\tilde{\mu}^*$ is a rounded approximation of $\bar{\mu}^* = \bar{\mu} + \bar{e}$ and $\bar{\mathfrak{v}}$ is a "test" polynomial. Specifically, if $\boldsymbol{\bar{c}} = (\bar{a}_1, \dots, \bar{a}_n, \bar{b})$ then

$$-\tilde{\mu}^* = -\tilde{b} + \sum_{j=1}^{n} s_j\, \tilde{a}_j \pmod{2N}$$

where

$$\begin{cases} \tilde{b} = \left\lceil \frac{2N(\bar{b} \bmod q)}{q} \right\rfloor \\ \tilde{a}_j = \left\lceil \frac{2N(\bar{a}_j \bmod q)}{q} \right\rfloor \quad (1 \le j \le n) \end{cases}$$

[1] As originally described, TFHE is defined over the real torus $\mathbb{R}/\mathbb{Z}$. We rather consider the discretized torus $q^{-1}\mathbb{Z}/\mathbb{Z}$ and identify its elements with integers modulo q.

and the test polynomial $\bar{v}$ is programmed as a look-up table so that $X^{-\tilde{\mu}^*} \cdot \bar{v}$, up to a random indexing error (*a.k.a.* drift), encodes $f(\bar{\mu})$ for a chosen function f; see [8,14] for details.

The blind rotation is detailed in Algorithm 1. It requires n bootstrapping keys, $\mathsf{bsk}[j] \leftarrow \mathrm{GGSW}_{\mathfrak{s}'}(s_j)$ for $1 \le j \le n$; GGSW denotes a (general) Gentry–Sahai–Waters encryption [13] derived from a gadget matrix $\mathbf{G}$. The blind rotation can be calculated as a series of CMux operations. On input a GGSW ciphertext $\overline{\boldsymbol{\mathscr{C}}}$ encrypting a bit $\mathrm{b} \in \{0,1\}$ and two GLWE ciphertexts $\bar{\boldsymbol{c}}_\mathbf{0}$ and $\bar{\boldsymbol{c}}_\mathbf{1}$, the CMux operation outputs a ciphertext encrypting the same plaintext as $\bar{\boldsymbol{c}}_\mathbf{b}$,

$$\bar{\boldsymbol{c}}' \leftarrow \mathrm{CMux}(\overline{\boldsymbol{\mathscr{C}}}, \bar{\boldsymbol{c}}_\mathbf{0}, \bar{\boldsymbol{c}}_\mathbf{1}) := \bar{\boldsymbol{c}}_\mathbf{0} + \overline{\boldsymbol{\mathscr{C}}} \boxdot (\bar{\boldsymbol{c}}_\mathbf{1} - \bar{\boldsymbol{c}}_\mathbf{0})$$

where $\boxdot$ denotes the external product of ciphertexts. The external product dominates the cost in a blind rotation. Given a GGSW ciphertext $\boldsymbol{\mathscr{C}}_\mathbf{1} \leftarrow \mathrm{GGSW}(\mu_1)$ and a GLWE ciphertext $\boldsymbol{c}_\mathbf{2} \leftarrow \mathrm{GLWE}(\mu_2)$, their external product is defined as $\boldsymbol{\mathscr{C}}_\mathbf{1} \boxdot \boldsymbol{c}_\mathbf{2} = G^{-1}(\boldsymbol{c}_\mathbf{2}) \cdot \boldsymbol{\mathscr{C}}_\mathbf{1}$ where $G^{-1}(\boldsymbol{c}_\mathbf{2})$ is the gadget decomposition of $\boldsymbol{c}_\mathbf{2}$. The transformation G^{-1} flattens the vector of polynomials $\boldsymbol{c}_\mathbf{2} \in \mathbb{Z}_{N,q}[X]^{k+1}$ into a row vector of $(k+1)\ell$ polynomials of $\mathbb{Z}[X]/(X^N+1)$ with small signed coefficients. The goal is to contain the noise growth. For better efficiency, the underlying polynomial multiplications are carried out in the Fourier domain (dyadic multiplications). Most of the time in an external product of ciphertexts is spent in going back and forth in the Fourier domain. Again, we refer the reader to [8,14] for details.

Algorithm 1: Blind rotation (binary case).

```
1 $\mathsf{acc} \leftarrow (0,\ldots,0,X^{-\tilde{b}} \cdot \bar{v})$
2 for $j = 1$ to $n$ do
3     $\mathsf{acc} \leftarrow \mathrm{CMux}(\mathsf{bsk}[j], \mathsf{acc}, X^{\tilde{a}_j} \cdot \mathsf{acc})$
          /* $\mathsf{acc} \leftarrow \mathsf{acc} + \mathsf{bsk}[j] \boxdot \big((X^{\tilde{a}_j} - 1) \cdot \mathsf{acc}\big)$ */
4 end for
5 return acc
```

It can be verified that at the end of the for-loop in Algorithm 1, the accumulator acc contains a GLWE encryption of $X^{-\tilde{\mu}^*} \cdot \bar{v}$ under key $\mathfrak{s}'$.

Proposition 1. *Algorithm 1 is correct.*

Proof. First, at initialization, acc contains $(0,\ldots,0,X^{-\tilde{b}} \cdot \bar{v}) \in \mathbb{Z}_{N,q}[X]^{k+1}$, which is a valid GLWE encryption of $X^{-\tilde{b}} \cdot \bar{v}$. We so have $\bar{\boldsymbol{c}}'_\mathbf{0} := (0,\ldots,0,X^{-\tilde{b}} \cdot \bar{v}) = \mathrm{GLWE}_{\mathfrak{s}'}(X^{-\tilde{b}} \cdot \bar{v})$. Next, by induction, we assume that the result is correct for $i = j-1$. We must prove that it remains correct for $i = j$. We have:

$$\begin{aligned}
\overline{\boldsymbol{c}}'_{\boldsymbol{j}} &:= \mathrm{CMux}(\mathsf{bsk}[j], \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}}, X^{\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}}) \\
&= \begin{cases} \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} & \text{if } s_j = 0 \\ X^{\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} & \text{if } s_j = 1 \end{cases} \\
&= X^{s_j\,\tilde{a}_j} \cdot \overline{\boldsymbol{c}}'_{\boldsymbol{j-1}} \\
&= X^{s_j\,\tilde{a}_j} \cdot \mathrm{GLWE}_{\mathfrak{s}'}(X^{-\tilde{b}+\sum_{i=1}^{j-1} s_i\,\tilde{a}_i} \cdot \overline{v}) = \mathrm{GLWE}_{\mathfrak{s}'}(X^{-\tilde{b}+\sum_{i=1}^{j} s_i\,\tilde{a}_i} \cdot \overline{v})
\end{aligned}$$

which proves the correctness of Algorithm 1. □

3 Using Ternary Keys

As seen in the above proof, the bootstrapping for the TFHE family crucially makes use of the identity

$$\begin{aligned}
X^{s_j\,\tilde{a}_j} &= \begin{cases} 1 & \text{if } s_j = 0 \\ X^{\tilde{a}_j} & \text{if } s_j = 1 \end{cases} \\
&= s_j\,(X^{\tilde{a}_j} - 1) + 1
\end{aligned}$$

for $s_j \in \{0, 1\}$. It is therefore inherently restricted to binary keys. This section exposes two different strategies to extend TFHE and the likes to ternary keys.

3.1 Micciancio–Polyakov's Approach

In [15], Micciancio and Polyakov astutely notice that any vector $\boldsymbol{s} = (s_1, \ldots, s_n)$ with ternary entries $s_j \in \{0, 1, -1\}$, $1 \le j \le n$, can be expressed as the difference of two binary vectors $\boldsymbol{s^1} = (s_1^1, \ldots, s_n^1)$ and $\boldsymbol{s^2} = (s_1^2, \ldots, s_n^2) \in \mathbb{B}^n$:

$$(s_1, \ldots, s_n) = (s_1^1, \ldots, s_n^1) - (s_1^2, \ldots, s_n^2)$$

where, for $1 \le j \le n$,

$$(s_j^1, s_j^2) = \begin{cases} (0,0) & \text{if } s_j = 0 \\ (1,0) & \text{if } s_j = 1 \\ (0,1) & \text{if } s_j = -1 \end{cases} .$$

The number of bootstrapping keys is doubled and is equal to $2n$. They are given by

$$\mathsf{bsk}[2(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{s}'}(s_j^i)$$

for all $1 \le j \le n$ and for all $1 \le i \le 2$. With the previous notation, the authors of [15] exploit the multiplicative nature of the bootstrapping. The GLWE encryption of $X^{-\tilde{b}+\sum_{j=1}^{n} s_j\,\tilde{a}_j} \cdot \overline{v} = \left(\prod_{j=1}^{n} X^{s_j^1\,\tilde{a}_j}\,(X^{-1})^{s_j^2\,\tilde{a}_j}\right) \cdot (X^{-\tilde{b}} \cdot \overline{v})$ is then obtained iteratively as

$$\begin{cases} \bar{\boldsymbol{c}}'_{\mathbf{2j-1}} \leftarrow \mathrm{CMux}(\mathsf{bsk}[2j-1], \bar{\boldsymbol{c}}'_{\mathbf{2j-2}}, X^{\tilde{a}_j} \cdot \bar{\boldsymbol{c}}'_{\mathbf{2j-2}}) \\ \bar{\boldsymbol{c}}'_{\mathbf{2j}} \leftarrow \mathrm{CMux}(\mathsf{bsk}[2j], \bar{\boldsymbol{c}}'_{\mathbf{2j-1}}, X^{-\tilde{a}_j} \cdot \bar{\boldsymbol{c}}'_{\mathbf{2j-1}}) \end{cases}$$

for $j = 1, \ldots, n$, with $\bar{\boldsymbol{c}}'_{\mathbf{0}} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \bar{v})$. Algorithmically, we have:

Algorithm 2: Blind rotation (ternary case).

```
1 $\mathsf{acc} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \bar{v})$
2 for $j = 1$ to $n$ do
3     $\mathsf{acc} \leftarrow \mathsf{acc} + \mathsf{bsk}[2j-1] \boxdot \big((X^{\tilde{a}_j} - 1) \cdot \mathsf{acc}\big)$
4     $\mathsf{acc} \leftarrow \mathsf{acc} + \mathsf{bsk}[2j] \boxdot \big((X^{-\tilde{a}_j} - 1) \cdot \mathsf{acc}\big)$
5 end for
6 return acc
```

With ternary keys, the evaluation of $\mathrm{GLWE}_{\mathfrak{s}'}\big(X^{-\tilde{b}+\sum_{j=1}^{n} s_j \tilde{a}_j} \cdot \bar{v}\big)$ thus involves $\underline{2n \text{ external products}}$. For a same value of n, this is twice more than in the binary case.

3.2 Proposed Approach

The blind rotation with ternary keys of Algorithm 2 can be largely improved. Instead of a multiplicative split, we observe that the monomial $X^{s_j \tilde{a}_j} = X^{(s_j^1 - s_j^2) \tilde{a}_j}$ with $s_j^1, s_j^2 \in \{0, 1\}$ can be expressed in an *additive* way as

$$X^{s_j \tilde{a}_j} = s_j^1 (X^{\tilde{a}_j} - 1) + s_j^2 (X^{-\tilde{a}_j} - 1) + 1 \ . \tag{1}$$

Remark 1. It is interesting to note that Eq. (1) can be equivalently expressed as $X^{s_j \tilde{a}_j} = (1 - s_j^1 - s_j^2) + s_j^1 X^{\tilde{a}_j} + s_j^2 X^{-\tilde{a}_j} = (1 - s_j^1 - s_j^1) X^{0 \cdot \tilde{a}_j} + s_j^1 X^{1 \cdot \tilde{a}_j} + s_j^2 X^{(-1) \cdot \tilde{a}_j}$.

As in Sect. 3.1, we define the $2n$ bootstrapping keys $\mathsf{bsk}[2(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{s}'}(s_j^i)$ for all $1 \le j \le n$ and for all $1 \le i \le 2$. An application of the GGSW encryption to the above relation (1) leads to

$$\mathrm{GGSW}_{\mathfrak{s}'}(X^{s_j \tilde{a}_j}) \leftarrow \\ (X^{\tilde{a}_j} - 1)\,\mathsf{bsk}[2j-1] + (X^{-\tilde{a}_j} - 1)\,\mathsf{bsk}[2j] + \mathrm{GGSW}_{\mathfrak{s}'}(1) \ .$$

We therefore obtain a new algorithm for the blind rotation. It is given in Algorithm 3.

Interestingly, the calculation of a blind rotation with ternary keys using Algorithm 3 only requires $\underline{n \text{ external products}}$.

Algorithm 3: Blind rotation (ternary case), revisited.

```
1 $\mathsf{acc} \leftarrow (0, \ldots, 0, X^{-\tilde{b}} \cdot \overline{v})$
2 for $j = 1$ to $n$ do
3     $\mathsf{acc} \leftarrow \mathsf{acc} + \big((X^{\tilde{a}_j} - 1)\,\mathsf{bsk}[2j-1] + (X^{-\tilde{a}_j} - 1)\,\mathsf{bsk}[2j]\big) \boxdot \mathsf{acc}$
4 end for
5 return acc
```

4 Extensions and Generalizations

4.1 Higher Radices

The proposed approach can be extended to support arbitrary formats of keys.

For full generality, we suppose that the keys are drawn from an arbitrary set $\mathcal{S}$ (e.g., $\mathcal{S} = \{0, 1, -1\}$ for ternary keys). We let $m = \#\mathcal{S}$ denote the cardinality of $\mathcal{S}$. The monomial $X^{s_j\,\tilde{a}_j}$ can be written additively as

$$X^{s_j\,\tilde{a}_j} = \sum_{t \in \mathcal{S}} \underbrace{\mathbb{1}\{t = s_j\}}_{:=\sigma_{j,t}} X^{t\,\tilde{a}_j} \quad . \tag{2}$$

In the above equation, $\mathbb{1}$ denotes the predicate function (*i.e.*, $\mathbb{1}\{t = s_j\} = 1$ when $t = s_j$ and $\mathbb{1}\{t = s_j\} = 0$ otherwise).

We define the set $I = \{0, \ldots, m-1\}$. We also define the "alphabet" vector $\boldsymbol{A} \in \mathcal{S}^m$ whose components are the different elements of $\mathcal{S}$. For example, back to ternary keys, we have $m = 3$ and write $\boldsymbol{A} = (0, 1, -1)$ so that $\boldsymbol{A}[0] = 0$, $\boldsymbol{A}[1] = 1$ and $\boldsymbol{A}[2] = -1$. Vector $\boldsymbol{A}$ gives rise to function

$$\mathsf{A} \colon I \to \mathcal{S}, i \mapsto \mathsf{A}(i) = \boldsymbol{A}[i]$$

taking on input an index $i \in I$ and returning the i^{th} component[2] of $\boldsymbol{A}$. Equation (2) can therefore be equivalently rewritten as

$$X^{s_j\,\tilde{a}_j} = \sum_{i=0}^{m-1} \mathbb{1}\{\mathsf{A}(i) = s_j\}\, X^{\mathsf{A}(i)\,\tilde{a}_j} \quad . \tag{3}$$

The first digit in the alphabet vector is $\tau_0 := \boldsymbol{A}[0] = \mathsf{A}(0)$. As the predicate function $\mathbb{1}\{\mathsf{A}(i) = s_j\}$ is true for exactly one index $i \in \{0, \ldots, m-1\}$, we deduce from Eq. (3) that

$$X^{s_j\,\tilde{a}_j} = X^{\mathsf{A}(0)\,\tilde{a}_j} + \sum_{i=1}^{m-1} \mathbb{1}\{\mathsf{A}(i) = s_j\}\left(X^{\mathsf{A}(i)\,\tilde{a}_j} - X^{\mathsf{A}(0)\,\tilde{a}_j}\right) \quad . \tag{4}$$

Interestingly, compared to Eq. (3), the formulation of Eq. (4) involves only $m-1$ predicate evaluations.

[2] Starting at $i = 0$.

Hence, defining $n(m-1)$ bootstrapping keys $\mathsf{bsk}[(m-1)(j-1)+i] \leftarrow \mathrm{GGSW}_{\mathfrak{s}'}(\sigma_{j,t})$ with $\sigma_{j,t} = \mathbb{1}\{t = s_j\}$ and $t = \mathsf{A}(i)$, for all $1 \le j \le n$ and for all $1 \le i \le m-1$, we obtain Algorithm 4. Somewhat surprisingly, the number of external products remains equal to n. For a given security level, we note the value of n is a decreasing function of m; see Appendix A.

Algorithm 4: Blind rotation (higher-radix case).

```
1 $\mathsf{acc} \leftarrow (0,\dots,0,X^{-\tilde{b}}\cdot\overline{v})$
2 for $j = 1$ to $n$ do
3   | $\mathsf{acc} \leftarrow X^{\mathsf{A}(0)\,\tilde{a}_j}\cdot\mathsf{acc} + \big(\sum_{i=1}^{m-1}(X^{\mathsf{A}(i)\,\tilde{a}_j} - X^{\mathsf{A}(0)\,\tilde{a}_j})\,\mathsf{bsk}[(m-1)(j-1)+i]\big)\boxdot\mathsf{acc}$
4 end for
5 return acc
```

Remark 2. If the digit alphabet $\mathcal{S}$ contains the digit 0, it is advantageous to set the first digit τ_0 to 0 (and thus $\mathsf{A}(0) = 0$) so that Line 3 in Algorithm 4 simplifies to

$$\mathsf{acc} \leftarrow \mathsf{acc} + \Big(\sum_{i=1}^{m-1}(X^{\mathsf{A}(i)\,\tilde{a}_j} - 1)\,\mathsf{bsk}[(m-1)(j-1)+i]\Big)\boxdot\mathsf{acc}$$

4.2 Multi-digit Approach

The number of external products can be further decreased. We combine and extend the multi-bit approach of [4,17] to the case of a secret key expressed as a series of digits in a higher radix. The idea is to process several digits concurrently.

In the previous section, the monomial $X^{s_j\,\tilde{a}_j}$ is additively expressed under the form $X^{s_j\,\tilde{a}_j} = \sum_{t\in\mathcal{S}}\sigma_{j,t}\cdot X^{t\,\tilde{a}_j}$ with $\sigma_{j,t} = \mathbb{1}\{t = s_j\}$. Let $\tau_0 = \boldsymbol{A}[0] = \mathsf{A}(0)$ denote the first digit in the alphabet vector. With two digits, we can express the corresponding monomial as

$$\begin{aligned}
&X^{s_{j_1}\,\tilde{a}_{j_1} + s_{j_2}\,\tilde{a}_{j_2}} \\
&\quad= X^{s_{j_1}\,\tilde{a}_{j_1}}\,X^{s_{j_2}\,\tilde{a}_{j_2}} \\
&\quad= \Big(\sum_{t\in\mathcal{S}}\sigma_{j_1,t}\cdot X^{t\,\tilde{a}_{j_1}}\Big)\Big(\sum_{t\in\mathcal{S}}\sigma_{j_2,t}\cdot X^{t\,\tilde{a}_{j_2}}\Big) \\
&\quad= \sum_{t_1\in\mathcal{S}}\sum_{t_2\in\mathcal{S}}\sigma_{j_1,t_1}\sigma_{j_2,t_2}\cdot X^{t_1\,\tilde{a}_{j_1}}X^{t_2\,\tilde{a}_{j_2}} \\
&\quad= X^{\tau_0(\tilde{a}_{j_1}+\tilde{a}_{j_2})} + \sum_{(t_1,t_2)\in\mathcal{S}^2\setminus\{(\tau_0,\tau_0)\}}\sigma_{j_1,t_1}\sigma_{j_2,t_2}\cdot\Big(X^{t_1\,\tilde{a}_{j_1}+t_2\,\tilde{a}_{j_2}} - X^{\tau_0(\tilde{a}_{j_1}+\tilde{a}_{j_2})}\Big)\ .
\end{aligned}$$

More generally, for d digits, we get

$$X^{\sum_{l=1}^{d} s_{j_l}\,\tilde{a}_{j_l}} = X^{\tau_0 \sum_{l=1}^{d} \tilde{a}_{j_l}} + \sum_{(t_1,\ldots,t_d)\in\mathcal{S}^d\setminus\{(\tau_0,\ldots,\tau_0)\}} \left(\bigwedge_{l=1}^{d} \sigma_{j_l,t_l}\right)\cdot\left(X^{\sum_{l=1}^{d} t_l\,\tilde{a}_{j_l}} - X^{\tau_0\sum_{l=1}^{d}\tilde{a}_{j_l}}\right) .$$

To ease the presentation, we henceforth assume that $d \mid n$. We can write $\sum_{j=1}^{n} s_j\,\tilde{a}_j = \sum_{h=1}^{n/d}\sum_{l=1}^{d} s_{(h-1)d+l}\,\tilde{a}_{(h-1)d+l}$. We define $\frac{n}{d}(m^d-1)$ bootstrapping keys

$$\mathsf{bsk}[(m^d-1)(h-1)+\textstyle\sum_{l=1}^{d} i_l\,m^{l-1}] \leftarrow \mathrm{GGSW}_{\mathfrak{s}'}\left(\bigwedge_{l=1}^{d}\sigma_{(h-1)d+l,t_l}\right)$$

with $(t_1,\ldots,t_d) = (\mathsf{A}(i_1),\ldots,\mathsf{A}(i_d))$, for all $1\le h\le n/d$ and for all $(i_1,\ldots,i_d)\in\{0,\ldots,m-1\}^d$ with $(i_1,\ldots,i_d)\ne(0,\ldots,0)$. The number of external products decreases to n/d.

Algorithm 5: Blind rotation (multi-digit case).

```
1 $\mathsf{acc} \leftarrow (0,\ldots,0,X^{-\tilde{b}}\cdot\bar{\boldsymbol{v}})$
2 for $h = 1$ to $n/d$ do
3     $\mathsf{acc} \leftarrow X^{\mathsf{A}(0)\sum_{l=1}^{d}\tilde{a}_{(h-1)d+l}}\cdot\mathsf{acc} + \Big(\sum_{\substack{0\le i_1,\ldots,i_d\le m-1\\(i_1,\ldots,i_d)\ne(0,\ldots,0)}}\big(X^{\sum_{l=1}^{d}\mathsf{A}(i_l)\,\tilde{a}_{(h-1)d+l}} - X^{\mathsf{A}(0)\sum_{l=1}^{d}\tilde{a}_{(h-1)d+l}}\big)\,\mathsf{bsk}[(m^d-1)(h-1)+\sum_{l=1}^{d} i_l\,m^{l-1}]\Big)\boxdot\mathsf{acc}$
4 end for
5 return acc
```

Remark 3. Again, similarly to Remark 2, when digit $0\in\mathcal{S}$, setting $\boldsymbol{A}[0] = \mathsf{A}(0) = 0$, Line 3 in Algorithm 5 simplifies and becomes

$$\mathsf{acc}\leftarrow\mathsf{acc} + \Big(\sum_{\substack{0\le i_1,\ldots,i_d\le m-1\\(i_1,\ldots,i_d)\ne(0,\ldots,0)}}\big(X^{\sum_{l=1}^{d}\mathsf{A}(i_l)\,\tilde{a}_{(h-1)d+l}} - 1\big)\,\mathsf{bsk}[(m^d-1)(h-1) + \textstyle\sum_{l=1}^{d} i_l\,m^{l-1}]\Big)\boxdot\mathsf{acc}$$

5 Performance Analysis and Experiments

In this section, we analyze the performance of our extended blind rotation. Our efficiency comparisons are based on a C library called `zlib` specifically designed to conduct experiments on TFHE. Lightweight and modular, the purpose of `zlib` is to finely tune the bootstrapping procedure, establish the performance profile of various algorithmic strategies on CPUs, and see how they compare depending on their parameters.

5.1 LWE Estimator for Security Estimates

The security of LWE encryption depends on its parameters in a way that is dictated by the current state-of-the-art attacks on LWE. We rely on the LWE Estimator [1][3] to enforce a desired security level. Given an LWE dimension n, a secret key uniformly drawn from $\mathcal{S}^n$ with $\#\mathcal{S} = m$, a ciphertext precision p and a noise variance v, this tool provides a security estimate

$$\lambda = \mathsf{LWE\text{-}security}(n, m, p, v)$$

against an IND-CPA adversary to whom an unbounded number of encryptions of zero are given. The $\mathsf{LWE\text{-}security}$ function is also valid for GLWE and GGSW ciphertexts when n is replaced with kN.

5.2 Nominal Setting

Our nominal setting, and basis for comparison, is set to $(d, m) = (1, 2)$. This means that the LWE secret key embedded in the bootstrapping key is a usual (non-sparse) binary key $s \in \{0, 1\}^n$ and each element of the bootstrapping key is a GGSW encryption of bit s_j, $1 \leq j \leq n$.

FFT-Based External Product. Each external product is performed as follows. The input accumulator is a GLWE ciphertext; *i.e.*, a vector of $k + 1$ torus polynomials modulo $X^N + 1$ with coefficients of p_{acc} bits. We fix $k = 1$, $N = 1024$ and $p_{acc} = 64$. Each torus polynomial is decomposed into ℓ integer polynomials with coefficients in $\{-B/2, \ldots, B/2\}$ where we pick $\ell = 3$ and $B = 2^\beta = 2^8$. We then apply a radix-2 negacyclic FFT to every one of these integer polynomials, resulting in a matrix of $\ell(k+1)$ complex arrays in $\mathbb{C}^{N/2}$. We easily compute the tensor

$$T = (X^{\tilde{a}_j} - 1) \cdot \mathsf{bsk}[j]$$

on the fly since the FFT coefficients of $X^{\tilde{a}_j}$ are just cyclic powers of $e^{i\pi/N}$ and can be derived from the twiddle factors of the FFT that are already precomputed and stored. Applying a complex matrix-tensor product then gives us $k+1$ arrays in $\mathbb{C}^{N/2}$, which are converted back to torus polynomials in the standard domain by applying a radix-2 backward FFT and rounding. The $k + 1$ polynomials are then added up to the accumulator modulo $2^{p_{acc}} = 2^{64}$, which gives the output value of the accumulator.

Running Time. Instrumenting `zlib` with this setting allows us to measure the average number of clock cycles $\mathrm{time}^{\mathrm{XP}}(1, 2)$ required to perform one external product on a reference architecture. The total running time of the blind rotation is then

$$\mathrm{time}^{\mathrm{BR}}(1, 2) = n \cdot \mathrm{time}^{\mathrm{XP}}(1, 2)$$

where we fix $n = 640$. This particular setting for $n, N, k, \ell, \beta, p_{acc}$ complies with a parameter set often used in implementations of TFHE.

[3] Available at https://bitbucket.org/malb/lwe-estimator/src/master/.

Output Variance. We neglect the approximation errors that are due to the use of 64-bit floating-point numbers in FFT conversions and operations in the FFT domain. One can show that one external product increases the variance of the accumulated noise by

$$\mathrm{var}^{\mathrm{XP}}(1,2) = (k+1) \cdot N \cdot \mathbb{M}_2(\ell, B) \cdot \mathrm{var}^{\mathrm{bsk}}$$

where the term

$$\mathbb{M}_2(\ell, B) = \ell \cdot \frac{(B+2)(B^2-B+1)}{12 \cdot (B+1)} + (1 - (-\tfrac{1}{B})^{\ell}) \cdot \frac{B^2}{4 \cdot (B+1)^2}$$

is the second statistical moment of the coefficients of the decomposed integer polynomials right before their conversion to the FFT domain. Finally, $\mathrm{var}^{\mathrm{bsk}}$ is the noise variance of the GGSW ciphertexts that compose the bootstrapping key. We fix the overall security level to $\lambda = 128$ and find $\mathrm{var}^{\mathrm{bsk}}$ by solving

$$\mathsf{LWE\text{-}security}(kN, p_{acc}, 2, \mathrm{var}^{\mathrm{bsk}}) = 128$$

which yields $\mathrm{var}^{\mathrm{bsk}} = 2^{-50.32}$. The output variance of the blind rotation is then

$$\mathrm{var}^{\mathrm{BR}}(1,2) = n \cdot \mathrm{var}^{\mathrm{XP}}(1,2) \;.$$

5.3 Extended Setting

Keeping the same parameters as in the nominal setting, we now generalize it to arbitrary pairs (d, m) with $d \geq 1$ and $m \geq 2$.

Extended External Product. In comparison with the nominal setting, the only adaptation in the external product is that the tensor

$$T = (X^{\tilde{a}_j} - 1) \cdot \mathsf{bsk}[j]$$

is now generalized to

$$T = \sum_{\substack{0 \leq i_1, \ldots, i_d \leq m-1 \\ (i_1, \ldots, i_d) \neq (0, \ldots, 0)}} (X^{\sum_{l=1}^{d} \mathsf{A}(i_l)\, \tilde{a}_{(h-1)d+l}} - 1)\, \mathsf{bsk}[(m^d - 1)(h-1) + \textstyle\sum_{l=1}^{d} i_l\, m^{l-1}]$$

which requires $m^d - 1$ operations of the form $(X^{\alpha} - 1) \cdot \mathsf{bsk}[j]$ instead of just one.

Running Time. We measure the average number of clock cycles $\mathrm{time}^{\mathrm{XP}}(d, m)$ using `zlib` and find that

$$\mathrm{time}^{\mathrm{XP}}(d, m) = \big(1 + \big(m^d - 2\big) \cdot \varDelta\big) \cdot \mathrm{time}^{\mathrm{XP}}(1, 2)$$

with $\varDelta \approx 0.1557$.

Output Variance. The noise variance added by an external product is now

$$\mathrm{var}^{\mathrm{XP}}(d,m) = (k+1)\cdot N \cdot \mathbb{M}_2(\ell,B)\cdot \big(m^d-1\big)\cdot \mathrm{var}^{\mathrm{bsk}} = \big(m^d-1\big)\cdot \mathrm{var}^{\mathrm{XP}}(1,2)$$

and the output variance of the blind rotation is

$$\mathrm{var}^{\mathrm{BR}}(d,m) = \frac{n}{d}\cdot \mathrm{var}^{\mathrm{XP}}(d,m)$$

However, the value of n can now be decreased slightly due to $m \geq 2$. Indeed, $n = 640$ was chosen to verify

$$\textsf{LWE-security}(n,2,64,v) = 128$$

for a certain noise variance v, whereas we now require

$$\textsf{LWE-security}(n,m,64,v) = 128$$

for the same variance v in our extended setting. Based on the data-points collected from LWE Estimator and given in Appendix A, we experimentally find replacement values for $n = n(m)$ as shown on Table 1.

Table 1. Optimal values of n as a function of $m \in \{2,\ldots,10\}$ for 128-bit security.

m	2	3	4	5	6	7	8	9	10
$n(m)$	640	610	591	579	569	561	555	549	544

5.4 Finding Optimal Settings

Putting it all together, our generalization gives the following ratios:

$$\frac{\mathrm{var}^{\mathrm{BR}}(d,m)}{\mathrm{var}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{m^d-1}{d}\,, \tag{5}$$

$$\frac{\mathrm{time}^{\mathrm{BR}}(d,m)}{\mathrm{time}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{1+\big(m^d-2\big)\cdot\mathit{\Delta}}{d}\,, \qquad \mathit{\Delta}\approx 0.1557 \tag{6}$$

$$\frac{\mathrm{keysize}^{\mathrm{BR}}(d,m)}{\mathrm{keysize}^{\mathrm{BR}}(1,2)} = \frac{n(m)}{n(2)}\cdot\frac{m^d-1}{d}\,. \tag{7}$$

We see that the ratios (5) and (7) are identical and we denote them by 2^u, while the ratio (6) is denoted 2^v. A given (u,v) pair indicates that a degradation of the output variance and key size by a factor 2^u yields a speedup factor of 2^{-v}.

What we are after is to find the values of (d, m) that provide the most interesting (u, v) pairs, namely, ones where both u and v are minimized.

Due to the nature of these formulas, we easily see that no optimum exists for (d, m) that would simultaneously minimize the key size on one hand and the running time on the other. Instead, we take the set S of pairs (u, v) derived from all the possible settings (d, m) where $d \in \{1, \ldots, 10\}$ and $m \in \{2, \ldots, 10\}$, and eliminate from S the dominated points; *i.e.*, pairs $(u_2, v_2) \in S$ such that there exists $(u_1, v_1) \in S$ verifying

$$(u_1 \leq u_2 \text{ and } v_1 < v_2) \text{ or } (u_1 < u_2 \text{ and } v_1 \leq v_2) \ .$$

Dominated points can be safely eliminated since they are strictly less efficient than other reachable points. We end up with a subset $S_{\text{best}} \subseteq S$ of non-dominated points known as the Pareto front of S. The elements of S_{best} identify the best possible trade-offs among all possible settings, and are displayed on Fig. 1.

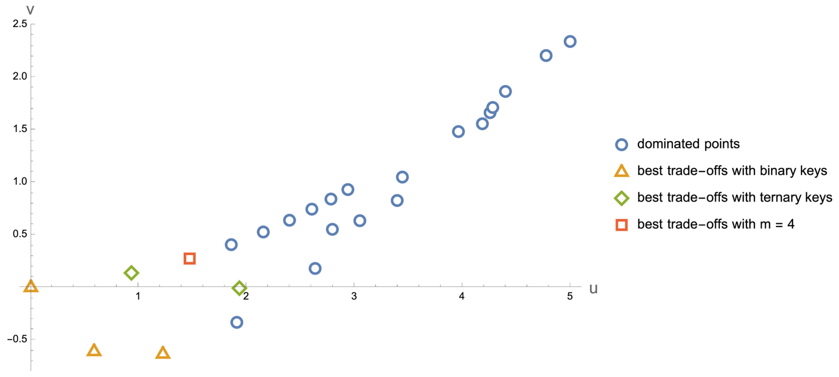


Fig. 1. Points (u, v) obtained from screening m through $\{2, \ldots, 10\}$ and d through $\{1, \ldots, 10\}$, and their Pareto front (orange triangles). Other points exist outside of the displayed range but they are all dominated. The nominal parameters of the blind rotation are $N = 1024$, $k = 1$, $\ell = 3$, $\beta = 8$ and $n = 640$. (Color figure online)

To conclude our experiments, we see that in addition to the nominal setting that we have chosen ($u = v = 0$), two other trade-off points of interest appear:

– $\boldsymbol{(d, m) = (2, 2)}$, for which a 50% increase in key size and output variance increases speed by 52.5%, and
– $\boldsymbol{(d, m) = (3, 2)}$, for which a 133% increase in key size and output variance increases speed by 55.1%.

Should one move away from binary keys for security reasons, the next best trade-off points become

– $\boldsymbol{(d, m) = (1, 3)}$, for which a 91% increase in key size and output variance decreases speed by 9.22%, and
– $\boldsymbol{(d, m) = (2, 3)}$, for which a 281% increase in key size and output variance increases speed by 0.4%.

This assumes that the security estimates for ternary keys and $m \geq 4$ are left untouched by an alleged attack against binary keys.

6 Conclusion

This paper adapted the blind rotation as used in the bootstrapping in FHEW or TFHE to support ternary keys. The resulting implementation is about twice faster than a previous method by Micciancio and Polyakov and with the same memory requirements for the bootstrapping keys as in their method. We also extended the proposed approach to higher radices and generalized it by processing several digits at a time—at the expense of further memory requirements. An analysis of the various trade-offs provided by the choice of the radix and of the number of processed digits was provided. In particular, useful trade-offs include processing 2 or 3 bits concurrently for binary keys and 1 or 2 digits for ternary keys.

Acknowledgements. We are grateful to Ben Curtis for his help in compiling Tables 2 and 3. We are also grateful to the anonymous referees for useful comments.

A Tables

Although for higher radices the number of external products remains equal to n, the value of n is a decreasing function of m. Playing with LWE Estimator, at a security level of 128 bits, we get the following tables.

Table 2. LWE dimension n as a function of m for different values for the noise standard deviation σ, for $q = 2^{32}$

σ	m								
	2	3	4	5	6	7	8	9	10
−30	1208	1176	1160	1152	1136	1128	1120	1120	1112
−29	1168	1144	1120	1112	1104	1088	1088	1080	1072
−28	1136	1104	1088	1072	1064	1056	1048	1040	1040
−27	1096	1064	1048	1032	1024	1016	1008	1000	1000
−26	1056	1024	1008	1000	984	976	968	968	960
−25	1016	992	968	960	952	944	936	928	920
−24	976	952	936	920	912	904	896	888	880
−23	944	912	896	880	872	864	856	848	848
−22	904	872	856	848	832	824	816	816	808
−21	864	840	816	808	800	784	784	776	768
−20	824	800	784	768	760	752	744	736	728
−19	792	760	744	728	720	712	704	696	696
−18	752	720	704	688	680	672	664	664	656
−17	712	680	664	656	640	632	632	624	616
−16	672	648	624	616	608	600	592	584	576
−15	640	608	592	576	568	560	552	544	544
−14	600	568	552	536	528	520	512	512	504
−13	560	528	512	496	488	480	472	472	464
−12	528	496	472	464	456	448	440	432	424
−11	488	456	432	424	416	408	400	392	392
−10	448	416	400	384	376	368	360	352	352
−9	408	376	360	344	336	328	320	320	312
−8	376	336	320	312	296	288	288	280	272
−7	336	304	280	272	256	256	248	240	240
−6	296	264	240	232	224	216	208	208	200

Table 3. LWE dimension n as a function of m for different values for the noise standard deviation σ, for $q = 2^{64}$

σ	m								
	2	3	4	5	6	7	8	9	10
−62	2424	2392	2376	2368	2352	2344	2336	2336	2328
−61	2384	2360	2336	2328	2320	2304	2304	2296	2288
−60	2344	2320	2304	2288	2280	2272	2264	2256	2248
−59	2304	2280	2264	2248	2240	2232	2224	2216	2208
−58	2272	2240	2224	2208	2200	2192	2184	2176	2176
−57	2232	2200	2184	2176	2160	2152	2144	2144	2136
−56	2192	2168	2152	2136	2128	2120	2112	2104	2096
−55	2152	2128	2112	2096	2088	2080	2072	2064	2056
−54	2120	2088	2072	2056	2048	2040	2032	2024	2024
−53	2080	2056	2032	2024	2016	2000	2000	1992	1984
−52	2048	2016	2000	1984	1976	1968	1960	1952	1952
−51	2008	1976	1960	1952	1936	1928	1920	1912	1912
−50	1968	1944	1920	1912	1896	1888	1880	1880	1872
−49	1928	1904	1888	1872	1864	1856	1848	1840	1832
−48	1888	1864	1848	1832	1824	1816	1808	1800	1792
−47	1856	1824	1808	1792	1784	1776	1768	1760	1760
−46	1816	1784	1768	1760	1744	1736	1728	1728	1720
−45	1776	1752	1728	1720	1712	1696	1696	1688	1680
−44	1736	1712	1696	1680	1672	1664	1656	1648	1640
−43	1704	1672	1656	1640	1632	1624	1616	1608	1608
−42	1664	1640	1616	1608	1600	1592	1584	1576	1568
−41	1624	1600	1584	1568	1560	1552	1544	1536	1528
−40	1592	1560	1544	1528	1520	1512	1504	1496	1496
−39	1552	1520	1504	1488	1480	1472	1464	1456	1456
−38	1512	1480	1464	1456	1440	1432	1424	1424	1416
−37	1472	1448	1424	1416	1408	1400	1392	1384	1376
−36	1432	1408	1392	1376	1368	1360	1352	1344	1336
−35	1400	1368	1352	1336	1328	1320	1312	1304	1304
−34	1360	1336	1312	1304	1296	1280	1280	1272	1264
−33	1320	1296	1280	1264	1256	1248	1240	1232	1224
−32	1288	1256	1240	1224	1216	1208	1200	1192	1192
−31	1248	1216	1200	1184	1176	1168	1160	1152	1152
−30	1208	1176	1160	1152	1136	1128	1120	1120	1112
−29	1168	1144	1120	1112	1104	1088	1088	1080	1072
−28	1136	1104	1088	1072	1064	1056	1048	1040	1040
−27	1096	1064	1048	1032	1024	1016	1008	1000	1000

(*continued*)

Table 3. (*continued*)

σ	m								
	2	3	4	5	6	7	8	9	10
−26	1056	1024	1008	1000	984	976	968	968	960
−25	1016	992	968	960	952	944	936	928	920
−24	976	952	936	920	912	904	896	888	880
−23	944	912	896	880	872	864	856	848	848
−22	904	872	856	848	832	824	816	816	808
−21	864	840	816	808	800	784	784	776	768
−20	824	800	784	768	760	752	744	736	728
−19	792	760	744	728	720	712	704	696	696
−18	752	720	704	688	680	672	664	664	656
−17	712	680	664	656	640	632	632	624	616
−16	672	648	624	616	608	600	592	584	576
−15	640	608	592	576	568	560	552	544	544
−14	600	568	552	536	528	520	512	512	504
−13	560	528	512	496	488	480	472	472	464
−12	528	496	472	464	456	448	440	432	424
−11	488	456	432	424	416	408	400	392	392
−10	448	416	400	384	376	368	360	352	352
−9	408	376	360	344	336	328	320	320	312
−8	376	336	320	312	296	288	288	280	272
−7	336	304	280	272	256	256	248	240	240
−6	296	264	240	232	224	216	208	208	200

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
2. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 1–20. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_1
3. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_17
4. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 483–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_17
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3):13:1–13:36 (2014). https://doi.org/10.1145/2633600. Earlier version in ITCS 2012

6. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) 5th Innovations in Theoretical Computer Science (ITCS 2014), pp. 1–12. ACM Press (2014). https://doi.org/10.1145/2554797.2554799
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2019). https://doi.org/10.1007/s00145-019-09319-x
8. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) CSCML 2021. LNCS, vol. 12716, pp. 1–19. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78086-9_1
9. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24
10. Gama, N., Izabachène, M., Nguyen, P.Q., Xie, X.: Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 528–558. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_19
11. Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3), 97–105 (2010). https://doi.org/10.1145/1666420.1666444. Earlier version in STOC 2009
12. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_1
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
14. Joye, M.: Guide to fully homomorphic encryption over the [discretized] torus. Cryptology ePrint Archive, Report 2021/1402 (2021). https://ia.cr/2021/1402
15. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like cryptosystems. In: Brenner, M., et al. (eds.) 9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2021), pp. 17–28. ACM Press (2021). https://doi.org/10.1145/3474366.3486924
16. Rivest, R.L., Adleman, L., Detouzos, M.L.: On data banks and privacy homomorphisms. In: DeMillo, R.A., et al. (eds.) Foundations of Secure Computation, pp. 165–179. Academic Press (1978). https://people.csail.mit.edu/rivest/pubs.html#RAD78
17. Zhou, T., Yang, X., Liu, L., Zhang, W., Li, N.: Faster bootstrapping with multiple addends. IEEE Access **6**, 49868–49876 (2018). https://doi.org/10.1109/ACCESS.2018.2867655

Monitoring Time Series with Missing Values: A Deep Probabilistic Approach

Oshri Barazani[1] and David Tolpin[2](✉)

[1] PUB+, Beersheba, Israel
[2] Ben-Gurion University of the Negev, Beersheba, Israel
david.tolpin@gmail.com

Abstract. Systems are commonly monitored for health and security through collection and streaming of multivariate time series. Advances in time series forecasting due to adoption of multilayer recurrent neural network architectures make it possible to forecast in high-dimensional time series, and identify and classify novelties early, based on subtle changes in the trends. However, mainstream approaches to multi-variate time series predictions do not handle well cases when the ongoing forecasts must include uncertainty, nor they are robust to missing data. We introduce a new architecture for time series monitoring based on combination of state-of-the-art methods of forecasting in high-dimensional time series with full probabilistic handling of uncertainty. We demonstrate advantage of the architecture for time series forecasting and novelty detection, in particular with partially missing data, and empirically evaluate and compare the architecture to state-of-the-art approaches on a real-world data set.

1 Introduction

Modern information systems and operation environments are commonly monitored through collection and streaming of multivariate time series. The monitoring tasks comprise both forecasting, for planning of resource allocation and decision making, and novelty detection and characterization, for ensuring faultless functioning and early mitigation of failures and threats. Advances in time series forecasting due to adoption of multilayer recurrent neural network architectures made it possible to forecast in high-dimensional time series, and identify and classify novelties (anomalies) early, based on subtle changes in the trends. However, mainstream approaches to multi-variate time series modelling do not handle well cases when uncertainty is involved, either in the input, when some of the observations are missing, or in the output when the distribution of future observations, rather than their point values, is predicted. For forecast uncertainty modelling, stochastic latent variable variants of high-dimensional time series models where introduced, but so far have had to rely on sampling to account for uncertainty, limiting the performance of data handling. Imputation schemes were proposed for dealing with missing data, however, they do not generally give a satisfactory solution in presence of transient unavailability of some

https://doi.org/10.1007/978-3-031-07689-3_2

of the data sources (e.g. when a sensor stops working, or a transport channel malfunctions), which is a common case with monitoring of complex systems.

A systematic and theoretically founded approach to handling both input and output uncertainty would thus constitute a significant and welcome contribution to the theory and practice of monitoring of multivariate time series. It would also be highly desirable for such approach to facilitate efficient offline (learning) and online (inference) computations. In this ongoing research, we propose a deep learning architecture which uses a simple but powerful extension of traditional recurrent neural network (RNN) architecture which allows both

- to handle missing inputs in some or all of the components in a multivariate time series,
- and to accomplish multi-step probabilistic forecasting

in high-dimensional time series, paving a path to better decision making and finer and more robust anomaly detection and characterization. We evaluate the architecture on a real-world data set of multivariate time series collected from a network for cloud computing, and empirically demonstrate advantage of the proposed architecture over commonly used approaches.

2 Problem: Multivariate Time Series Forecasting

The core problem we address is forecasting in a multivariate time series. Formally, a *time series* is a matrix X of shape $T \times N$, where T is the number of time steps and N is the number of dimensions. The time steps are assumed to be equispaced. A k-step probabilistic *forecast* $\mathcal{F}_{tk}$ at time t is the belief distribution of time series $X_{t+1:t+k}$ for time steps $t+1...t+k$ given the observed time series $X_{1:t}$ for time steps $1...t$.

The forecasting is accomplished by applying model $\mathcal{M}_\theta$ parameterized by parameters θ to the observed time series:

$$\mathcal{F}_{tk} = \mathcal{M}_\theta(X_{1:t}) \tag{1}$$

The machine learning task is to devise θ^* that gives the best forecast, in terms of a certain loss function. A natural loss in the probabilistic setting is the average negative log likelihood of θ given a training data set $\mathcal{X}$ of multiple time series:

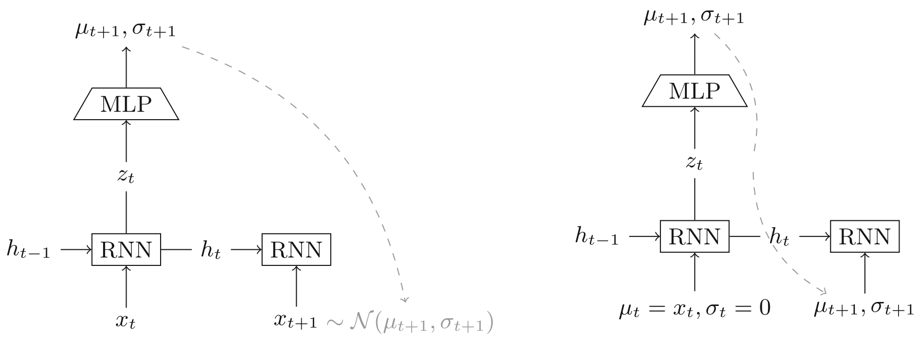
$$\theta^* = \arg\min_\theta \mathbb{E}_{X\in\mathcal{X},t\in 1...T-k}\left[-\log \Pr(X_{t+1:t+k}|M_\theta(X_{1:t}))\right] \tag{2}$$

When the model is differentiable by θ, the task is usually accomplished by performing a stochastic gradient loss minimization.

In the basic case, X is real-valued, $X \in \mathbb{R}^{T\times N}$. Here, we are interested in an extension of the basic case, in which some of the elements can be missing from X, that is $X \in (\mathbb{R} \cup \perp)^{T\times N}$.

3 Architecture: Recurrent Neural Network with Uncertainty Propagation

We introduce here a recurrent neural network architecture which facilitates uncertainty propagation. The architecture is capable both of handling missing values and of multi-step forecasting. We begin with description of conventional forecasting with RNNs. Then, we describe our proposed architecture as an extension to the conventional model.


μ_{t+1}, σ_{t+1}
MLP
z_t
h_{t-1}
RNN
h_t
RNN
x_t
$x_{t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1})$
μ_{t+1}, σ_{t+1}
MLP
z_t
h_{t-1}
RNN
h_t
RNN
$\mu_t = x_t, \sigma_t = 0$
μ_{t+1}, σ_{t+1}

(a) Conventional model

(b) Model with uncertainty propagation

Fig. 1. Time series models

3.1 Conventional Forecasting

A popular realization of the forecasting model $\mathcal{M}_\theta$ is a recurrent neural network (RNN), with θ corresponding to the network parameters. There is a range of neural recurrent models of varying complexity to deal with time series forecasting. Most models include a recurrent unit which threads the state through the time steps, accepts data as inputs and produces next step predictions as outputs. The simplest model is an RNN with a fully-connected readout layer to produce forecasts (Fig. 1a). RNN can be based on LSTM [12], GRU [8], or another architectural variant, and is often multi-layer. Architectures may also include intermediate modules, and sampling-based variational layers [10,20]. The overall architecture stays almost the same, with more connections, intermediate modules and sampling-based variational layers.

Input and Output. This architecture normally accepts observation vectors and outputs vectors of distribution parameters for the belief distribution of the observations at the next time step. In the simplest case, the network produces a single output for each input, that is the dimensions of the input and the output vector coincide. This corresponds to the assumption of homoskedasticity of epistemic noise, and either the mean squared error (corresponding to the Gaussian error distribution) or the mean absolute error (corresponding to the Laplace error distribution) is minimized.

More generally though, the epistemic noise is better modelled heteroskedastically, using a two-parameter loss distribution, with the location and the scale as the parameters. In the case of the frequently used normal (Gaussian) distribution, the output vector consists of means μ (location) and standard deviations σ (scale) of all dimensions and is twice as wide as the input.

Training. The model is trained to maximize probability of prediction. In the most basic case, called *out-of-sample one-step* forecasting, a single step is predicted for each time step in the series. In an n-step time series, steps $1...n-1$ are used as the input, and steps $1...n$ as the ground truth. Following (2), the network is trained to minimize negative log probability of true observations given the predicted belief distributions. More generally, a model can also be trained to predict more than a single step at once into the future, however this is rarely used in practice because the necessary size of the training data set grows exponentially with the prediction depth. Instead, future predictions are produced recurrently during forecasting.

Forecasting. Forecasting is accomplished by passing past observations through the model to obtain forecasts for the future time steps. In the *out-of-sample one-step* mode, a single step into the future is forecast. If a longer forecast is required, the current forecast is entered as the input at the next time step, time after time, up to the required length. Either the location (the point forecast) or a random sample from the belief distribution is used as the future input. Using random samples also allows to assess uncertainty multiple steps into the future: one can repeatedly sample from the belief distribution at each future step, and feed the sample as the input to the following step. Then, based on produced samples at future steps, one can estimate uncertainty intervals. Such Monte-Carlo handling of uncertainty is quite expensive computationally though, because the standard deviation of prediction error decreases as slowly as $\sqrt{N}$ with the number of samples N, on one hand, and uncertainty may, in general, grow exponentially with prediction depth, on the other hand.

Novelty Detection. Forecasts produced by the model can be used for a number of purposes, including decision making and, in particular, novelty (anomaly) detection. There are two related but different phenomena indicating a novelty in time series behavior:

1. Predicted volatility of the time series is high, that is, future observations can only be forecast uncertainly (with high variance).
2. Probability of actual observations, when observed, given a prediction from a past state, is low.

Either phenomenon, or both of them, can be used to alert about novelties in the time series. In recurrent neural network architectures, the hidden state (h_t in Fig. 1) can be used to identify and classify anomalies.

3.2 Forecasting with Uncertainty Propagation

The basic scheme outlined above poses difficulty in applications with high-dimensional time series and partially missing observations. Sampling based uncertainty assertion impacts performance, and missing observations are often imputed heuristically [15,17]. An architecture which incorporates confidence about data and in which observed and predicted data are interchangeable is highly desirable. For example, if out of 5 components 3 were measured and 2 predicted from an earlier step we want to input all of them into the next time step for further forecasting. In addition, the model architecture should be capable of robust uncertainty prediction and benefit from training with multiple steps of out-of-sample data.

Our proposed architecture is based on the observation that if (at least) the location and the scale are used to represent forecasts, an observation (that is, certain knowledge at a given step) can also be expressed using two parameters, by setting the location to the observation, and the scale to 0. For the normal distribution $\mathcal{N}(\mu, \sigma)$, the location and scale parameterization is straightforward, corresponding to μ and σ, however other belief distributions can be parameterized by location and scale as easily, e.g. the log-normal, Gamma, or Laplace distribution. For conciseness, we will confine further discussion to the case of independent normal belief distributions for each component; however, other distribution shapes can also be used. Based on this observation, **we propose the following extension** to the conventional RNN-based forecasting model (Fig. 1b):

1. *The input, as well as the output, is a vector of distribution parameters.* For the independent normal distributions, the distribution parameter vector consists of the means followed by the standard deviations. If the data has 5 components, the input will be 10-dimensional. For observed data—measurements present at the current time step—the standard deviation is zero. For missing data the input is the mean and the standard deviation as predicted from the preceding time steps.
2. Training can, in principle, be accomplished on data with missing values, but training on data with missing values incurs performance drawbacks and should be avoided. First, handling missing values and replacing them with early predictions introduces contingency in the forward run of the RNN and slows down significantly the execution during training. Second, missing values should, in general, themselves be viewed as anomalies. One must be able to handle them during inference, but should not rely on their presence in the training data.

   Therefore, we devise a scheme for *training our model on data that does not contain missing values.* Even in applications where missing values are common in inference, training data without missing values is usually readily available. However, since we introduce confidence into the input, we cannot train the network myopically, in out-of-sample one-step manner—the standard deviations in the input data will always be zero, and the network will never learn how to use them. To overcome this, we train on multiple predicted steps. We feed each prediction, without sampling, as input to the next step and compute the loss as negative log probability of this number of future points versus our prediction.

To illustrate, given the data set of 5 dimensions, the input has 10 dimensions. If we train with 3 time steps lookahead, the ground truth will be a matrix of size 3×5. The prediction against which the likelihood of this ground truth is computed will be a matrix of size 3×10. Intuitively, we would expect the predicted standard deviation to increase along the time axis for each component.

The ability of probabilistic forecasting with uncertainty, in the form of multivariate normal distributions, far into the future, opens opportunity for application to more robust novelty detection approaches. Instead of detecting novelty based on log probability of observations given predictions from the past [6], which is prone to false positives due to observation noise, novelties can be detected and analysed by comparing predictions of the same time point from different points in the past. In this case, KL-divergence between predictions provides a theoretically sound and robust mechanism for detection of anomalies, and is in particular relevant for monitoring of large operation environments with high dimensionality of time series and occasional missing values and heteroskedastic noise [2,18].

4 Case Study: Monitoring a Computer Cloud

We evaluate the proposed architecture on a data set of monitoring a cluster of 100 computing nodes in the cloud. For each node, the incoming and the outgoing network traffic (in bytes) and the CPU usage (relative) are logged with 1 min resolution. 240 h were logged, resulting in 12000 120-minute 3-dimensional samples. We split the dataset into the training, validation, and test as 80%, 10%, and 10% correspondingly. Since the original data set does not have many missing data points, we emulated data sets with missing data by randomly removing 5%, 10%, 20%, and 50% of the data.

We used a 3-layer GRU-based recurrent neural network with hidden size 64 and 20% dropout between layers. We trained the network with lookahead depths (number of steps to forecast in the future) 2, 4, 8, and 16 using the Adam optimizer with learning rate 0.001, training for 20 epochs (sufficient for convergence). We performed the training on a cloud computing node with 1 NVIDIA T4 GPU, 4 Intel Xeon Platinum CPUs, and 64 Gb memory. The training of a single model took 20 min.

We compared our approach with conventional imputation methods 'replace by the mean' and 'replace by a random sample'. In the 'replace by the mean' method, a missing value is replaced by the mean of the forecast. In the 'replace by a random sample' method, a missing value is replaced by a random sample drawn from the forecast. As a performance metrics, we used per-point negative log-likelihood loss on the test set. Tables 1 and 2 show the difference in loss between uncertainty propagation and 'replace by the mean' and 'replace by a

Table 1. Uncertainty propagation vs. 'replace by the mean'.

Missing	2	4	8	16
5%	0.001	0.001	0.06	0.10
10%	0.001	0.002	0.08	0.11
20%	0.003	0.003	0.11	0.13
50%	0.004	0.006	0.12	0.16

Table 2. Uncertainty propagation vs. 'replace by a random sample'.

Missing	2	4	8	16
5%	0.04	0.04	0.13	0.14
10%	0.06	0.06	0.16	0.24
20%	0.11	0.12	0.20	0.27
50%	0.18	0.19	0.28	0.30

random sample', correspondingly. The greater is the number, the worse is the forecasting by each of the methods compared to uncertainty propagation. One can see that in all cases uncertainty propagation provides better forecasts than either of the conventional methods.

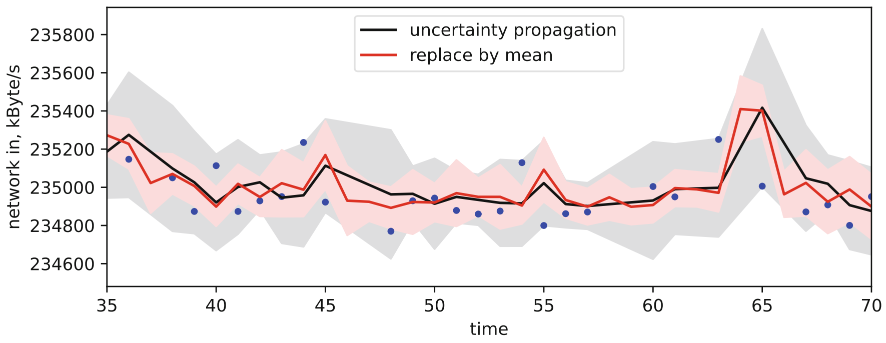


Fig. 2. Uncertainty propagation vs 'replace by the mean'. 95% confidence intervals are shaded.

As an illustration of the advantage of uncertainty propagation, consider Fig. 2, which shows forecasts using uncertainty propagation and 'replace by the mean' in presence of missing values. Forecasts through uncertainty propagation result in adequate confidence intervals. However, when missing values are replaced by the mean of the belief distribution, further forecasts are overconfident and too many observations fall outside of 95% confidence intervals.

The code and data for the case studies are available at https://bitbucket.org/dtolpin/dbts-studies/.

5 Related Work

There appear to be two interconnected areas related to this research. One area is uncertainty representation and propagation in recurrent neural models. The other area is handling of missing values in time series, again in the context of recurrent neural models in particular.

The importance of uncertainty quantification in deep learning is well understood [1]. Recurrent neural networks can express forecast uncertainty through predicting distribution parameters, such as the mean and the standard deviation, instead of point values [12]. When expressing uncertainty by closed-form distributions is insufficient, stochastic latent variables are introduced into RNNs [10,11,20]. Uncertainty representation in RNNs is related to uncertainty propagation and multi-step forecasting. For multi-step forecasting, uncertainty must be propagated multiple steps into the future. Uncertainty propagation is usually achieved through random sampling during training or inference [3,14,20]. Our approach differs in that conventional RNN architectures are leveraged to represent uncertainty in both the input and the output, and that uncertainty propagation is accomplished deterministically, without resorting to random sampling, which facilitates efficient training and inference.

Handling of missing values in time series has inspired research for decades due to the fact that many otherwise efficient and robust algorithms, in particular those based on recurrent neural architectures, require that all values in the time series are present and lie within a valid range [19]. A widespread approach is to *impute* the data, that is, to replace missing values with values inferred from other values in the same time series or in other time series in the data set [13, 17]. Alternatively, a missing value is treated as an observation itself, often by introducing an auxiliary indicator variable [4,15]. In our work, we take a third approach—a missing value, either due to an absent observation or in the course of multi-step forecasting, is replaced by a parametrically specified belief distribution of the value based on the past observations.

6 Discussion and Future Research

We presented a deep probabilistic architecture for uncertainty propagation in multivariate time series. This architecture organically handles two important problems in deep time series modelling: missing data and multi-step forecasting. Empirical evaluation demonstrated that our approach outperforms conventional baselines in terms of forecasting accuracy, while still being easy to implement. Since, unlike some other approaches to uncertainty propagation, our architecture avoids sampling, uncertainty can be propagated efficiently and represented in closed parametric form, rather than approximated by samples and posterior intervals.

We confined most of the discussion to the normal uncertainty shape. Other distributions can be used instead of the normal distributions where appropriate, provided their parameterization allows to express a certain observation as well as an uncertain belief. Analysis of distributions for representing uncertainty and their feasible parameterization is a subject of ongoing research. Another research direction worth exploring is extension of the presented architecture to bidirectional recurrent neural networks [5]. Bidirectional RNNs allow to account for both past and future observations where appropriate, but apparently make uncertainty propagation more complicated. Still, preliminary results suggest that uncertainty in bidirectional RNNs can be handled in a similar manner, further facilitating efficient probabilistic uncertainty propagation in a broader class of deep learning models for time series.

Acknowledgements. We thank PUB+ for providing computational facilities for conducting the empirical evaluation. David Tolpin is partially supported by Israel-U.S. Industrial Research and Development Foundation's *Cybersecurity technology for critical power infrastructure AI-based centralized defense and edge resilience* project.

References

1. Abdar, M., et al.: A review of uncertainty quantification in deep learning: techniques, applications and challenges. Inf. Fusion **76**, 243–297 (2021)
2. Afgani, M., Sinanovic, S., Haas, H.: Anomaly detection using the Kullback-Leibler divergence metric. In: 2008 First International Symposium on Applied Sciences on Biomedical and Communication Technologies, pp. 1–5 (2008)
3. Alaa, A., Van Der Schaar, M.: Frequentist uncertainty in recurrent neural networks via blockwise influence functions. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, 13–18 July 2020, vol. 119, pp. 175–190. PMLR (2020)
4. Bansal, P., Deshpande, P., Sarawagi, S.: Missing value imputation on multidimensional time series. Proc. VLDB Endow. **14**(11), 2533–2545 (2021)
5. Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., Karhunen, J.T.: Bidirectional recurrent neural networks as generative models. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28. Curran Associates, Inc. (2015)
6. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. **41**(3), 1–58 (2009)
7. Che, Z., Purushotham, S., Cho, K., Sontag, D., Liu, Y.: Recurrent neural networks for multivariate time series with missing values. Sci. Rep. **8**(1), 1–12 (2018)
8. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: encoder-decoder approaches. In: Wu, D., Carpuat, M., Carreras, X., Vecchi, E. (eds.) Proceedings of SSST 2014–8th Workshop on Syntax, Semantics and Structure in Statistical Translation, pp. 103–111. Proceedings of SSST 2014–8th Workshop on Syntax, Semantics and Structure in Statistical Translation, Association for Computational Linguistics (ACL) (2014). Funding Information: The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. Publisher Copyright: 2014 Association for Computational Linguistics; 8th Workshop on Syntax, Semantics and Structure in Statistical Translation, SSST 2014; Conference date: 25-10-2014

9. Choi, K., Yi, J., Park, C., Yoon, S.: Deep learning for anomaly detection in time-series data: review, analysis, and guidelines. IEEE Access **9**, 120043–120065 (2021). https://doi.org/10.1109/ACCESS.2021.3107975
10. Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., Bengio, Y.: A recurrent latent variable model for sequential data. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS 2015, pp. 2980–2988. MIT Press, Cambridge (2015)
11. Fraccaro, M., Sønderby, S.R.K., Paquet, U., Winther, O.: Sequential neural models with stochastic layers. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc. (2016)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
13. Kim, Y.J., Chi, M.: Temporal belief memory: Imputing missing data during RNN training. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, pp. 2326–2332. International Joint Conferences on Artificial Intelligence Organization (2018)
14. Li, L., Yan, J., Yang, X., Jin, Y.: Learning interpretable deep state space model for probabilistic time series forecasting. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019, pp. 2901–2908. AAAI Press (2019)
15. Lipton, Z.C., Kale, D., Wetzel, R.: Directly modeling missing data in sequences with RNNs: improved classification of clinical time series. In: Doshi-Velez, F., Fackler, J., Kale, D., Wallace, B., Wiens, J. (eds.) Proceedings of the 1st Machine Learning for Healthcare Conference. Proceedings of Machine Learning Research, 18–19 August 2016, vol. 56, pp. 253–270. PMLR, Northeastern University, Boston (2016)
16. Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., Pei, D.: Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, pp. 2828–2837. Association for Computing Machinery, New York (2019)
17. Suo, Q., Yao, L., Xun, G., Sun, J., Zhang, A.: Recurrent imputation for multivariate time series with missing values. In: 2019 IEEE International Conference on Healthcare Informatics (ICHI), pp. 1–3 (2019). https://doi.org/10.1109/ICHI.2019.8904638
18. Tolpin, D.: Population anomaly detection through deep gaussianization. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, pp. 1330–1336. Association for Computing Machinery, New York (2019)
19. Wen, Q., et al.: Time series data augmentation for deep learning: a survey. In: Zhou, Z.H. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 4653–4660. International Joint Conferences on Artificial Intelligence Organization (2021). Survey Track
20. Yin, Z., Barucca, P.: Stochastic recurrent neural network for multistep time series forecasting. In: Mantoro, T., Lee, M., Ayu, M.A., Wong, K.W., Hidayanto, A.N. (eds.) ICONIP 2021. LNCS, vol. 13108, pp. 14–26. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92185-9_2

Time, Memory and Accuracy Tradeoffs in Side-Channel Trace Profiling

Hen Hayoon(✉) and Yossi Oren(✉)

Department of Software and Information Systems Engineering, Ben-Gurion University, Beersheba, Israel
hayoonh@post.bgu.ac.il and, yos@bgu.ac.iland

Abstract. Template attacks are one of the most powerful classes of side-channel attacks. Template attacks begin with an offline step, in which the side-channel traces are profiled, and decoders are created for each side-channel leak. In this paper, we analyze the compression step of the trace profiling process. This compression step, which is a central part of the decoder's training process, is used to reduce the amount of time, memory consumption, and data required to successfully perform the attack; various practical methods have been proposed for this step, including one which uses an efficient means both for selecting the points of interest (POI) in the power trace and for preprocessing noisy data.

We investigate ways to improve the efficiency of the attack by implementing several compression methods which select the most informative power consumption samples from power traces. We develop a unique dedicated evaluation system to compare the performance of various decoders with different compression methods on real-world power traces. Our findings indicate that our proposed decoder for side-channel traces outperforms the current state of art in terms of speed, resource consumption, and accuracy. We also demonstrate our decoder's effectiveness under resource-constrained conditions, and show that it achieves over 70% accuracy even if there are fewer than 1,000 traces in the profiling phase.

1 Introduction

Side-channel attacks (SCAs) [4,9,15] have been shown to be effective and practical for attacking implementations of cryptographic algorithms. These attacks reveal cryptographic device secrets by observing the physical properties of the device [15]. Adversaries can obtain sensitive information from side-channels, such as the timing of operations, power consumption, electromagnetic emanations, etc. [4,15,17]. When a cryptographic operation is performed, the device emits a data dependent side-channel leak. Leaks are the internal state functions of the device under test (DUT), and they are modulated into a power/EM trace, along with some noise. In constrained devices, such as chip-cards, straightforward implementations of cryptographic algorithms can be broken easily, since the power consumption of the cryptographic device is dependent on the intermediate values of the cryptographic algorithm executed. When the amount of

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 29–46, 2022.
https://doi.org/10.1007/978-3-031-07689-3_3

leakage is smaller relative to the noise, statistical techniques such as DPA are applicable. As stated by Chari et al. [6], DPA relies on the statistical analysis of a large number of samples where the same keying material is used to operate on different data. The most powerful type of side-channel attack is called the template attack (TA). The TA's evaluation of side-channel information relies on a multivariate model of the side-channel traces. This attack contains an offline and online phase, and one of its advantages is that it requires just a few samples in the online phase, and works well even if the DUT's power consumption does not conform to the Hamming weight leakage model.

In order to recover the secret key in a template-based side-channel attack, the attacker's operations consist of three phases. First, in the offline *profiling phase*, a device totally controlled by the attacker and similar to the DUT is profiled and characterized. This DUT analysis, like correlation power analysis (CPA) [4], identifies the position of the leaking operations in the traces by identifying a small section of the power trace T depending only on a few unknown key bits. The profiling phase outputs a series of *decoders*, each mapping a certain set of points of interest (POIs, also known as features) in the trace to a certain set of secret values. The Hamming weight model is an example of the mapped output of this phase [17,25]. The second phase consists of an online *decoding phase*, where the attacker is provided with a few power traces, generally a single one, and uses the decoders created in the profiling phase to recover leak vectors from the power trace. These leak vectors may contain some errors due to noise. The last phase is the *solving phase*, where the correct key is discovered from among the most likely candidates by using the brutre force, the maximum-likelihood method [12] or by the use of a constraint solver [14,25].

1.1 Contribution

Trace compression is the initial step of the profiling phase. In this step the power trace is replaced with a smaller-sized vector, in order to improve the decoder's performance, in terms of the number power traces required, and its overall performance in both the online and offline phases. In this paper, we investigate which parameters for this compression step deliver the best combination of accuracy and runtime performance. Specifically, our paper makes the following contributions: we design and implement a unique evaluation performance system which can analyze the compression step. We then use this system to explore and compare several profiling methods. The profiling methods differ based on the compression techniques and preprocessing model (for the training set traces) utilized. Three well-known and different classical compression methods were implemented. In addition, we implement the method used by the smart decoder proposed in [18] and presented in Sect. 3.4, it offers an efficient searching algorithm to find the most leaking points in the trace using a unique compression method. Finally, we propose our own optimal profiling method, based on the guidelines of [24], as a tradeoff that performs well under conditions of data and resource restrictions.

1.2 Related Work

Weisse et al. [18] presented a method for profiling the training set of power traces into an accurate decoder to be used as part of an algebraic side-channel attack aimed at recovering the secret key. The authors provided two scoring methods used to identify the best leaking points in the trace; their methods are effective when there is limited representation of some values in the training set traces. Kocher et al. [13] were the first to present the difference of means (DOM) as an alternative to correlation for power computation measurements analysis in order to determine the secret keys of a DES operation. Before that, Chari et al. [6] showed the best visualization for the DOM of side-channel samples for a TA with a single sample against an RC4 implementation.

Mangard et al. [15] used the DOM as an alternative to correlation not only for the binary power model but for the Hamming-weight model, directly on power traces to perform a DPA attack on the S-box of an AES implementation [8]. In another chapter of their book the authors suggested the model-based integration of SNR (signal-to-noise ratio) techniques to compress power traces, using the sum of their signal and noise in a defined time interval. Gierlichs et al. [10] suggested the sum of squared pairwise differences (SOSD) instead of the regular sum of pairwise of the DOM for the selection of interesting points in a TA for an SCA against the AES. Rechberger et al. [24] presented a practical TA using an advanced version of the maximum extraction compression method to select the interesting points in a power trace. Instead of choosing the highest points, the authors defined a few properties that must exist at the selected points.

Other authors revealed the importance of feature selection in an SCA and compared them in other scenarios. Zheng et al. [31] compared known feature selection techniques and evaluated their accuracy for profiled SCAs. Picek et al. [23] investigated advanced feature selection techniques from the machine learning domain used to improve attack accuracy, examining the influence of the number of features in the process. Cagli et al. [5] presented an accuracy comparison analysis for feature selection in SCAs through linear, non linear, and neural-network models.

2 Background

In this work we assume a DUT conforming to the Hamming weight power leakage model.

Let k be an encryption key (bytes), and p and c are respectively the plaintext and ciphertext of the cryptographic algorithm, which is uniformly chosen. The multivariate power trace measured is denoted as $\overrightarrow{X} = X_1, ..., X_S$, where S is the number of time samples (also called features).

In the offline phase, the attacker, who controls the DUT, estimates the leakage model using a set of N profiling traces $\overrightarrow{X}_1, ...\overrightarrow{X}_N$ (multi dimensional $S \times N$) and the knowledge of k. In the online phase, additional power traces of the DUT $\overrightarrow{X}_1, ...\overrightarrow{X}_M$ itself are measured (one or more), and the attacker's objective is to recover k from these power traces using signal classification techniques.

2.1 Template Attacks

Template attacks are a method for performing power analysis attacks which works by creating a characterization of a device. The attack usually consists of an offline phase, in which device characterization takes place, followed by a online phase, in which the characterization is used for the attack.

First, a series of templates of all possible operations (i.e., all of the cryptographic algorithms are executed using all of the possible subkey values) is constructed. Then, the attack starts, and the trace of a single operation is captured. Using the templates created, which represent all key values, the side-channel information of the attacked device is classified and assigned to one or more of the templates. The goal is to significantly reduce the number of possible keys, optimally concluding the attack with a single possible value for the secret key.

Building Templates. In the state-of-the-art template attack the power samples are considered dependent by the TA; accordingly, the traces are characterized with the multivariate normal distribution [6,7,15,24,26–28]. The characterization defined by the "template" of the multivariate normal distribution, is the pair (m, CM) where m represents the mean vector and CM represents the covariance matrix for each class. When building templates with the power model, we determine templates for certain sequences of instructions by executing them with different and known data d_i and keys k_j, in order to record the resulting power consumption. Then, we group the corresponding traces to the pair of (d_i, k_j) and estimate (m, CM).

As a result, we obtain a template for every data and key pair. Then, using the template, along with $(m, CM)_{d_i,k_j}$ and the power trace x, we evaluate the probability density function of the multivariate normal distribution :

$$p(x; (m, CM)_{d_i,k_j}) = \frac{exp(-\frac{1}{2} \cdot (x-m)' \cdot CM^{-1} \cdot (x-m))}{\sqrt{(2\pi)^{\tau} \cdot det(CM)}}$$

The probabilities for each template measure how well they fit to a given trace. If the noise level is sufficiently low, the maximum-likelihood decision rule can be applied, and the template with the highest probability indicates the correct key.

2.2 Dataset

The dataset used in this study is the DPA contest v4 dataset [20], which provides measurements of a masked AES implementation. In that attack contest, the goal was to use the smallest number of power consumption traces to identify the first 128 bits of the encryption key. **The hardware** used for the cipher implementation was the Atmel ATMega-163 smart card, which was sampled using a LeCroy Waverunner 6100A oscilloscope at the rate of 500 MS/s. The dataset provides 100,000 power traces, each of which consists of 435,002 samples and corresponds to the execution of an AES-256 round. **The countermeasure** of the AES-256 implementation was "Rotating S-Box Masking" (RSM) [21]; all of the power traces provided were running the "AES-RSM" implementation with the same 256-bit key, but different plaintext was used for each measurement. This implementation contains the following features:

- An arbitrary fixed 16-byte mask is added on top of the classic AES. A random offset $0 \leq o \leq 15$ is drawn as the first stage of the encryption process. Let m^0 denote the cyclic rotation of the mask added by offset o.
- The masked plaintext pm is the result of the XOR operation of the 16 bytes of plaintext with m^0. Let pm_i denote the result $pm_i = p_i \oplus m_i^0, 0 \leq i \leq 15$.
- The AddRoundKey phase uses pm for each sub-round key.
- The masked S-boxes, which are derived from the value of m^0 of, are used.
- The ShiftRows and MixColumns sub-round phases are unchanged.

We generated and parsed the plaintext, offsets and key files into comma-separated value (CSV) files, according to the size expected by the parsing code used by [18], and matched them (populating from index 0) with the RSM trace indices, according to the trace index file found on the DPA contest site [21].

2.3 The Hamming Weight Leakage Model

General Assumption. The assumed form of the Hamming weight (HW) leakage of information in power consumption described in [15] is: $P_{total} = P_{exp}(HW(s_i)) + P_n$, where $HW(s_i)$ denotes the HW of the intermediate state byte s_i for a certain leak i, and P_n denotes the noise component which is assumed to be normally distributed with unknown parameters. This form is exactly the probabilistic model used to construct a Bayesian classifier. As was done in [22,24,30], we therefore use a *naive Bayes* (NB) classifier. The NB classifier returns, for each feature, a mean and variance for each class of the 9 possible HW classes (0–8).

AES-RSM Leaks. The leakage model for the AES-RSM implementation is the HW model. The desired leakage of information of the AES-RSM implementation is the Hamming weight of the S-box state bytes they process. The following leaks can be derived from the traces of DPA v4 used in our study: 16 bytes of the masked plaintext pm_i, as describe in Sect. 2.2, 16 bytes of the output of the AddRoundKey computation, 16 bytes of the output of SubBytes, and finally 52 bytes from the MixColumns computation. The first 16 bytes were added by the RSM countermeasure, The rest 84 bytes are the same as enumerated in [18]. In aggregate, there are 100 leaks from 100 intermediate byte values.

3 Compression Methods

The compression step is the first step performed during the profiling phase of an SCA. Compression methods are usually used to reduce the complexity of power analysis attacks, by reducing the length (dimension) of the power traces. This is done in cases in which there are not enough traces for a full rank covariance matrix CM, to cope with computational or memory restrictions, as the size of the CM grows quadratically with the number of samples in the trace.

The motivation for using compression methods stems from the amount of redundancy present in long power traces, as these methods are able to remove this redundancy without significant loss of leak information. To ensure an efficient compression process, it is necessary to know which points in the power traces points the “points of interest” (POIs) and contain information relevant to an attacker. These samples have the highest information leakage, which is reflected in their high correlation to the number of transitions that occur in the chip, where we implicitly assume that the number of transitions depends on the operation performed and on the data being processed. Identification of the POI by the attacker is the first step in device characterization, and this information is used to build the templates.

There are two main compression method approaches: The first is the “selection of samples” approach, which is based on some criteria, and the second is the “usage of linear combinations” of the leakage vectors approach, which based on the principal components or Fisher’s linear discriminant.

In this section, we describe methods from both approaches in order to cover a wide range of methods in our performance comparison. We first present a few classical methods; then we discuss the compression method used by the smart decoder proposed in [18], and finally we describe the compression method used in our decoder.

3.1 Principal Component Analysis

Principal component analysis (PCA) is mainly used in multivariate statistics to reduce the dimensionality of a dataset while retaining the most variance [3], by finding patterns within the dataset. PCA searches for linear combinations with the greatest variance, and divides them into principal components (PCs) where the greatest variance is captured by the highest component. The first PC is required to have the greatest variance. The second PC must be orthogonal to the first component while capturing the greatest variance within the dataset in that direction; subsequent components cover less and less of the remaining data variance.

The maximum number of PCs (dimensionality) is equal to the number of samples in the power trace. Choosing the right number of PCs (designated by n) is essential for obtaining optimal results, as shown in [11], by maximizing the variance in the original data and minimizing the reconstruction error of the data transformation.

The PCA method is based on the usage of the linear combinations approach (mentioned above), and we chose to implement it, since according to [7,19], its success, unlike the Linear discriminant analysis (LDA) method, depends on the condition of equal covariance (known as homoscedasticity). We use the MATLAB implementation of PCA [1] which returns a vector containing the percentage of the total variance explained by each PC. Then, the mean of each relevant feature from all of the training traces is calculated.

The training part calculates the principal features by multiplying the coefficient by the difference of each feature with the mean. The vector returned is the implementation of the "cumulative percentage of total variation" method for choosing n, which is the method recommended by [11,19]; we experiment with n values in the range $[5, 10, 30]$, since the performance measurements with n $<$ 5 and n $>$ 30 are extreme and unstable.

3.2 Difference of Means

The difference of means (DOM) compression method is based on the selection of samples approach, as mentioned above. The point selection is based on a pre-calculated signal strength estimate. The DOM method only considers the differences of values, and not the corresponding variances of the power traces, as essential for comparison – an important factor in our decision to implement it.

The DOM method is used to determine the relationship between the recorded power consumption matrix T (traces) and the columns of binary assumption-based matrix H. The H matrix is created by the attacker under the assumption that the power consumption for certain intermediate values is different for all other values; the binary value of H is a function of the input data d and a key hypothesis k_i, $h_{i,j} = HW(v_{i,j}), v_{i,j} = f(d_i, k_j)$. As suggested in [15], to reduce the HW model to a binary model, we set $h_{i,j} = 1$, if $HW(v_{i,j} \geq 4)$ and $h_{i,j} = 0$, if $HW(v_{i,j} < 4)$.

According to h_i, the attacker splits T into two sets of power traces (rows) for k_i's correctness check. The first set contains those T row indices corresponding to the indices of the zeros in the vector h_i, while vector m'_{0_i} denotes the mean of those rows $m_{0_{i,j}} = \frac{\sum_{l=1}^{n}(1-h_{l,j})\cdot t_{l,j}}{n_{0_i}}$. The second set contains all remaining rows in T, while vector m'_{1_i} is their mean vector $m_{0_{i,j}} = \frac{\sum_{l=1}^{n} h_{l,j}\cdot t_{l,j}}{n_{1_i}}$, where n denotes the number of rows in H and $n_{0_{i,j}} = \sum_{l=1}^{n}(1 - h_{l,i})$, $n_{1_{i,j}} = \sum_{l=1}^{n} h_{l,i}$.

A significant difference between the mean vectors m'_{0_i} and m'_{1_i} at some point in time indicates the correctness of key hypothesis k_i.

Each row in the results of matrix R: $R = M_1 - M_0$ corresponds to the differences between the mean vectors m'_{0_i} and m'_{1_i} of one key hypothesis.

3.3 Integration SNR

The integration methodology is a robust compression technique based on the selection of samples approach, which uses all of the recorded points in the power traces, and not just the peak/diff values. Unlike the DOM method, the integration SNR method takes the variance of the traces into consideration, an important factor in our decision to implement it.

The signal-to-noise ratio of a power sample is given by the following equation:

$$SNR = \frac{Var(P_{exp})}{Var(P_{sn} + P_{en})}$$

where P_{exp} is the exploitable power consumption, Psn is the switching noise, and Pen is the electronic noise. The SNR quantifies how much information is leaking from a power trace. The higher the SNR, the greater the leakage. The integration of power trace in a time interval affects the SNR, since the signal and the noise of the recorded points in the time interval are summed. The SNR can be increased or decreased by the integration, depending on the time interval size used for the integration.

The time interval size is a decisive parameter for the compression's success; in cases in which there are many points with a strong signal in the time interval, the SNR will be high, and in cases where a single point is combined with points that leak little to no information (or no information at all), the SNR will be lower than the single point. Therefore choosing an appropriate time interval for the integration, like the length of a clock cycle (*cc*), is essential for a good compression process [15,24].

We implemented the following methods for integration-based power trace compression:

Integration Row (IR) computes the sum of all points of each time interval.

Sum of Absolute Values (SOA) computes the sum of the absolute values of all points of each time interval.

Finally, **Sum of Squares** (SOS) computes the sum of the squares of all points of each time interval.

We tested all methods with time interval ranges of [0.5, 1, 2] *cc*.

3.4 Top Score

The state-of-the-art profiling methods for this paper are based on the profiling methodology of [18]. In this work, Weisse et al. introduced a smart method for profiling the training set of power traces into an accurate decoder for an algebraic side-channel attack. The authors developed an efficient searching algorithm to identify the points in a trace that leak the most.

For the feature selection process, they proposed a scoring method for evaluating the amount of information each sample contains about a specific leak. Their profiling phase consists of the following steps:

1. Find regions of interest (ROI) in the traces for every leak using the Pearson correlation coefficient [4].
2. Calculate the feature scores for the features within the ROI of the evaluated leak identified in the previous step. The feature score is set as the average of 200 a-posteriori probabilities (of 200 evaluation traces) assigned to the correct Hamming weight by the Bayesian classier trained on the feature.
3. Create the best feature set (the set which contains the most information regarding the specific leak), which is used as input for the classier. Using the same Bayesian classifier and evaluation traces as the previous step, the mutual scores of all features in the best feature set are calculated; eventually, only the features that increase the score are included in the set. The best feature set size was statically limited to 500 features.

In our research, we use the success-rate scoring method of Weiss et al.'s by stabilizing the code from [29] and adjusting it as a compression method in the following way:

- Download the code base of [29]
- Parsing the RSM traces as describe in Sect. 2.2
- Mapping and isolating only the relevant code parts for the profiling and decoding analysis phases
- Adjusting the code base to matlab v2017 and rewriting respectively.

We denote this method as "Top Score" (TS). In our experiments, we limit the best feature set size to half the size of the training set.

3.5 Optimal Selection

Rechberger et al. [24] presented a practical TA using an efficient means of selecting the POI in a power trace and preprocessing noisy data. Their method is based on the maximum-extraction compression approach, in which the maximum peak values of the recorded samples in a clock cycle are simply extracted [15]. Instead of choosing the highest points, their advanced selection method defines a few properties that must exist at the selected points in the compressed trace:

- The minimum distance between the points should be approximately one clock cycle or more, since additional points in the same clock cycle do not provide additional information.
- The minimal height of a selected point should be higher than the noise floor of the sum of differences (SOD) trace.

In our research we implement the guidelines for the selection of POI from Rechberger's paper [24], as a substitute for operations of correlation calculation and scoring the features in the profiling phase. We adjust it as a compression method in our optimal suggested decoder, We call it "Optimal Selection" (OS). The term "optimal" was chosen due to its good trade-off between the performance parameters.

Three different minimum distances were chosen for testing; [0.5, 1, 2] *cc*. Not in accordance with Rechberger's recommendation, we also performed with 0.5 cc to test the selection of features in a situation where at the same clock cycle there are 2 samples at the same height (2 peaks).

The calculation of the noise floor is performed by multiplying the maximum value in the SOD traces with the noise factor which we set as 0.6 after testing the range of [0,1]. Unlike Rechberger, who considered constant numbers of POI ranging from 1 to 40 and set the level of the noise factor accordingly, we considered all the points which are higher than the noise floor calculated with the constant noise factor equal to 0.6, even if a higher number of points of interest is chosen.

4 Evaluation of Methods

We analyzed the performance of many TA profiling phase variants on real-world data, comparing all of the compression methods described in Sect. 3 using various configurations.

4.1 Evaluation System

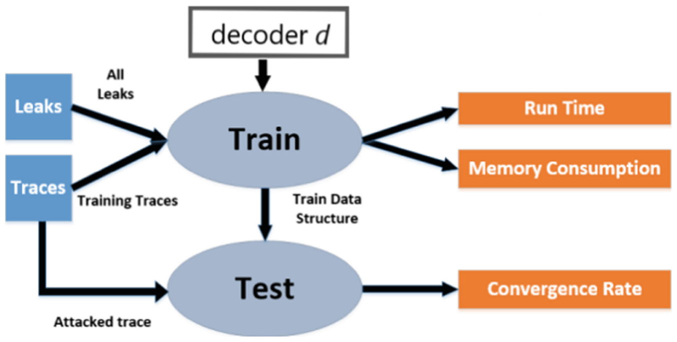


Fig. 1. Evaluation system architecture

The design of our evaluation system is a black-box for benchmark of a given decoder. The input of the system is decoder d, and the output is decoder d's training and test results for three parameters: runtime, memory consumption, and convergence rate. The system contains a data-set of traces and leaks for the training and test operation measurements. Our system architecture can be seen in 1. In our performance analysis we measured the following:

Memory Consumption: We measured each decoder's RAM usage (the total memory usage in bytes) as it processed the training set during the profiling phase. This was measured using the OS standard memory reporting function *vmstat* (Linux), when only the decoding process was running in our development environment.

Run-Time: We measured the operational runtime and overall convergence time of decoder as it processed the training set in the profiling phase. For the run-time measurements, we use the MATLAB timing functions *timeit,* to time how long the decoder code takes to run, and *tic-toc,* to measure the convergence time and operations' performance timing (the total execution time in seconds).

Convergence Rate: We estimated each decoder's quality based on its online convergence rate. This was measured by examining the number of traces that must be provided in the offline phase in order for each decoder to obtain reasonable results in the online phase. The convergence vector for each decoder is a Boolean vector representing the success/failure of the decoder's test results, with the overall convergence rate calculated as the mean of this vector.

4.2 Definition of the Training and Test Methods

The **training method** is used for model construction, classification and retention within its data structure; its input is pairs of (Trace T, Leak L). Training consists of measuring run time and memory consumption and takes place in three steps: First, in the feature extraction step, a compression method is used. Next, in the feature pre-processing step, the selected features are grouped and labeled. Finally, in the template learning step, a naive Bayes classifier is trained for each leak (to distinguish between HW classes) using the features selected in the feature extraction step.

Preprocessing operations, such as leak calculation in Sect. 2.3 or trace parsing in Sect. 2.2, were not part of our measurements, since running them adds constant time which is not correlated with the decoding method.

The **test method** evaluates decoder d by receiving trace T (the attacked trace) as input, and examining its output, which consists of the HW of a certain leak based on its training process. This is validated on its data structure during testing. The validation results of *true* (success) or *false* (failure) are presented in the Boolean convergence vector of decoder d.

4.3 Experimental Setup

In our experiments, we explore profiled template attacks' feature selection (compression) methods for extracting the best subset of power samples for the classification of features according to the HW classes. All experiments are performed with MATLAB v2017, installed on an Intel Xeon E5-2620 CPU with 128 GB of RAM, running Ubuntu 18.04. The compression methods previously described are implemented to reduce the dimensions of the power traces. The subset sizes are selected based on the guidelines described in related studies. For each technique, we analyzed several configurations, and only the one yielding the best result is used in our overall comparison of the methods. Once the compressed trace is set, we implement the training and test methods (describe in Sect. 4.2) for each compression technique to evaluate its performance. We use 2,000 power traces from the initial dataset.

The steps of training and measuring performance are implemented in the following way:

We used Bayesian classifier for training template classifiers with HW leakage model for every leak. Using the *fitcnb* matlab function [16] which fits a naive Bayes classifier to data, we train a naive Bayes model on the selected features with the HW classes using normal distributions. We train a Bayesian classifier whose input features are those selected in the compression method. The training set consists of approximately half the number of traces examined in the experiment, and the other half serves as the evaluation set, in order to avoid an over-optimistic performance evaluation. The classifiers are trained to distinguish between Hamming weights 0–8. To measure the classification accuracy (%) of the trained classifiers during training, we used MATLAB's *predict* function [2]on the set of the evaluation traces (different from those used for training and testing), where the accuracy is the number of correctly classified leaks divided by the total number of leaks over the traces. To test the decoder, as described in Sect. 4.2 we used the classifier's data structure built during training and ran the *predict* function on the attacked trace taken from the other half of the examined traces, which is denoted as the *test set*. We report the results of the performance measured during training and testing for each decoder.

5 Results

Tables 1 and 2 present the memory consumption (KB) and runtime (seconds) results during training. Table 3 presents the convergence rate (%) results for the testing phase. In each table, the row with the best configuration for each method appears in black (these results are included in our overall comparison). Figure 2 present the overall performance results for the compression methods examined. Table 1 presents the memory consumption results for each decoder during the training phase, with different amounts of training traces. Optimal Selection (OS) is the best performing compression method when considering memory usage; except for its low usage, identical values were obtained for training using different distances (*cc* values). Difference of Means (DOM) has the highest rates of memory consumption. Principal Component Analysis (PCA) follows, with 10 *pc* to keep found as the best performing setting for PCA. All SNR-based methods show the same trend in which the usage with 0.5 *cc* was higher than it was with 1 *cc*, and the usage was slightly lower with 2 *cc* than it was with 1 *cc*. The results obtained for all of the methods show that increasing the number of training traces results in higher memory consumption.

Table 2 presents the training phase's runtime results for each decoder, using different amounts of training traces. As seen in the table, TS has the longest run-times, and this is followed by DOM. Next comes OS and DOM, which shows identical values for different distances (*cc*). PCA follows with similar results found for 5 *pc* and 10 *pc*, while 30 *pc* shows longer results. The SNR methods had the shortest run-times, with Sum of Absolute Values (SOA) being the faster among these methods. In all methods we observed that increasing the distance by one *cc* cuts the runtime in half. The results for all methods show that as the number of training traces increases, the runtime also increases.

Table 3 presents the accuracy results for each decoder during testing phase, based on the number of traces in the training set. The "Top Score" (TS) method of [29] was found to be as the most accurate, and it is followed with OS, which has an accuracy greater than 70%; similar results were obtained for 1 *cc* and 2 *cc*. DOM also obtained good results, while PCA, which obtained the same results for all pc configurations, came next. The SNR methods obtained low results; while SOA performed the worst, the same trend was observed for all three SNR methods - increasing the distance by 1 *cc* significantly reduces the accuracy, sometimes by half. The results for all methods show that as the amount of training traces increases, the accuracy also increases.

Table 1. Memory consumption for all compression methods

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
DOM		2413736	3998680	4124432	4753541	4902672	5694248	6044480	6100464
PCA	5 *pc*	783445	850248	2088544	4268990	4562111	4672332	5091123	5100939
PCA	10 *pc*	772504	851496	2085968	4272824	4523424	4609552	4961832	5139784
PCA	30 *pc*	1116992	1206768	2406654	4466343	4809202	4999765	5163442	5432887
TS		338160	954512	1182768	2180192	2749328	3894856	4733248	4812720
IR	0.5 *cc*	281396	4011232	4088761	4298778	4311023	4695341	4810116	4955809
IR	1 *cc*	182304	2931551	3116088	3353981	3385451	3722401	3774523	3911296
IR	2 *cc*	175772	2864598	3108973	3294332	3334423	3566878	3672445	3668915
SOA	0.5 *cc*	190203	1973771	2154481	3700231	4378922	4399872	4752112	4998213
SOA	1 *cc*	111408	1353984	1404264	2856560	3421216	3554512	3856481	4131128
SOA	2 *cc*	100897	1184562	1302213	2671333	3302451	3267763	3676884	4102893
SOS	0.5 *cc*	278965	338767	387291	3909845	4144521	4453201	5022318	5296659
SOS	1 *cc*	199432	442704	290176	3014256	3271528	3661376	4149342	4350296
SOS	2 *cc*	190567	240561	264367	2889416	3240567	3653299	4103292	4203389
OS	0.5 *cc*	96236	132271	280038	279913	450221	718134	841761	1125032
OS	1 *cc*	96160	130488	281840	282224	449848	715920	839288	1137648
OS	2 *cc*	95988	129850	282098	290013	451211	719656	836114	1135451

Table 2. Run-Time for all compression methods using various

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
TS		617.2	964	995.2	1316.1	2037.3	2297.3	2418.9	2428
DOM		506.9	570.3	598.4	610.2	703.6	720.3	747.2	818.3
OS	0.5 *cc*	153.1	264	302.9	323.7	384.2	442.5	497.1	525.9
OS	1 *cc*	152.4	262.3	301.5	323.8	383.2	443.6	495.4	525.8
OS	2 *cc*	152.2	263.7	301.8	324.1	383.1	442.6	496.1	523.9
PCA	5 *pc*	91.2	173.1	211	235.4	289.5	325.9	353.2	387.8
PCA	10 *pc*	91.3	173.2	211.2	235.7	289.9	326.7	353.9	388.6
PCA	30 *pc*	102.5	181.9	220.7	248.1	301.4	339.8	377.8	405.2
IR	0.5 *cc*	78.2	91.3	168.9	230.8	257.8	293.6	310.3	341.8
IR	1 *cc*	37.1	46.8	85.6	115.5	130.4	145	156.6	171.9
IR	2 *cc*	16.2	23.6	43.4	59.1	66.6	74.1	79.8	85.2
SOS	0.5 cc	128.2	132.3	170.6	176.2	343.5	380.7	403.2	438.9
SOS	1 *cc*	65.9	68.7	87.2	139.4	174.2	192.2	206.8	222.8
SOS	2 *cc*	32.7	36.4	44.8	68.1	89.5	97.1	102.5	110.1
SOA	0.5 *cc*	92.5	132.7	177.5	236.8	264.2	280.6	361.7	431.2
SOA	1 *cc*	47.2	67.6	90.3	119	133.1	144.3	182.5	218.1
SOA	2 *cc*	25.8	34.8	47.1	62.6	66.6	73.5	94.7	111.3

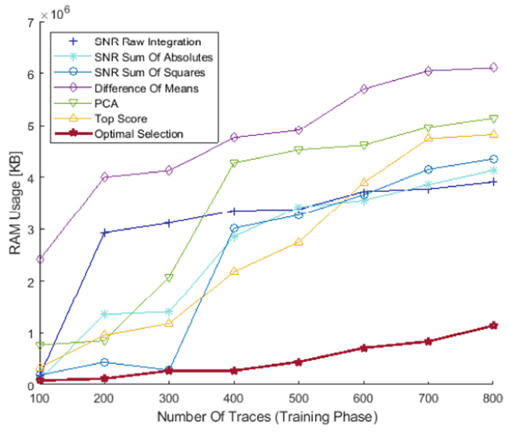
Table 3. Convergence rate for all compression methods

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
TS		0.64	0.66	0.7	0.71	0.71	0.724	0.75	0.785
OS	0.5 *cc*	0.44	0.46	0.48	0.5	0.5	0.52	0.552	0.56
OS	1 *cc*	0.52	0.57	0.61	0.636	0.65	0.653	0.68	0.72
OS	2 *cc*	0.52	0.57	0.61	0.635	0.65	0.65	0.68	0.72
DOM		0.55	0.56	0.58	0.632	0.64	0.642	0.673	0.71
PCA	5 *pc*	0.2	0.24	0.24	0.23	0.25	0.257	0.271	0.29
PCA	10 *pc*	0.22	0.24	0.24	0.246	0.252	0.26	0.274	0.3
PCA	30 *pc*	0.22	0.241	0.244	0.248	0.255	0.26	0.27	0.3
IR	0.5 *cc*	0.06	0.07	0.07	0.08	0.09	0.092	0.092	0.097
IR	1 *cc*	0.18	0.19	0.2	0.2	0.204	0.21	0.219	0.22
IR	2 *cc*	0.1	0.11	0.114	0.114	0.115	0.117	0.12	0.126
SOS	0.5 *cc*	0.04	0.06	0.1	0.1	0.105	0.114	0.14	0.146
SOS	1 *cc*	0.11	0.145	0.17	0.18	0.185	0.2	0.2	0.21
SOS	2 *cc*	0.07	0.1	0.123	0.13	0.128	0.14	0.144	0.16
SOA	0.5 *cc*	0.03	0.03	0.03	0.034	0.04	0.04	0.041	0.047
SOA	1 *cc*	0.11	0.13	0.15	0.15	0.16	0.166	0.17	0.19
SOA	2 *cc*	0.06	0.08	0.09	0.09	0.09	0.098	0.1	0.1

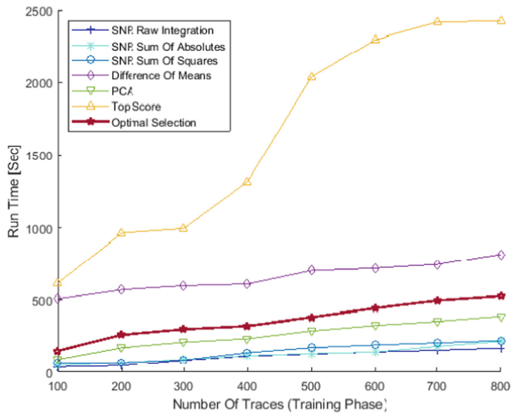
5.1 Observations

We now compare the decoder's best performing configurations for memory consumption, runtime, and convergence rate and make some general observations based on this comparison.

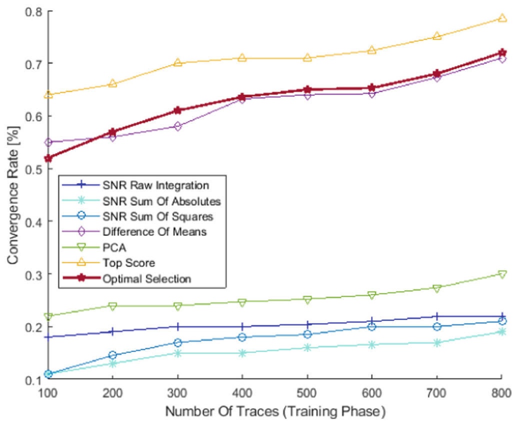
– Testing 0.5 *cc* as a minimum distance between the selected features presented in all measurements of all characteristics provides identical or worse results to the 1 *cc* configuration.
– The 10 *pc* configuration was determined to be the best PCA configuration, since, on average, it has the same runtime as the 5 *pc* configuration, but this is obtained with lower memory consumption. On the other hand, the 10 *pc* configuration has identical accuracy to the 30 *pc* configuration, but the runtime and memory consumption of the 10 *pc* configuration are significantly shorter and lower, respectively, than that of the 30 *pc* configuration.
– For the SNR methods, we determined that the 1 *cc* configuration was the most accurate; its runtime was half that of the 0.5 *cc* configuration, and it also had much lower memory consumption; when compared to 30 *pc*, the 1 *cc* configuration had significantly shorter run-times and lower memory consumption.
– OS has identical runtime and memory consumption results for the various configurations; in terms of accuracy, 1 *cc* and 2 *cc* were found to be identical, so the configuration of 1 *cc* was selected for our overall comparison.



(a) Memory consumption - overall comparison



(b) Run Times - overall comparison



(c) Convergence rate - overall comparison

Fig. 2. Overall comparison

Figure 2 presents a comparison of the performance of all of the compression methods, using the best configuration for each method. Based on this comparison, we can see that TS is never the best method in terms of runtime (see Fig. 2b), and the same can be said for DOM in terms of memory consumption (see Fig. 2a in the Appendix). The SNR-based methods show the same performance trend for all measures, where in most cases, IR is the best performing of these methods and SOA is the worst performing. The PCA method was never abnormally slow or poor performing, but it was always outperformed by at least one of the other methods we compared in this study. Optimal selection (OS), our proposed compression method, had the lowest memory consumption of all of the decoders (see Fig. 2a in the Appendix) and the shortest runtime of the three decoders with the highest accuracy (see Fig. 2c). After TS, it was found to be most accurate with a 72% convergence rate given only 800 traces used in the training phase (see Fig. 2c).

6 Conclusion

In this paper, we strove to advance the profiling step of the template attack, by seeking a practical compression method which requires a smaller dataset and has better performance, both in the online and offline phases. We addressed the challenge of finding the most informative traces regarding the leaked Hamming weight values by building an optimal variant of the state-of-the-art decoder presented in [18], based on the optimal feature selection guidelines of [24].

For the performance challenge, we designed a unique evaluation system which measured runtime, memory consumption, and accuracy. This system used to compare the performance of various decoders, which were found to differ based on the compression methods and configurations used. We demonstrated the importance of both choosing the correct number of principal components for the PCA-based method, and the correct number of clock cycles as a minimum distance between the selected points; one clock cycle was clearly found to be optimal in all of the best performing configurations. In terms of accuracy, the scoring, optimal, and DOM methods outperformed the PCA and SNR-based methods. When considering runtime and memory the opposite is true, except in case of our OS decoder.

The experimental results demonstrate our decoder's ability to outperform the other methods evaluated in terms of memory consumption; however, while it also has shorter run-times than the state of the art, it is slightly less accurate. There is thus a tradeoff in that while our decoder is fast and has low memory consumption, this comes at a cost in terms of the accuracy rate; therefore, our decoder is optimal in cases in which there are time or data restrictions, for example, a small dataset or online data.

References

1. MATLAB PCA. https://www.mathworks.com/help/stats/pca.html
2. MATLAB predict. https://in.mathworks.com/help/ident/ref/predict.html
3. Bohy, L., Neve, M., Samyde, D., Quisquater, J.J.: Principal and independent component analysis for crypto-systems with hardware unmasked units (2003)
4. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
5. Cagli, E.: Feature extraction for side-channel attacks. Ph.D. thesis, Sorbonne University, France (2018)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
7. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 253–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_17
8. Division, C.S.: Announcing the Advanced Encryption Standard (AES). Information Technology Laboratory, Gaithersburg, MD (2001)
9. Elaabid, M.A., Guilley, S.: Practical improvements of profiled side-channel attacks on a hardware crypto-accelerator. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 243–260. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12678-9_15
10. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 15–29. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_2
11. Hogenboom, J.: Principal component analysis and side-channel attacks (2010)
12. Kay, S.M.: Fundamentals of Statistical Signal Processing: Estimation Theory. Signal Processing Series, 1st edn. (1998)
13. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
14. Renauld, M., Standaert, F.-X.: Algebraic side-channel attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16342-5_29
15. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smart Cards. Springer, Cham (2007). https://doi.org/10.1007/978-0-387-38162-6. ISBN 978-0-387-30857-9
16. MathWork: MATLAB fitcnb. https://in.mathworks.com/help/stats/fitcnb.html
17. Oren, Y., Renauld, M., Standaert, F.-X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 140–154. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_9
18. Oren, Y., Weisse, O., Wool, A.: Practical template-algebraic side channel attacks with extremely low data complexity. In: HASP@ISCA, p. 7. ACM (2013)
19. Oswald, D., Paar, C.: Improving side-channel analysis with optimal linear transforms. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 219–233. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37288-9_15
20. ParisTec: DPA contest v4 2013. http://www.dpacontest.org/v4/rsm_traces.php

21. ParisTec: Description of the masked AES - DPA contest v4 (2013). http://www.dpacontest.org/v4/data/rsm/aes-rsm.pdf
22. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. J. Cryptogr. Eng. **7**(4), 343–351 (2017). https://doi.org/10.1007/s13389-017-0172-7
23. Picek, S., Heuser, A., Jovic, A., Batina, L.: A systematic evaluation of profiling through focused feature selection. IEEE Trans. Very Large Scale Integr. Syst. **27**(12), 2802–2815 (2019)
24. Rechberger, C., Oswald, E.: Practical template attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 440–456. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31815-6_35
25. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_8
26. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting time samples for multivariate DPA attacks. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 155–174. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_10
27. Stallings, W.: Cryptography and Network Security, 6th edn. (2014)
28. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Profiling attack using multivariate regression analysis. IEICE Electron. Express **7**(15), 1139–1144 (2010)
29. Weiss, O.: Github - new methods for side channel cryptanalysis code base github (2016). https://github.com/oweisse/dpav4-contest/commits/master
30. Weisse, O.: New methods for side channel cryptanalysis (2013)
31. Zheng, Y., Zhou, Y., Yu, Z., Hu, C., Zhang, H.: How to compare selections of points of interest for side-channel distinguishers in practice? In: Hui, L.C.K., Qing, S.H., Shi, E., Yiu, S.M. (eds.) ICICS 2014. LNCS, vol. 8958, pp. 200–214. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21966-0_15

Design of Intrusion Detection System Based on Logical Analysis of Data (LAD) Using Information Gain Ratio

Sneha Chauhan[1,2](✉) and Sugata Gangopadhyay[1]

[1] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India
sugata.gangopadhyay@cs.iitr.ac.in

[2] Department of Computer Science and Engineering, National Institute of Technology Uttarakhand, Srinagar, India
schauhan1@cs.iitr.ac.in

Abstract. An intrusion detection system is proposed which is capable of detecting penetration, break-ins and other security breaches in near real time. The system has been developed using Logical Analysis of Data (LAD) where the attack is detected by monitoring the network traffic. LAD generates positive and negative patterns from historical observations to classify the unknown observations. It uses the concepts of partially defined Boolean functions and its extensions to extract patterns for classification. The Information Gain ratio technique is used to produce the support set of features. The performance of the proposed technique has an advantage over other techniques as it can detect anomalous behavior in near real time. WEKA tool has been used to build the classifiers and their performance is compared with the proposed model. Detection of abnormal behaviour is significantly achieved by the proposed implementation than the LAD-WEKA.

Keywords: Logical Analysis of Data (LAD) · Intrusion Detection System (IDS) · Partially defined Boolean function (pdBf) · Information gain ratio

1 Introduction

We live in an era of technologies and the Internet has taken up services in almost each and every field, such as education, finance, industry etc. Along with the increase in use of the Internet, the network attacks and security threats have also become the harsh reality in today's world. Denial of Service (DoS) attack, network scanning activity, spreading of malware files are some cyber attacks which exploit the shortfalls of the software and disrupt the routine activities of a system [20]. Firewall, Antivirus and Intrusion Detection Systems (IDS) are some of the softwares which are used to fight against the attackers. An IDS is a software that monitors the network traffic and detect abnormal behavior of the traffic and alerts the system administrator.

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 47–66, 2022.
https://doi.org/10.1007/978-3-031-07689-3_4

The intrusion detection algorithms can be broadly classified into two types: Misuse detection and Anomaly detection [14]. In misuse detection, rules are generated from previous attack signatures to look for an attack. In this type of IDS, new attacks cannot be detected as they may not have similar patterns as the known attacks. The anomaly detection is based on the hypothesis that the suspicious activity has different pattern from the normal activity [1]. If a new activity has deviations from the known behavior, it is classified as an attack by an anomaly based IDS. Such IDSs are useful in detecting new (zero-day) attacks.

The detection of cyber attacks by monitoring the network audit trails was first introduced by Denning [13]. The IDS model proposed by Denning was based on the assumption that abnormal behavior in system usage can be used to detect attacks. Machine learning algorithms and soft computing techniques have been used to develop IDS in recent times. The techniques like Neural Network, Self Organizing Map (SOM), Decision Tree, Support Vector Machine (SVM) etc. are machine learning algorithms that have performed excellent in intrusion detection [20].

Performance statistics of such systems are better as they have low false positive rates but parameters such as usability, competence and accuracy make them unreliable in a system that handles large volume of data on a daily basis. The training process of IDS model may require huge amount of data, which is a challenge, as sufficient data given a specific problem domain, may not be available. Also such models require high computational power and memory to train large amount of data. Due to the complex decision making process, these models may not detect anomalous behavior in real time.

In order to detect and prevent an intrusion, it is important to have knowledge of the patterns related to normal and abnormal activity. LAD is a technique that focuses on generating patterns that can differentiate the abnormal behavior from the normal behavior. The LAD technique was first proposed by Hammer et al. [10,16] for binary data. In this they used partially defined Boolean function to find the suitable combination of food items resulting in headache.

Logical Analysis of Data is a technique that aims at study of numerical data based on combinatorial and optimization approach. The solution of this technique can help us to understand how a certain phenomena works, the factors which govern them and also find ways which can directly affect their development. The Logical Analysis of Data can be used to solve variety of problems including classification, detection of abnormalities and inconsistencies in the network traffic and databases, selection of features, analysis of medical data [2], recognition of patterns for decision support system etc.

In this paper, we have combined LAD with Gain ratio method to design an intrusion detection system. The C4.5 is a classification algorithm which produces decision rules based on entropy and information gain ratio pair of each feature. The feature having maximum Information Gain Ratio will be selected. The Information Gain Ratio criteria has been included with LAD to select features. The LAD technique is then used to extract patterns for classification.

The significant contribution of the paper are: 1) Use of information gain ratio to calculate the discriminating power of features and generate support set based on the score. 2) Developed an IDS model based on LAD technique that can detect intrusions in near real time. 3) Compared different classifiers using WEKA tool.

The paper is organized as follows. In Sect. 2, existing intrusion detection models have been described. An implementation of LAD based IDS model is presented in Sect. 3. Section 4 describes the results obtained by the proposed model and comparative results are also mentioned. Finally, the study is concluded in Sect. 5.

2 Related Work

The author in the paper [24] has proposed a technique to detect intrusions using Classification and Regression trees. A decision tree is built to verify the incoming traffic depending on the data available. The approach to build the system involves 3 stages: the preprocessing stage, normalization stage and decision tree building. The preprocessing stage assigns a random number to each string present in the dataset as strings cannot be compared directly. The dataset obtained from this stage may not be uniform. The next stage normalizes the dataset to provide the characteristic data shrink. In the last stage, decision tree is obtained by applying the Classification and Regression Trees methodology. The results are analyzed based on the computation time.

The paper [7] focuses on implementing decision tree and KNN techniques on IDS and evaluate their performance based on their accuracy. Data preprocessing is carried out to handle the missing data and also to apply one hot encoding to convert categorical data to quantitative variables. Feature selection process is important in order to remove redundant attributes from the data set. Feature selection technique consists of 4 steps: 1. Selection of required features for a particular problem. 2. Function that evaluates the set of features selected. 3. A criteria that will identify if the features are able to stop the search. 4. A validation process to assess the quality of features. In order to find the features for a label, the univariate selection with ANOVA F-test is performed which determines the relationship between features and label. Decision tree is used as a classifier in the paper. To classify the data, gini impurity is used. Gini impurity is a variation of entropy that measures how often a random element from the dataset is incorrectly labeled if it was randomly classified according to the distribution of labels in the subset. Decision tree evaluates all possible outcomes and makes the best possible decision. But any small change in the dataset will lead to large change in the decision tree.

Another classifier used here is KNN(K-nearest neighbour). This classifier produces results only when requested as it does not have a learning phase. It has to compute the value of K and Euclidean distance only so it is simple and fast algorithm. It is efficient for multiclass problems.

Both the classifiers are tested on the NSL-KDD dataset and result is shown by confusion matrix. As per the result obtained, the overall performance of decision tree is better than KNN with accuracy of 99.15%.

Mishra et al. [20] provided a survey in which they have tried to find out the drawbacks of various machine learning techniques in detecting the intrusions. They have also discussed the importance of factors in selection of algorithms to detect a specific type of attack. Machine learning techniques consists of two phases: Training where mathematical calculations are performed on the training dataset to learn the patterns in the traffic and Testing where a new observation is classified as positive or negative based on the learned patterns. Following are the techniques used to detect intrusions:

Decision Tree: A tree like structure is developed to illustrate the outcome of a decision. This method works on both discrete and continuous dataset. The three elements of a tree are decision node which represents a condition over a feature, branch indicates the possible values of a feature and leaf node represents the class label to which the instance belongs. **ID3** algorithm uses greedy search approach in which the conditions are selected based on information gain criteria. In ID3 algo, the data may be overfitted and overclassified. Also it cannot handle missing values and numeric features. A better version of ID3 is **C4.5**. It can handle discrete and continuous values and also use error based pruning technique to solve overfitting. Information gain ratio is used to split the tree. CART algorithm uses towing criteria to split tree. Logistic Model Tree (LMT) uses linear regression model to develop decision trees. Decision trees outperform other classifiers as it uses entropy and information gain for feature selection. The feature having higher information gain is more capable of discriminating the output classes. In a decision tree, finding the probabilities of different branches possible, choosing the best split for each node and pruning are complicated tasks requiring high computational cost.

Naive Bayes Classifier: This classifier can detect intrusions at a high speed and its design is simple compared to other classifiers. Due to its assumption that features are independent, it is not suitable to use it on KDD'99 dataset. However, Hidden Naive Bayes is an exception as it achieves an accuracy of 99.6% for detecting DoS attack.

The researchers in [6] proposed a deep neural network based Network Intrusion Detection (NID) model. Four hidden layers have been used with Rectified Linear Unit (ReLU) as the activation function. Principal Component Analysis (PCA) has been applied for dimensionality reduction. This model is applied to NSL-KDD, UNSW-NB15 and CSE-CIC-IDS2018 dataset. Each dataset was split into 70% training and 30% testing set. The Accuracy achieved for the datasets were CSE-CIC-IDS2018 (76.47%), UNSW-NB15 (89.99%) and NSL-KDD (97.89%).

The paper [5] describes the use of machine learning classifier such as Decision Tree, Random Forest and XGBoost for detecting malicious behaviour in the Software Defined Network. NSL-KDD dataset has been used to carry out the experiment. Five features were selected after some trials. Tree based algorithms have been used to do multi class classification. To evaluate the model's performance, accuracy, precision, recall and F1-measure were used.

The authors of the paper [18] have introduced a wrapper approach that incorporates Genetic Algorithm (GA) as a feature selection and Logistic Regression (LR) as a learning method. This approach is used to find the feature subset which has minimum features but gives maximum classification accuracy. To perform the classification, C4.5, Random Forest (RF) and Naive Bayes Tree (NBTree) which are different categories of decision tree classifiers are used. The performance of the proposed approach is measured on KDD99 and UNSW-NB15 datasets.

In the paper [11], authors have proposed an Anomaly Detection System (ADS) that can detect anomalies by monitoring at physical level. Thus, fault can be identified in real time leading to financial and operational benefits. The disadvantage suffered by most of the techniques used to model the Cyber Physical System(CPS) IDS is the requirement of high computing power and time. Thus, with this proposed model, the authors have tried to achieve near real time detection capability and also provide reason behind the anomalous behavior which may help in prevention and localization of error. The patterns or signatures from historical observations have to be extracted which can differentiate normal and abnormal activity. A LAD based classifier is a binary classifier with the ability to explain the classification result using rules. In this paper, a rule based Anomaly Detection System is designed using the data from a secure water treatment testbed (SWaT). When a plant starts operating, initially it remains unstable. Similarly, when an attack occurs, the system becomes unstable for some duration. So the aim of Anomaly detection system should be to detect attacks as well determine the stability of the system. Two LAD based classifier have been used to achieve these objectives. The first classifier uses pure patterns to label records as stable or unstable. The second classifier uses impure patterns to label the unstable records as attack or normal. The designed classifier has the best F1 score among the other classifiers. The LAD based ADS can perform using laptop class processing power.

One disadvantage of LAD is that if data is large then processing will be difficult as patterns cannot be learned incrementally. In case of power networks, in order to cope with failure or faults, system dynamics may be altered which may change the behavior of normal patterns. Thus, the LAD classifier has to be redesigned.

3 Proposed Work

The computer systems may have vulnerabilities which lead to network intrusions compromising the confidentiality, integrity and availability of the services. In order to develop an effective IDS model, use of logical analysis of data is proposed.

The Logical Analysis of Data (LAD) is a data analysis method that uses combinatorics and optimization models for classifying the observations into positive and negative results. The LAD combines differentiation/integration approach on a subspace of $\mathbb{R}^n$ which contains positive and negative observations and also the new ones. In the differentiation step, a family of small subsets of $\mathbb{R}^n$ which has same structural properties as well as strong positive and negative characteristics is identified. In the integration step, union of subsets of such positive or negative subsets are proposed as approximations of the areas of $\mathbb{R}^n$ containing the positive or negative new or old observations [2]. The basic steps of LAD are described in the following subsections.

3.1 Binarization

The Logical Analysis of Data (LAD) was originally developed to deal with analysis of datasets having attributes that take only binary (0–1) values. In real life scenario, the features mostly take real values so a binarization method was proposed. The idea behind binarization method is to associate several binary attributes to each numerical attribute. A numerical attribute α is represented by a binary attribute using two types of Boolean variables, i.e., level and interval variables. The parameter that decide what values will these variables will take are called cut-points. A level variable $b(\alpha, c_p)$ is introduced such that each binary attribute either takes value 1 or 0 according to the value of the numerical attribute lying above or below a certain threshold given by a cut-point c_p [17].

$$b(\alpha, c_p) = \begin{cases} 1, \text{if } \alpha \geq c_p. \\ 0, \text{otherwise} \end{cases}$$

Interval variables $b(\alpha, c_p^i, c_p^j)$ are introduced for each pair of cutpoints c_p^i and c_p^j as follows

$$b(\alpha, c_p^i, c_p^j) = \begin{cases} 1, \text{if } c_p^i \leq \alpha < c_p^j. \\ 0, \text{otherwise} \end{cases}$$

The cut-points should be chosen such that they can easily distinguish between positive and negative observations. Consider an attribute X with values $X_1, X_2, \dots X_n$ arranged in descending order. In order to maintain the disjointness of the positive and negative set of observation denoted by Ω^+ and Ω^-, only those threshold values are considered which belong to the interval X_i, X_{i+1} where X_i and X_{i+1} are the values that produce different result (one being a positive observation and other being negative one). The threshold value is given by $\frac{X_i + X_{i+1}}{2}$ for each of the intervals.

The binarization method is described here. Table 1 is the table with real world data [8]. The first and fourth attributes in this table are numerical while second and third column correspond to nominal attributes. A nominal attribute is converted into binary by associating each value v_i of α with a Boolean variable $b(\alpha, v_i)$ such that

$$b(\alpha, v_i) = \begin{cases} 1, \text{if } \alpha = v_i. \\ 0, \text{otherwise} \end{cases}$$

Table 2 is obtained after binarization using the level variables given in Table 3 and the interval variables shown in Table 4.

Table 1. Original table

	x_1	x_2	x_3	x_4
S^+	1	Green	Yes	31
	4	Blue	No	29
	2	Blue	Yes	20
	4	Red	No	22
S^-	3	Red	Yes	20
	2	Green	No	14
	4	Green	No	7

Table 2. Binarized table

	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}
a	0	0	0	1	0	0	1	1	1	0	0	0	0
b	1	1	1	0	1	0	0	1	1	0	0	0	0
c	1	0	0	0	1	0	1	1	0	1	1	0	1
d	1	1	1	0	0	1	0	1	1	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	1	1	1
f	1	0	0	1	0	0	0	0	0	1	1	0	0
g	1	1	1	1	0	0	0	0	0	0	0	0	0

Table 3. Level variables

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9
$x_1 \geq 1.5$	$x_1 \geq 2.5$	$x_1 \geq 3.5$	$x_2 = green$	$x_2 = blue$	$x_2 = red$	$x_3 = yes$	$x_4 \geq 17$	$x_4 \geq 21$

Table 4. Interval variables

b_{10}	b_{11}	b_{12}	b_{13}
$1.5 \leq x_1 < 2.5$	$1.5 \leq x_1 < 3.5$	$2.5 \leq x_1 < 3.5$	$17 \leq x_4 < 21$

3.2 Support Set Minimization

The data set obtained by binarization of numerical attributes may contain a number of redundant attributes which needs to be eliminated. A set of binary features is called Support Set if the dataset obtained after elimination of all other features will remain "contradiction-free" (i.e. the positive and negative observations are disjoint) [8]. A support set is said to be irredundant or minimal if eliminating an attribute from the set, results in the collision of positive and negative observation i.e. there exists an observation in the dataset that is positive as well as negative. The problem of obtaining a minimal support set for a binary dataset can be considered as a set covering problem. A few algorithms on solving the set cover problem are discussed in [4,10]. In order to find a minimal support set for our binary dataset, we have proposed the use of Information Gain ratio to select the best attribute in our dataset. The gain ratio is used in the C4.5 algorithm to select the best feature for building the decision tree [19]. The information gain ratio of each attribute is calculated based on its entropy.

The attribute having high information gain ratio is added to the support set. The entropy for binary classification is calculated using the Eq. 1 where D represents the dataset. Assume a feature X has n distinct values then the dataset can be partitioned into n disjoint subsets. The information gain of a feature X in dataset D is calculated in Eq. 2 using the entropy given in Eq. 1. In order to find the gain ratio, we use split information given in Eq. 3 as it normalizes the information gain [19]. The support set generated for the data given in Table 2 is $\{b_8, b_{12}\}$.

$$E(D) = -(P(0) * log_2(P(0)) + P(1) * log_2(P(1))) \tag{1}$$

$$\text{Info-Gain(D,X)} \ = E(D) - \sum_{i=0}^{n} \frac{|D_i|}{|D|} * E(x) \tag{2}$$

$$\begin{aligned} \text{SplitInfo (X)} \ &= -\sum_{j=1}^{n} \frac{|D_j|}{|D|} * log_2 \frac{|D_j|}{|D|} \\ \text{Gain Ratio} \ &= \text{Info-Gain(D,X) / SplitInfo (X)} \end{aligned} \tag{3}$$

3.3 Pattern Generation

In Boolean algebra, a Boolean variable or its complement is called a literal. The conjunction of literals is called a term. A term T covers a point $p \in \{0,1\}^n$ if $T(p) = 1$. A term covering only positive (negative) observations is termed as positive (negative) pattern [11].

Pattern play an important role in LAD in detection of subclasses, selection of feature, classification and other problems. A positive ω - pattern where $\omega \in \{0,1\}^t$, is a pattern that covers ω. A maximum positive ω - pattern P is a positive ω - pattern having maximum coverage (i.e. the cardinality of $|P \cap \Omega_S^+|$ is maximum where Ω_S^+ is the set of positive observations projected on support set S). Similarly, a maximum negative ω - pattern can be defined [17]. A pattern can be generated using a combinatorial enumeration technique which is explained below. Let us associate an elementary conjunction C which is a product of some complemented Boolean variables and some uncomplemented variables to a binary vector $\omega = (\omega_1, \omega_2, \ldots, \omega_t) \in \{0,1\}^t$. Next, we define a binary variable $y_i (i = 1, 2, \ldots, t)$ which is associated to each attribute $x_i (i = 1, 2, \ldots, t)$. If an attribute x_i is in support set S then $y_i = 1$ otherwise $y_i = 0$. Now,

1. if $\omega_i = y_i = 1$ then x_i is included in C,
2. if $\omega_i = 0, y_i = 1$ then $\overline{x_i}$ is included in C,
3. if $\omega_i = y_i = 0$ then neither x_i nor $\overline{x_i}$ is included in C.

In our implementation, we have used Breadth first enumerative technique which involves a top-down approach using bottom-up strategy. Bottom-up approach is used to generate small degree patterns and then top-down approach is applied to cover all the remaining uncovered observations. Using this technique, all the positive patterns are produced at each stage d along with the candidate terms which are to be examined in the next stage $d+1$. A candidate term covers at least one positive and one negative observation. The terms of degree d are obtained at stage d by eliminating any of its literals from the candidate term generated at stage $d-1$. These terms of degree d are partitioned into 3 sets:

- P_d is the set of terms that cover only positive observation and no negative observation.
- C_d is the set of candidate terms covering at least one positive and one negative observation.
- G_d is the set of remaining terms.

Hence, P_d is the set of all positive patterns of degree d. In order to generate terms of degree $d+1$, literals are added in lexicographical order to a term $T \in C_d$. Consider the literals in T be in order $i_1 < i_2 < \ldots < i_d$. Let a literal $i > i_d$ is added to T to obtain T_1. Now T_2 is obtained by dropping i^{th} literals. If $T_2 \notin C_d$, then no need to examine T_1 as T_1 cannot be a prime pattern or a candidate term [8]. Now we have to check whether T_1 covers at least k positive observations and no negative observation then T_1 is a prime pattern otherwise $T_1 \in C_{d+1}$. The value of k can be set greater than or equal to 1. If value of k is correctly chosen then it ensures that 90–95 % instances are covered by the patterns. The authors in the paper [11] and [12] have described the pattern generation algorithm which we have used in our model to generate the patterns. The Algorithm 1 describes how the patterns are generated to distinguish normal observation from abnormal ones.

3.4 Classifier Design

The patterns generated above are transformed into rules which are used to build a classifier. Let's consider $b_8\overline{b_{12}}$ is the pattern obtained for the data given in Table 2. The meaning of b_8 is whether $(x_4 \geq 17)$ is true or false. Similarly, b_{12} means whether $(2.5 \leq x_1 \leq 3.5)$ is true or false. Thus the rule generated from the pattern $b_8\overline{b_{12}}$ is $(x_4 \geq 17) \wedge \neg(2.5 \leq x_1 \leq 3.5) \implies L = 1$. The pseudo code for the above rule is as follows:

```
if ($(x_4 \geq 17) \wedge \neg(2.5 \leq x_1 \leq 3.5)$) then
Class Label $L = 1$
end if
```

Similarly, rules from other positive patterns can be combined into an 'if else' structure to build a classifier [12].

Algorithm 1. Pattern Generation Algorithm

```
Input: $\Omega_S^+, \Omega_S^- \subset \{0,1\}^n$
D: maximum degree of generated patterns
n : no. of input variables
Result: $P$ : Set of prime patterns.
$P = \phi$
$C_0 = \{\phi\}$
for $d = 1, \ldots, D$ do
    if $d < D$ then
        $C_d = \phi$
    end
    for $T \in C_{d-1}$ do
        $p =$ maximum index of literal in $T$
        for $s = p+1, \ldots, n$ do
            for $l_{new} \in \{l_s, \bar{l_s}\}$ do
                $T_1 = T||l_{new}$
                for $i = 1, \ldots, d-1$ do
                    $T_2 =$ remove $i^{th}$ literal from $T_1$
                    if $T_2 \in C_{d-1}$ then
                        Continue
                    else
                        Break and continue with next term of $C_{d-1}$
                    end
                end
            end
            if $k \leq \sum_{v \in \Omega_S^+} T_1(v)$ then
                if $1 \notin T_1(\Omega_S^-)$ then
                    $P = P \cup \{T_1\}$
                    Delete the covered observations from the dataset
                else
                    if $d < D$ then
                        $C_d = C_d \cup \{T_1\}$
                    end
                end
            end
        end
    end
end
```

4 Results and Discussion

For our experiment, NSL-KDD and UNSW-NB15 dataset is used. NSL-KDD is a widely used dataset for IDS models. The dataset contains 41 features including the class label. Different attacks are presented in the dataset which we have combined to form a single class label as 'anomaly'. There are two classes in our dataset namely, 'normal' and 'abnormal or anomaly' represented by 1 and 0 respectively. For training the classifier, we have used KDDTrain_20 percent dataset containing 25000 observations and to test the classifier, KDDTest$^+$ and KDDTest21 are used. Both the datasets are part of NSL-KDD dataset.

The UNSW-NB15 is a more recent dataset as compared to NSL-KDD dataset. As per [21], UNSW-NB15 includes modern normal and contemporary attack scenarios of the network traffic. This dataset consists of observations that are grouped under nine different cyberattack categories namely Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Shellcode, Reconnaissance and Worms and rest of the observations are under Normal category. For performing binary classification, we have labelled all the attack observations as Attack/Abnormal or 0 and normal observations as Normal or 1. There are 43 features including class label in the dataset. Three of them are categorical features and rest are numeric. There are two sets of dataset, one for training which has 82,332 observations and other one is for testing having 175,341 observations.

The experiments have been performed using HP Z440 Workstation loaded with Windows 10 with Intel Xeon processor and 32 GB RAM and using python language. The performance of the intrusion detection system is evaluated using accuracy, sensitivity, precision, specificity and F1-score. The confusion matrix shown in Table 5 is considered for our experiment. For comparing the results of the classifiers, WEKA tool has been used to build the machine learning IDS models on the same dataset.

Table 5. Confusion matrix

		Predicted label	
		Normal	Abnormal
True label	Normal	True positive	False negative
	Abnormal	False positive	True negative

The first step of LAD is binarization which is a preprocessing step for generating binary variables for all the numeric variables. In the binarization step, both level and interval variables are generated for all features. In order to avoid large number of binary variables associated with a feature, we have used a threshold of 175 i.e. if number of cutpoints of a feature is greater than 175 then that feature is neglected. The idea behind the threshold is that more number of cutpoints for a feature indicates more randomness in the feature. Thus, such a feature will have less influence on the classification of observations. In this step, 11565 binary variables were produced in case of NSL-KDD dataset. For the UNSW-NB 15 dataset, binarization process produced a dataset of 4804 binary variables.

The second step is support set minimization in which we have used Information Gain ratio criteria to measure how effective is the feature on the classification. For NSL-KDD dataset, total 42 features were selected depending on their discriminating power. 30 features were obtained in the support set for the UNSW-NB15 dataset.

In pattern generation step, patterns are extracted using the reduced dataset obtained after the second step. In the Algorithm 1, k denotes the number of observations covered by a pattern. If $k = 1$ then we noted that many patterns are generated and they overlap with each other. Hence after conducting experiments, we fixed k value at 60 for NSL-KDD dataset. Twenty one positive patterns of degree 3 and 4 are generated with the NSL-KDD dataset. In case of UNSW-NB15 dataset, 17 positive or normal patterns of degree 3 and 4 are generated where each pattern covers more than 100 normal observations and less than 10 abnormal observations. This value has been considered to cover maximum observations in the dataset.

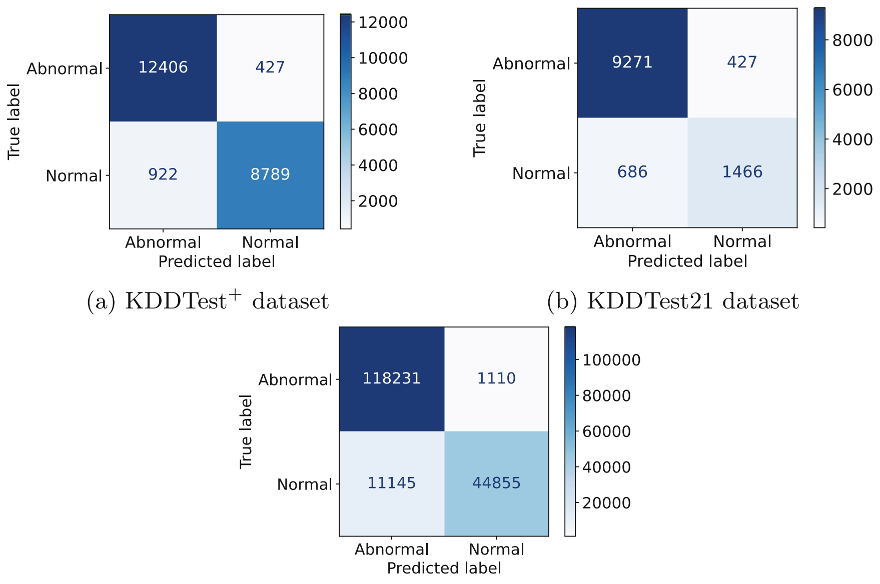
The classifiers for both the datasets (NSL-KDD and UNSW-NB15) are developed using the patterns obtained from the pattern generation process.

The KDDTest$^+$ and KDDTest21 dataset are used to validate the classifier having 21 rules. The confusion matrix for KDDTest$^+$ and KDDTest21 dataset obtained by the proposed method are shown in Fig. 1. We can deduce from the confusion matrices that most abnormal observations are correctly classified. The proposed method correctly classifies 12406 abnormal observations on KDDTest$^+$ and only 427 abnormal observations are incorrectly classified as normal. Thus, the specificity is 96.67% and only 3% abnormal observations were incorrectly detected by the proposed IDS. 922 normal observations out of the total 9411 observations, have been incorrectly classified as attack. Therefore, the precision and recall value is 95.36% and 90.5% respectively. Based on the confusion matrices, the overall performance of the classifier on both datasets in terms of accuracy, sensitivity, specificity and F1-score are shown in Table 6. The proposed classifier achieves accuracy of 94.02% and precision 95.36% on the KDDTest$^+$. The results for the KDDTest21 are lower than that of KDDTest$^+$ as the instances which are easily classified by KDDTest$^+$ have been removed from the KDDTest21.

Table 6. Result of the proposed IDS based on LAD

Dataset	Accuracy	Precision	Sensitivity	Specificity	F1-score
KDDTest$^+$	0.940	0.953	0.905	0.966	0.929
KDDTest^{21}	0.906	0.774	0.681	0.956	0.723
UNSW-NB15	0.930	0.975	0.801	0.991	0.875

For the UNSW-NB15 dataset, the classifier has 17 rules which is validated by the UNSW-NB15 testing dataset. The confusion matrix for the UNSW-NB15 dataset (Fig. 1c) shows that the classifier is able to detect abnormal observations with detection rate of 99.1%. The precision is 97.5%. The sensitivity is 80.1% as many normal instances have been misclassified as attack. The overall performance of the classifier on the UNSW-NB15 dataset is given in the Table 6.



(a) KDDTest$^+$ dataset

(b) KDDTest21 dataset

(c) UNSW-NB15dataset

Fig. 1. Confusion Matrix for LAD based IDS

Table 7 shows the performance comparison of our proposed classifier with some of the classifiers mentioned in the paper [5] and [9] with respect to accuracy, sensitivity, specificity and F1-score. From the table, we can conclude that our method has second highest accuracy, first being the XGBoost algorithm [5]. The sensitivity and F1 score is low compared to XGBoost algorithm as our classifier has misclassified normal instances as attack. But our method has a detection rate of 96.67% for abnormal instances which is higher than the other methods. The Fig. 2 shows that our LAD based IDS has an equivalent performance with other methods with accuracy of 94.02%. We have included only those results which were obtained on the smaller dataset i.e. only 20% of the NSL-KDD dataset. In some research, 10 fold cross validation and splitting of the dataset has been used so those results have not been shown here.

The classifier on UNSW-NB15 dataset is compared with other machine learning classifiers based on accuracy and false alarm rate. The LAD based IDS has achieved an accuracy of 93.01% which is higher than the classifiers like Support Vector Machine (SVM) (90.11%), J48 (90.48%) [3], Geometric Area Analysis-Anomaly-based Detection System (GAA-ADS) (91.8 %), GAA-principal components (92.8%) [23] etc. The false alarm rate of our model is 10.41% which is higher than the GALR-DT method (6.39%) [18] due to high false negatives. These results are shown in the Table 8.

Apart from python implementation, we have used WEKA (Waikato Environment for Knowledge Analysis) tool to implement traditional classifiers and LAD on the NSL-KDD and UNSW-NB15 dataset. WEKA consists of 49 tools

Table 7. Performance of our proposed method compared to other machine learning methods for NSL-KDD dataset

Classifiers	KDDTest$^+$ accuracy	Sensitivity	Specificity	F1-score
BayesNet	90.66	85.8	96.18	90.7
Logistic	84.96	87.34	82.2	86.0
J48	89.67	86.48	93.29	89.9
Random Forest	91.52	88.68	88.04	91.7
RandomTree	90.87	87.74	94.43	91.1
XGBoost	95.5	98.0	–	95.55
Proposed IDS	94.02	90.5	96.67	92.9

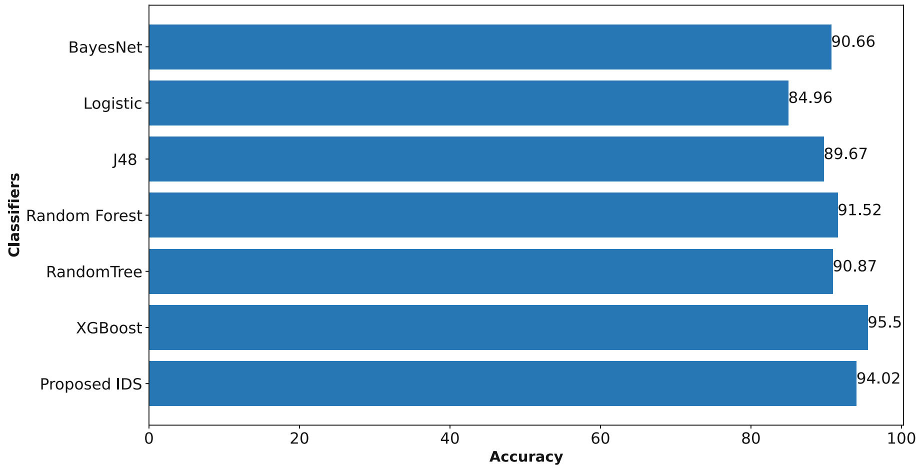


Fig. 2. Performance comparison of proposed IDS based on LAD against machine learning algorithms based on the accuracy using NSL-KDD dataset.

for data preprocessing, 15 methods for attribute evaluation, 10 feature selection techniques and 76 classifiers [15]. The training and testing dataset are same as used in the above implementation. We have used Naive Bayes, SVM, Logistic Regression, J48, Random Forest, Random Tree and LAD classifiers on both the datasets. The WEKA configuration used is presented below:

weka.classifiers.bayes.NaiveBayes, weka.classifiers.functions.LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model, weka.classifiers. functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4, weka.classifiers.trees. J48 -C 0.25 -M 2, weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1, weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1.

The performance of all the classifiers evaluated using WEKA is shown in the Table 9 and Table 10 along with the results of our classifier on respective datasets. Table 9 shows that our method has achieved higher accuracy, specificity and

Table 8. Comparison Results of implemented classifier with other ML methods on UNSW-NB15 Dataset

Classifiers	Accuracy (%)	FAR (%)
SVM [3]	90.11	–
J48 [3]	90.48	–
GALR-DT [18]	81.42	6.39
DT [22]	85.56	15.78
LR [22]	83.15	18.48
NB [22]	82.07	18.56
ANN [22]	81.34	21.13
EM [22]	78.47	23.79
GAA-ADS-original features ($K = 10$) [23]	91.8	–
GAA-principal-components ($K = 10$) [23]	92.8	–
Proposed IDS	93.01	10.41

Table 9. Comparison of proposed IDS with the results obtained using WEKA on NSL-KDD Dataset

Classifiers	KDDTest$^+$ accuracy	Sensitivity	Specificity	F1-score
NaiveBayes	0.765	0.926	0.643	0.773
SVM	0.712	0.980	0.509	0.746
Logistic Regression	0.745	0.929	0.606	0.758
J48	0.790	0.971	0.653	0.80
Random Forest	0.775	0.973	0.625	0.788
RandomTree	0.784	0.970	0.644	0.795
LAD-WEKA	0.708	0.982	0.501	0.744
Proposed IDS	0.940	0.905	0.966	0.929

F1-score over NSL-KDD dataset. The sensitivity or recall of all the classifiers is in range of 0.90 to 0.98 which shows that they are able to distinguish the normal observation correctly as normal observation is considered positive in our experiment. But the detection of abnormal observation or negative observation is not satisfactory as the specificity of all the classifiers is below 0.75. WEKA also has an implementation of LAD. The result of which has been included in the Table 9. The Fig. 3 shows the comparison of our LAD with respect to LAD-WEKA tool. From the Fig. 3, it is evident that the accuracy of our LAD python implementation is better than LAD-WEKA which is 94.02%. Though LAD-WEKA has higher sensitivity, the true negative rate and F1-score of our model is 96.6% and 92.9% which is far better than LAD-WEKA (in our implementation, abnormal observation is considered negative). Thus, our LAD implementation performs better than LAD-WEKA.

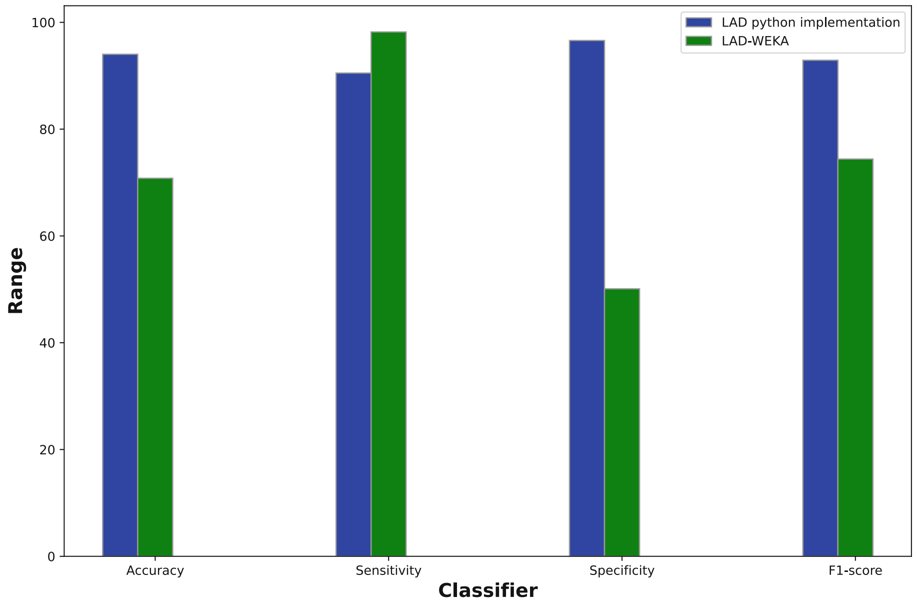


Fig. 3. Comparison of the proposed LAD implementation with LAD-WEKA on NSL-KDD dataset, using Accuracy, Sensitivity, Specificity and F1-score.

Table 10. Comparison of proposed IDS with the results obtained using WEKA on UNSW-NB15 dataset

Classifiers	Accuracy	Sensitivity	Specificity	F1-score
NaiveBayes	0.625	0.933	0.481	0.614
SVM	0.677	0.998	0.527	0.664
Logistic Regression	0.831	0.941	0.780	0.781
J48	0.894	0.975	0.855	0.854
Random Forest	0.900	0.980	0.862	0.862
RandomTree	0.885	0.967	0.846	0.843
Proposed IDS	0.930	0.801	0.991	0.875

With respect to UNSW-NB15 dataset, Table 10 has all the results obtained using WEKA. NaiveBayes and SVM do not achieve good accuracy on this dataset. Random Forest, J48 and Random Tree have achieved an accuracy of 90.0%, 89.4% and 88.5% which is comparable to our IDS model with accuracy 93%. These classifiers have high sensitivity as they have correctly classified more normal instances compared to our model. Our model is able to detect abnormal behaviour better than the WEKA classifiers which is shown by the specificity of our model being 99.1%. It is highest among all the classifiers. The F1-Score of our model is 87.5% which is highest with respect to other WEKA classifiers.

5 Conclusion

In our paper, we have used NSL-KDD and UNSW-NB15 datasets, popular datasets for intrusion detection, to train and test our LAD based IDS model. LAD is the data analysis technique which works on combinatorics and optimization. To reduce the dimension of the binarized dataset, information gain ratio is used, for selecting minimal support set. The patterns generated are able to classify the unknown observations. The study shows that the LAD based IDS is robust and has similar performance with the existing techniques using less amount of data and computational resources. The machine learning classifiers are trained and tested on WEKA tool and average performance has been recorded. Further, we have compared our python implementation of LAD with the LAD-WEKA tool and results show that our model has performed significantly better than the LAD-WEKA tool in detecting attacks for NSL-KDD dataset. Even though some techniques perform better than the LAD based technique in few attack scenarios, these techniques cannot localize the error and also they cannot perform detection in near real time. LAD has an advantage over other machine learning techniques as it is able to identify the features involved in an attack. Thus we can focus on those attributes which are more vulnerable for intrusions. In future, the performance of LAD can be improved further by training it on larger dataset and also combining it with other feature selection techniques and classifiers.

References

1. Ahmed, M., Mahmood, A.N., Hu, J.: A survey of network anomaly detection techniques. J. Netw. Comput. Appl. **60**, 19–31 (2016). https://doi.org/10.1016/j.jnca.2015.11.016
2. Alexe, G., Alexe, S., Bonates, T.O., Kogan, A.: Logical analysis of data-the vision of Peter L. Hammer. Ann. Math. Artif. Intell. **49**(1–4), 265–312 (2007). https://doi.org/10.1007/s10472-007-9065-2
3. Almomani, O.: A feature selection model for network intrusion detection system based on PSO, GWO, FFA and GA algorithms. Symmetry **12**(6), 1046 (2020). https://doi.org/10.3390/sym12061046
4. Almuallim, H., Dietterich, T.G.: Learning Boolean concepts in the presence of many irrelevant features. Artif. Intell. **69**(1–2), 279–305 (1994). https://doi.org/10.1016/0004-3702(94)90084-1
5. Alzahrani, A.O., Alenazi, M.J.F.: Designing a network intrusion detection system based on machine learning for software defined networks. Future Internet **13**(5) (2021). https://doi.org/10.3390/fi13050111
6. Amaizu, G.C., Nwakanma, C.I., Lee, J.M., Kim, D.S.: Investigating network intrusion detection datasets using machine learning. In: 2020 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1325–1328 (2020). https://doi.org/10.1109/ICTC49870.2020.9289329

7. Ashwini Pathak, S.P.: Study on decision tree and KNN algorithm for intrusion detection system. Int. J. Eng. Res. Technol. (IJERT) **9**, 376–381 (2020). https://doi.org/10.17577/IJERTV9IS050303
8. Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.: An implementation of logical analysis of data. IEEE Trans. Knowl. Data Eng. **12**(2), 292–306 (2000). https://doi.org/10.1109/69.842268
9. Choudhury, S., Bhowal, A.: Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. In: 2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), pp. 89–95 (2015). https://doi.org/10.1109/ICSTM.2015.7225395
10. Crama, Y., Hammer, P.L., Ibaraki, T.: Cause-effect relationships and partially defined Boolean functions. Ann. Oper. Res. **16**(1), 299–325 (1988). https://doi.org/10.1007/BF02283750
11. Das, T.K., Adepu, S., Zhou, J.: Anomaly detection in industrial control systems using logical analysis of data. Comput. Secur. **96**, 101935 (2020). https://doi.org/10.1016/j.cose.2020.101935
12. Das, T.K., Gangopadhyay, S., Zhou, J.: SSIDS: semi-supervised intrusion detection system by extending the logical analysis of data. CoRR arXiv:2007.10608 (2020)
13. Denning, D.E.: An intrusion-detection model. IEEE Trans. Softw. Eng. **2**, 222–232 (1987). https://doi.org/10.1109/TSE.1987.232894
14. Depren, O., Topallar, M., Anarim, E., Ciliz, M.K.: An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. Expert Syst. Appl. **29**(4), 713–722 (2005). https://doi.org/10.1016/j.eswa.2005.05.002
15. Dua, M., et al.: Attribute selection and ensemble classifier based novel approach to intrusion detection system. Procedia Comput. Sci. **167**, 2191–2199 (2020). https://doi.org/10.1016/j.procs.2020.03.271
16. Hammer, P.L.: Partially defined Boolean functions and cause-effect relationships. In: Proceedings of the International Conference on Multi-attribute Decision Making via OR-Based Expert Systems. University of Passau (1986)
17. Hammer, P.L., Bonates, T.O.: Logical analysis of data - an overview: from combinatorial optimization to medical applications. Ann. Oper. Res. **148**(1), 203–225 (2006). https://doi.org/10.1007/s10479-006-0075-y
18. Khammassi, C., Krichen, S.: A GA-LR wrapper approach for feature selection in network intrusion detection. Comput. Secur. **70**, 255–277 (2017). https://doi.org/10.1016/j.cose.2017.06.005
19. Li, L., Yu, Y., Bai, S., Hou, Y., Chen, X.: An effective two-step intrusion detection approach based on binary classification and k-nn. IEEE Access **6**, 12060–12073 (2018). https://doi.org/10.1109/ACCESS.2017.2787719
20. Mishra, P., Varadharajan, V., Tupakula, U., Pilli, E.S.: A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun. Surv. Tutor. **21**(1), 686–728 (2019). https://doi.org/10.1109/COMST.2018.2847722
21. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6 (2015). https://doi.org/10.1109/MilCIS.2015.7348942
22. Moustafa, N., Slay, J.: The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Inf. Secur. J. Glob. Perspect. **25**(1–3), 18–31 (2016). https://doi.org/10.1080/19393555.2015.1125974

23. Moustafa, N., Slay, J., Creech, G.: Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks. IEEE Trans. Big Data **5**(4), 481–494 (2019). https://doi.org/10.1109/TBDATA.2017.2715166
24. Shilpashree, S.: Decision tree: a machine learning for intrusion detection. Int. J. Innov. Technol. Explor. Eng. **8**, 5 (2019). https://doi.org/10.35940/ijitee.F1234.0486S419

Simulating a Coupon Collector

Dina Barak-Pelleg[1](✉) and Daniel Berend[2]

[1] Department of Mathematics, Ben-Gurion University, 84105 Beer Sheva, Israel
dinabar@post.bgu.ac.il

[2] Departments of Mathematics and Computer Science, Ben-Gurion University, 84105 Beer Sheva, Israel
berend@math.bgu.ac.il

Abstract. The coupon collector's problem (CCP) reads as follows: How many drawings are needed on average in order to complete a collection of n types of coupons, if at each step a single coupon is drawn uniformly randomly, independently of all the other drawings?

Since CCP was first introduced, numerous questions have been posed on its basis, and it also turned out to appear in many applications, such as DDoS cyber attacks and machine learning. It is well known that, in CCP, the convergence of various quantities of interest to their asymptotic values is rather slow. Thus, simulating the process to get a feeling for their behavior is often impractical.

We present here an alternative view of the process, which allows us, for equally probable coupons, to perform fast simulation for large values of the parameters.

Keywords: Coupon collector's problem · Simulation

1 Introduction

1.1 The Coupon Collector's Problem

Suppose that a company distributes a commercial product and that each package contains a single coupon. There are n types of coupons, and a customer wants to collect at least one of each. We want to know how many packages need to be bought on the average until getting all coupons. This is referred to as the *coupon collector's problem (CCP)*. The problem goes back at least as far as de Moivre [24], who mentioned it in a collection of problems regarding various games of chance.

The expected number of drawings is calculated in a straightforward manner. (Note, though, that if one does not take the right approach, the problem may become quite intricate; see [23].) After exactly j distinct coupons have been seen, the probability of drawing an as yet unseen coupon is $\frac{n-j}{n}$. Hence, the number D_j of drawings until we see such a new coupon is $\mathrm{Geom}(1-\frac{j}{n})$-distributed. The

Research supported in part by the Milken Families Foundation Chair in Mathematics and the Cyber Security Research Center at Ben-Gurion University.

total number of drawings is the sum of all these D_j's. The expected number of drawings is therefore nH_n, where H_n is the n-th harmonic number:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}, \qquad n = 1, 2, 3, \ldots .$$

Asymptotically, this expectation is $n(\ln n + \gamma) + O(1)$, where $\gamma = 0.577\ldots$ is the Euler-Mascheroni constant. We refer to [13] for more information.

1.2 Various Extensions of the CCP

The problem, and various extensions thereof, have drawn much attention for many years. Laplace [20], and also Erdős and Rényi [12], found the normalized asymptotic distribution of the number of drawings. Schelling [33,34] and Flajolet, Gardy, and Thimonier [14] considered the case where distinct coupons may show up with distinct probabilities. A (very lengthy) formula for the expected required time in this case was given in the latter of these papers. It turns out that various real-world problems may be modelled by this version. For example, see [3,4,29, 30,32] for an application to dealing with DDoS cyber attacks and [2,10,31,35] for applications to machine learning and cryptography.

The Collector's Brotherhood Problem. Suppose the collector wants to obtain at least m copies of each coupon and we ask how much time would he need to obtain his collection.

Denote this quantity by $T_{n,m}$. For fixed m and large n, Newman and Shepp [25] calculated the asymptotic expected number of required drawings

$$E(T_{n,m}) = n\left(\log n + (m-1)\log\log n + C_m + o(1)\right), \tag{1}$$

where C_m is a constant depending on m. This version is known as the *dixie cup problem* [25] or as the *collector's brotherhood problem* [16]. For other questions related to this case, see [11,19].

Given a sequence $(X_n)_{n=1}^{\infty}$ of random variables and a probability law $\mathcal{L}$, write $X_n \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{L}$ if the sequence converges to $\mathcal{L}$ in distribution. Recall that a random variable X is *Gumbel distributed* with parameters $\mu \in \mathbf{R}$ and $\beta > 0$, and we write $X \sim \text{Gumbel}(\mu, \beta)$, if its distribution function is given by [17,26]:

$$F(x) = e^{-e^{-(x-\mu)/\beta}}, \qquad x \in \mathbf{R}.$$

Again, for fixed m, Erdős and Rényi [12] found the limiting distribution of $T_{n,m}$:

$$\frac{T_{n,m} - n(\log n + (m-1)\log\log n)}{n} \xrightarrow[n\to\infty]{\mathcal{D}} \text{Gumbel}\left(-\log(m-1)!, 1\right), \tag{2}$$

and Flatto [15] provided an estimate on the tail of the distribution. This also let Erdős and Rényi find explicitly the constant C_m in (1):

$$E(T_{n,m}) = n\left(\log n + (m-1)\log\log n - \log(m-1)! + \gamma + o(1)\right).$$

Finally, they left open the question as to the distribution of $T_{n,m}$ if m increases with n [12, p. 219].

1.3 Simulating CCP

It has been observed in several coupon collecting problems that the convergence of the various relevant quantities is quite slow [7,18,19]. It is natural to run some simulations of the process to help verifying the behavior of such quantities. However, to obtain reliable asymptotic estimates, one needs to use very large values n (and on top of it run the simulation many times). Several papers discuss simulating the process [1,8,9], but their method is infeasible for large n.

In this paper we develop a fast simulation method, which allows us running thousands of iterations for very large values of n in a matter of minutes. The method relies on an alternative view of the process of collecting the coupons. As mentioned above, the asymptotic behavior of the time for collecting m coupons has not been studied for the case where m grows with n. Thus, to test the performance of our simulation algorithm, we run it on the collector's brotherhood problem, for large values of n and m.

2 A Fast Simulation for the Collector's Brotherhood Problem

A naive simulation of the process is trivial to design and implement. We simply choose a coupon uniformly randomly until all n coupons have arrived at least m times.

The problem is that, since at each step we "draw" a single coupon, each run costs $\Omega(n \log n)$ time w.h.p. Moreover, to save the data regarding the number of times each coupon has been seen, we need $\Theta(n)$ space. On top of that, as convergence of the various quantities in CCP is very slow [7,18,19], to get a feeling for the asymptotics, large values of n need to be examined. Thus, for large n, the naive algorithm becomes infeasible.

Algorithms for simulating the process have been suggested in [1,8,9]. However, even though they are interesting for theoretical purposes, and have the advantage of dealing with coupons of different probabilities also, they are infeasible for large n.

We developed faster and much less space-consuming methods. We first observe that it is unnecessary to know at each stage how many copies of each coupon have arrived by this time. It rather suffices to know how many coupons have:

- not arrived at all;
- arrived once;
- arrived twice;

  ⋮
- arrived $m-1$ times;

– arrived m or more times.

Indeed, this information lets us know what is the probability of obtaining at the next step a coupon who has arrived by now i times, where i is one of the numbers $0, 1, 2, \ldots, m-1$ or is in the range $[m, \infty)$. Thus, the required amount of memory is only $\Theta(m)$.

We start with an array of length m, denoted by `Current`. The first entry of `Current` indicates the number of coupons which have not been seen until now, the second – the number of coupons which appeared once, the third – twice, ..., the m-th – the number of coupons which appeared $m-1$ times. Initially, this array consists of the number n in the first entry and zeros elsewhere. To run the simulation faster, we proceed in multi-step leaps. At each leap, we take a coupon with a minimal number of appearances so far. We draw the time at which its m-th copy is obtained by a negative binomial variate. For example, suppose this coupon had appeared r times before this leap. Thus, $T' \sim \mathrm{NB}(m-r, 1/n)$ (where NB denotes the negative binomial distribution) is the number of steps it will take us to finish collecting m copies of this coupon. Now we discount this coupon from `Current`, namely decrement the first non-zero entry (which is the $(r+1)$-st entry) of `Current`. Consider the other *deficient coupons* – coupons that have arrived less than m times before the leap. (These are precisely the coupons accounted for in `Current`.) We draw binomially, for each of these coupons, the number of times it has arrived throughout the leap. For example, for the first deficient coupon we draw $X'_1 \sim \mathrm{Binom}(T' - m + r, 1/(n-1))$ arrivals, for the second – $X'_2 \sim \mathrm{Binom}(T' - m + r - X'_1, 1/(n-2))$, and so on. In this process, we "move" the coupons which are still deficient from `Current` to a new array `Next` of size m and dispose of the others, as follows: Each treated coupon is discounted from `Current`. If it is still deficient after the leap, then it is accounted for in the cell of `Next`: corresponding to its updated number of arrivals so far; otherwise, it is disposed of. After dealing with all coupons accounted for in `Current`, we replace `Current` by `Next`, reset `Next` to **0**, and continue to the next leap. We repeat the leaps until all coupons have been obtained m times, namely, `Current` consists of zeros. The process is defined more formally in Algorithm 1.

How long does the simulation run? At the first time we perform a leap, we take a coupon (which, as all coupons, has 0 appearances so far), and draw (using a negative binomial variate) the time until it arrives m times. A priori, each of the other coupons has a probability of $\frac{1}{2}$ of arriving m (or more) times before the first coupon does so. Thus, we expect roughly half the coupons to have been obtained at least m times by the time the first leap has finished. Since in the other leaps we take each time a coupon with a minimal number of appearances so far, we may expect to get rid of at least half the deficient coupons at each subsequent leap. Thus, we may expect the number of leaps to be at most about $\log_2 n$. The number of deficient coupons we need to deal with at the k-th leap is at most about $n/2^{k-1}$. Altogether, the runtime of Algorithm 1 is $\Theta(n)$ and the required memory is $\Theta(m)$.

In principle, suppose someone told us, at some point in the process, after some T_0 coupon drawings, how many coupons have not arrived, how many arrived

Algorithm 1: Rapid Coupon Collecting

```
Input: $n$ – number of coupon types,
       $m$ – number of required copies of each type
Output: $T$ – completion time
$T \leftarrow 0$;
Current $\leftarrow (n, 0, \dots, 0)$;                         /* vector of length $m$ */
while Current $\neq \mathbf{0}$ do
    Next $\leftarrow \mathbf{0}$;                              /* vector of length $m$ */
    Take a coupon with a minimal number $j$ of copies:
      $j = \min\{k : \texttt{Current}[k] > 0\} - 1$;
    Draw $T' \sim \mathrm{NB}(m - j, 1/n)$, the additional time required to have $m$ copies of
      this coupon;
    Current$[j+1]$ −−;                  /* discount the coupon from Current */
    $T \leftarrow T + T'$;
    EmptySlots $\leftarrow T' - m + j$;
    $i \leftarrow 1$;                   /* start a counter of the deficient coupons */
    while Current $\neq \mathbf{0}$ do
        Take any coupon accounted for in Current;
        Decrement the entry corresponding to it in Current;
        Draw $X \sim \mathrm{Binom}(\texttt{EmptySlots}, 1/(n-i))$;     /* appearances of this
          coupon in leap */
        if the coupon is still deficient then
            Update Next;          /* account for the this coupon in Next */
        EmptySlots $\leftarrow$ EmptySlots $- X$;
        $i$ ++;
    Current $\leftarrow$ Next;
return $T$
```

once, twice, ..., $m-1$ times. This would let us know the array Current at time T_0, and we would have all necessary information to continue running Algorithm 1 from that point. Moreover, if it turned out that, by time T_0, most coupons have arrived m or more times, the runtime would be reduced even further, as we would need to simulate only the remaining part of the process. Note that, on the other hand, T_0 should not be too large. Indeed, if it turned out that all coupons have arrived at least m times by T_0, we would only know that the process ended earlier. Summing up, we would like to find a time T_0 with the two properties (i) most coupons have probably arrived at least m times until T_0, and (ii) there is probably at least one coupon who has arrived less than m times by T_0.

Perhaps somewhat surprisingly, it turns out to be indeed possible to find a T_0 with the desired properties. To this end, let us recall the continuous model of coupon collecting (see [5,6,18,19]). In this model, the coupons arrive at continuous times rather than discrete. Consider the interarrival times of any particular type of coupon. Whereas in the discrete model these times are distributed $\mathrm{Geom}(1/n)$, in the continuous model they are distributed according to the continuous analogue, namely $\mathrm{Exp}(1/n)$. Thus, each coupon type arrives according to a

Poisson process with $\text{Exp}(1/n)$ interarrival times [28, Chapter 7]. The advantage of this model is that the arrival times of distinct coupon types are independent. When we consider all coupon types combined, the arrival times follow again a Poisson process, but now with interarrival times distributed $\text{Exp}(1)$ (being the minimum of independent exponential variables whose sum of parameters is 1).

One readily observes that the global statistics of coupon arrivals is quite similar in the discrete and the continuous models. The interarrival times of each coupon type are distributed according to a geometric distribution or its continuous analogue. In both models, coupons arrive on average once every unit time. Moreover, by the lack-of-memory property of the exponential distribution, each time we get a coupon, there is a probability of $1/n$ for each type to appear, same as in the discrete model. Thus, the order of the arriving coupons in the continuous model has the same distribution as that in the discrete one.

We start by choosing a large time t_0 for the continuous model, such that we may guess that most, but not all, coupons have arrived at least m times until t_0. More formally, we take a small (see below) $\varepsilon > 0$ and look for a t_0 such that $P(T_{n,m} \leq t_0) = \varepsilon$. According to the discussion above, the probability of each coupon to arrive exactly k times until time t_0 is $(t_0/n)^k e^{-t_0/n}/k!$ for every $k \geq 0$. Hence, our requirement from t_0 amounts to

$$\left(1 - e^{-\lambda} - e^{-\lambda}\lambda - e^{-\lambda}\lambda^2/2! - \cdots - e^{-\lambda}\lambda^{m-1}/(m-1)!\right)^n = \varepsilon, \tag{3}$$

where $\lambda = t_0/n$. (Alternatively, one may also use the technique presented in [21].) Note that a proportion of ε of the runs will be over by time t_0. In this case, we have actually collected more coupons than needed, and do not know how many coupons have been collected by the end of the process. (We will explain below what is actually done in this case.) Hence ε should be small. On the other hand, if it is too small, we will usually have many coupons who have not arrived m times by t_0, and therefore the memory and runtime required for the rest of the simulation will be large. In our experiments, we have taken ε between 10^{-4} and 10^{-3} and got a reasonable balance.

Given t_0, the probability of each coupon to arrive k times until t_0, in the continuous model, is $\lambda^k e^{-\lambda}/k!$, $k \geq 0$. Moreover, as the flows of the various coupons are independent, the number of coupons arriving exactly k times is distributed $\text{Binom}(n, e^{-\lambda}\lambda^k/k!)$. However, these numbers are dependent for different k-s. Thus, we draw the number Y_0' of coupons that do not arrive by time t_0 from $\text{Binom}(n, e^{-\lambda})$. The number of those that appear once is $Y_1' \sim \text{Binom}(n - Y_0', e^{-\lambda}\lambda/(1-e^{-\lambda}))$, twice – $Y_2' \sim \text{Binom}(n-Y_0'-Y_1', e^{-\lambda}\lambda^2/(2!(1-e^{-\lambda}-e^{-\lambda}\lambda)))$, and so on. We continue until all coupons are accounted for.

By the process described above, we easily find the number T_0 of coupons that have arrived until (continuous) time t_0. This process is given more formally in Algorithm 2. To complete the process of obtaining m copies of all types of coupons, we continue similarly to Algorithm 1. There are two differences with respect to Algorithm 1: (i) Instead of starting with $T = 0$, we start with $T = T_0$ coupons collected until now. In particular, if by time t_0 all coupons have been collected at least m times, we take T_0 as the number of coupons collected in

Algorithm 2: First Stage of the Process – Improved

```
Input: $n$ – number of coupon types,
       $m$ – number of copies we want to collect,
       $\varepsilon$ – error parameter
Output: Current – array of length $m$, the $i$-th cell contains the number of
        coupons which have obtained exactly $i$ copies,
        $T_0$ – number of coupons collected during this stage
Initiate variables:
Current $\leftarrow \mathbf{0}$;
$T_0 \leftarrow 0$;
copies $\leftarrow 0$;
$\lambda_0 \leftarrow$ (numerical) solution of (3);
$p_1 \leftarrow e^{-\lambda_0}$;
$p_2 \leftarrow 1$;
TreatedCoupons $\leftarrow 0$;
while TreatedCoupons $< n$ do
    Draw $W \sim \text{Binom}(n -$ TreatedCoupons$, p1/p2)$;
    TreatedCoupons $\leftarrow$ TreatedCoupons $+ W$;
    $T_0 \leftarrow T_0 + W \cdot$ copies;
    if copies $< m$ then
        Current[copies] $\leftarrow W$;
    Increment copies;
    $p_2 \leftarrow p_2 - p_1$;
    $p_1 \leftarrow (\lambda_0)^{\texttt{copies}} \cdot e^{-\lambda_0}/$copies!;
return $T_0$ and Current
```

this iteration. This introduces a relative error of at most ε on average in the simulation results. (ii) Instead of starting with the array `Current` $= (n, 0, \ldots, 0)$, we start with `Current` corresponding to the state we are at (discrete) time T_0.

By [27, Theorem 1], the maximum number of copies we obtain for any coupon until time T_0 is $\Theta(\max\{m, \log n\})$. Hence, the runtime of the first part of the algorithm, in which we obtain `Current` for some large time T_0, is $\Theta(\max\{m, \log n\})$. Altogether, the runtime of the whole algorithm is $\Theta(\max\{\log n, m\} + R)$, where R is the number of coupons arriving less than m times in the first T_0 drawings. According to our experiments, R is always very small relative to n, but it would be interesting to estimate it theoretically. The required memory is $\Theta(m)$.

We note in passing that drawing variates from some distributions takes a lot of time [22]. Thus, whenever appropriate, we have used the Poisson approximation of the binomial.

3 Simulation Results

We have tested the runtime of our algorithms versus that of the naive algorithm. Thus, we have three algorithms:

– Algorithm 0 – the naive algorithm which, draws the coupons one by one.
– Algorithm 1.
– Algorithm 2' – this is Algorithm 1, which is preceded by Algorithm 2, as explained in the previous section.

To illustrate the performance of the algorithms, we have run a simulation with several parameter pairs (n, m), as follows:

– $m = 1$ and $n = 10^3, 10^4, 10^5$.
– $m = 6, 8, 10, 12, 14, 20, 30, 40, 50$ and $n = \mathrm{round}(e^{m-1})$.

The point of taking $m = 1$ is to compare the simulation results with Erdős-Rényi's theoretical result (see (2) above). The point of the larger m-s is to illustrate a case in which the simulation may help predicting a pattern. As mentioned above, Erdős-Rényi's result relates to fixed m, and a natural question is how the time T required to collect all coupons behaves when m varies as a function of n (in our case, as $\log n + 1$). For each (n, m) we have performed a number of iterations, and then calculated the sample mean and variance. We also estimated, by the method of moments, the parameters of the Gumbel distribution corresponding to T. The simulation was carried out using Mathematica on a laptop.

In Table 1 we present the results for Algorithms 0, 1, and 2', where the collector wants to obtain at least one copy from each coupon and the number of coupons is $n = 10^3, 10^4, 10^5$. In this case we have preformed 10^3 iterations. In the first column we present the value of n, in the second – the algorithm's runtime, and in the following two columns – the sample mean and variance, respectively. The last four columns provide the values of the parameters μ and β of the Gumbel distribution corresponding to T. We present both the (approximate) theoretical values following from Erdős-Rényi's asymptotic result, namely $\mu = n \log n$ and $\beta = n$, and the estimates $\hat{\mu}$ and $\hat{\beta}$ for these parameters according to the simulation, based on the method of moments. For each n we have three rows of results, one for each algorithm.

Note that here the results obtained by the three algorithms agree with one another and with the theoretical results. As far as runtimes go, Algorithm 1 improves upon Algorithm 0 for the larger values of n by about 50%. Algorithm 2' provides a very major improvement over the first two algorithms.

One may wonder why the runtimes of Algorithm 2' drop when passing from $n = 10^4$ to $n = 10^5$. We have found out that most of the running time of the algorithm is spent on drawing the binomial random variates. The drop in the runtimes is probably connected to the way Mathematica draws such variates. Namely, one needs to know for what values of the parameters Mathematica uses a normal approximation. We guess that, for $n = 10^5$, most drawing of binomial random variates were replaced by Mathematica by drawings of normal variates, while this was not the case for $n = 10^4$. We have not delved deeper into this issue.

In Table 2 we present similar data for $m = 6, 8, 10, 12, 14, 20, 30, 40, 50$, and $n = \mathrm{round}(e^{m-1})$. For each (n, m) we have preformed 10^4 iterations. Algorithm 0

Table 1. Results of simulation with 1000 iterations where $m = 1$ and various n-s.

n	Alg.	Runtime (min.)	Sample mean	Sample variance	$\hat{\mu}$ (sample)	$\hat{\beta}$ (sample)	μ (theory)	β (theory)
10^3	0	0.93	7467	$1.72 \cdot 10^6$	6877	1022		
	1	0.98	7482	$1.62 \cdot 10^6$	6908	993	6908	1000
	2'	0.04	7406	$1.55 \cdot 10^6$	6846	970		
10^4	0	22.32	97868	$1.64 \cdot 10^8$	92110	9976		
	1	10.37	97430	$1.48 \cdot 10^8$	91965	9468	92103	10000
	2'	0.21	97583	$1.70 \cdot 10^8$	91713	10170		
10^5	0	162.77	$1.21 \cdot 10^6$	$1.69 \cdot 10^{10}$	$1.15 \cdot 10^6$	101382		
	1	99.17	$1.21 \cdot 10^6$	$1.65 \cdot 10^{10}$	$1.15 \cdot 10^6$	100022	$1.15 \cdot 10^6$	100000
	2'	0.07	$1.21 \cdot 10^6$	$1.57 \cdot 10^{10}$	$1.15 \cdot 10^6$	97691		

Table 2. Simulation with 10000 iterations; various m-s, $n = \text{round}(e^{m-1})$.

m	Alg.	Runtime (min.)	Sample mean	Sample variance	$\hat{\mu}$ (sample)	$\hat{\beta}$ (sample)	μ "theory"	β "theory"
6	0	2.58	2151	72235	2030	210		
	1	1.26	2154	69676	2035	206	1222	148
	2'	0.36	2149	73813	2027	212		
8	0	28.51	22592	$4.18 \cdot 10^6$	21677	1593		
	1	8.64	22595	$4.15 \cdot 10^6$	21678	1589	13270	1097
	2'	2.61	22576	$4.05 \cdot 10^6$	21671	1568		
10	0	289.79	216679	$2.39 \cdot 10^8$	209718	12060		
	1	60.80	216377	$2.29 \cdot 10^8$	209560	11810	129431	8103
	2'	4.44	216531	$2.30 \cdot 10^8$	209709	11817		
12	0	—	—	—	—	—		
	1	481.85	$1.97 \cdot 10^6$	$1.29 \cdot 10^{10}$	$1.92 \cdot 10^6$	88451	$1.19 \cdot 10^6$	59874
	2'	10.63	$1.97 \cdot 10^6$	$1.28 \cdot 10^{10}$	$1.92 \cdot 10^6$	86192		
14	2'	1.42	$1.73 \cdot 10^7$	$6.63 \cdot 10^{11}$	$1.69 \cdot 10^7$	634648	$1.05 \cdot 10^7$	442413
20	2'	2.39	$1.03 \cdot 10^{10}$	$1.12 \cdot 10^{17}$	$1.01 \cdot 10^{10}$	$2.60 \cdot 10^8$	$6.35 \cdot 10^9$	$1.78 \cdot 10^8$
30	2'	3.69	$3.49 \cdot 10^{14}$	$5.63 \cdot 10^{25}$	$3.46 \cdot 10^{14}$	$5.85 \cdot 10^{12}$	$2.18 \cdot 10^{14}$	$3.93 \cdot 10^{12}$
40	2'	5.37	$1.04 \cdot 10^{19}$	$2.63 \cdot 10^{34}$	$1.03 \cdot 10^{19}$	$1.26 \cdot 10^{17}$	$6.52 \cdot 10^{18}$	$8.66 \cdot 10^{16}$
50	2'	8.90	$2.89 \cdot 10^{23}$	$1.30 \cdot 10^{43}$	$2.87 \cdot 10^{23}$	$2.82 \cdot 10^{21}$	$1.81 \cdot 10^{23}$	$1.91 \cdot 10^{21}$

did not finish running within ten hours for $m = 12$ and $n = \text{round}(e^{11})$, and Algorithm 1 – for $m = 14$ and $n = \text{round}(e^{13})$. Thus, from that point on we have results only for Algorithm 2'.

Note that, differently from Table 1, in Table 2 the values obtained for $\hat{\mu}$ and $\hat{\beta}$ do not agree with those of μ and β. This is consistent with the fact that the theoretical results are valid for fixed m. Hence the quotes for "theory" in the columns corresponding to μ and β. Performing this simulation has been possible only by Algorithm 2', whose running times remain very reasonable even for huge values of n.

Regarding the runtime in Algorithm 2', we note that, as in Table 1, it is not monotonous in n. It drops, when passing from $m = 12$ and $n = \text{round}(e^{11})$ to $m = 14$ and $n = \text{round}(e^{13})$, for the same reason explained above

4 Conclusions and Future Work

The algorithm we have developed lets us estimate the distribution of the time by which we have finished collecting the desired number m of copies of each coupon. Other statistics of interest are not gathered. For example, one may ask how many coupons have appeared "few" times. Thus, Adler, Oren, and Ross [1] were interested, for the case of $m = 1$, in the number of coupons that arrived $2, 3, \ldots, k$ times for some k. Performing the simulation by Algorithm 0, we have immediate access to this data. When using Algorithm 2', we need some adaptations to get such data.

Moreover, it would be interesting to consider the case of unequal probabilities. This adaptation needs to take into consideration various aspects which effect the space needed for the algorithm and its runtime. For example, already the initial probability vector is of size n. Each time we deal with a coupon we must know its probability of being drawn. Therefore, at each leap, we need to keep track of the drawing probability of each deficient coupon as well as the number of times it arrived until now.

References

1. Adler, I., Oren, S., Ross, S.M.: The coupon-collector's problem revisited. J. Appl. Probab. **40**(2), 513–518 (2003)
2. Anderson, J., Goyal, N., Rademacher, L.: Efficient learning of simplices. In: Conference on Learning Theory, PMLR, pp. 1020–1045 (2013)
3. Barak-Pelleg, D., Berend, D.: The Time For Reconstructing the Attack Graph in DDoS Attacks (Submitted)
4. Barak-Pelleg, D., Berend, D., Robinson, T.J., Zimmerman, I.: Algorithms for Reconstructing DDoS Attack Graphs Using Probabilistic Packet Marking (Submitted)
5. Barbour, A., Holst, L.D., Janson, S.: Poisson Approximation. The Clarendon Press, Oxford (1992)
6. Boneh, A., Hofri, M.: The coupon-collector problem revisited - a survey of engineering problems and computational methods. Commun. Stat. Stochast. Mod. **13**(1), 39–66 (1997)
7. Brayton, R.K.: On the asymptotic behavior of the number of trials necessary to complete a set with random selection. J. Math. Anal. Appl. **7**(1), 31–61 (1963)
8. Brown, M., Peköz, E.A., Ross, S.M.: Coupon collecting. Probab. Eng. Inf. Sci. **22**(2), 221–229 (2008)
9. Brown, M., Ross, S.M.: Optimality results for coupon collection. J. Appl. Probab. **53**(3), 930–937 (2016)
10. Brunskill, E., Li, L.: The online coupon-collector problem and its application to lifelong reinforcement learning. arXiv preprint arXiv:1506.03379 (2015)

11. Doumas, A.V., Papanicolaou, V.G.: The siblings of the coupon collector. Theory Probab. Appl. **62**(3), 444–470 (2018)
12. Erdős, P., Rényi, A.: On a classical problem of probability theory. Publ. Math. Inst. Hung. Acad. Sci. Ser. A **6**, 215–220 (1961)
13. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1, 3rd edn. Wiley, New York, London, Sydney (1968)
14. Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discret. Appl. Math. **39**(3), 207–229 (1992)
15. Flatto, L.: The Dixie Cup Problem and FKG inequality. High Frequency **2**(3–4), 1–6 (2019)
16. Foata, D., Zeilberger, D.: The collector's brotherhood problem using the Newman-Shepp symbolic method. Algebra Universalis **49**(4), 387–395 (2003)
17. Gumbel, E.J.: Statistical Theory of Extreme Values and Some Practical Applications. A Series of Lectures. National Bureau of Standards Applied Mathematics Series No. 33, U. S. Government Printing Office, Washington, D. C. (1954), viii+51
18. Holst, L.: On birthday, collectors', occupancy and other classical urn problems. Int. Stat. Rev. **54**, 15–27 (1986)
19. Ilienko, A.: Limit theorems in the extended coupon collector's problem. arXiv preprint arXiv:2002.00650 (2020)
20. Laplace, P.S.: Théorie Analytique des Probabilités, vol. 2, p. 1812, Éditions Jacques Gabay, Paris (1995). (Reprint of the 1820 third edition)
21. Li, A., Chen, Y.: Convergence of coupon collecting process via Wormald's differential equation method. arXiv preprint arXiv:1912.02582 (2019)
22. Loukas, S., Kemp, C.D.: The computer generation of bivariate binomial and negative binomial random variables. Commun. Stat. Simul. Comput. **15**(1), 15–25 (1986)
23. Maunsell, F.G.: A problem in cartophily. Math. Gaz. **22**(251), 328–331 (1938)
24. de Moivre, A.: The Doctrine of Chances. 1756, Republished 1967 by Chelsea, New York (1967)
25. Newman, D.J., Shepp, L.: The double Dixie Cup Problem. Am. Math. Mon. **67**(1), 58–61 (1960)
26. Pinheiro, E.C., Ferrari, S.L.P.: A comparative review of generalizations of the Gumbel extreme value distribution with an application to wind speed data. J. Stat. Comput. Simul. **86**(11), 2241–2261 (2016)
27. Raab, M., Steger, A.: "Balls into Bins" — a simple and tight analysis. In: Luby, M., Rolim, J.D.P., Serna, M. (eds.) RANDOM 1998. LNCS, vol. 1518, pp. 159–170. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49543-6_13
28. Ross, S.M.: Introduction to Probability Models, 10th edn. Academic Press, Oxford (2009)
29. Sairam, A.S., Saurabh, S.: A more accurate completion condition for attack-graph reconstruction in probabilistic packet marking algorithm. In: 2013 National Conference on Communications (NCC), pp. 1–5. IEEE (2013)
30. Sairam, A.S., Saurabh, S.: Increasing accuracy and reliability of IP traceback for DDoS attack using completion condition. Int. J. Netw. Secur. **18**(2), 224–234 (2016)
31. Sasaki, Yu., Li, Y., Sakamoto, H., Sakiyama, K.: Coupon collector's problem for fault analysis against AES—high tolerance for noisy fault injections. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 213–220. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_18

32. Savage, S., Wetherall, D., Karlin, A., Anderson, T.: Practical network support for IP traceback. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 295–306 (2000)
33. von Schelling, H.: Auf Der Spur Des Zufalls. Deutsches Statistisches Zentralblatt **26**, 137–146 (1934)
34. von Schelling, H.: Coupon collecting for unequal probabilities. Am. Math. Mon. **61**, 306–311 (1954)
35. Zhou, M.G., et al.: Experimental quantum advantage with quantum coupon collector. arXiv preprint arXiv:2112.07884 (2021)

On the Undecidability of the Panopticon Detection Problem

Vasiliki Liagkou[1,5], Panagiotis E. Nastou[6,7], Paul Spirakis[2,3], and Yannis C. Stamatiou[1,4](✉)

[1] Computer Technology Institute and Press - "Diophantus", University of Patras Campus, 26504 Patras, Greece
liagkou@cti.gr, stamatiu@ceid.upatras.gr

[2] Department of Computer Science, University of Liverpool, Liverpool, UK
P.Spirakis@liverpool.ac.uk

[3] Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece

[4] Department of Business Administration, University of Patras, 26504 Patras, Greece

[5] Department of Informatics and Telecommunications, University of Ioannina, 47100 Kostakioi, Arta, Greece

[6] Department of Mathematics, Applied Mathematics and Mathematical Modeling Laboratory, University of the Aegean, Samos, Greece
pnastou@aegean.gr

[7] Center for Applied Optimization, University of Florida, Gainesville, USA

Abstract. In this paper we provide a theoretical framework for studying the detectability status of Panopticons based on two theoretical definitions. We show, using *Oracle Turing Machines*, that detecting modern day, ICT-based, Panopticons is an undecidable problem.

Keywords: Formal methods · Security · Privacy · Undecidability · Panopticon · Turing Machine · Oracle computations

1 Introduction

In this paper we, formally, investigate the complexity of detecting *Panopticons* (see the pioneering works of Bentham and Foucault [1,4]), as synonyms of *massive survaillance* in modern societies, based on *Turing Machines*. We provide two different, but not unrealistic, theoretical models of a Panopticon and show that there is no algorithm that can detect, systematically, all Panopticons under these two definitions. In other words, detecting Panopticons, at least the ones that fall under these two plausible definitions, is an undecidable problem, in principle.

The Research of P.E. Nastou has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code:T2EDK-01862). The work of the first, third and fourth coauthors was partially supported by the CyberSec4Europe project, funded by the European Union under the H2020 Programme Grant Agreement No. 830929.

More specifically, the first formal model we examine studies Panopticons whose Panopticon behaviour is manifested through the *execution* of states (actions) that belong to a specific set of states that characterizes Panopticon behaviour. In some sense, since the focal point of this model is the *execution* of states of a particular type, the model captures the *visible behaviour* of the Panopticon, according to the *actions* it performs, and, thus we call this model *behavioural*. The second formal model focuses on the *impact* or *consequences* of the actions of the Panopticon and not the actions themselves. In particular, this model captures an essential characteristic of Panopticons: acquiring, rather, *effortlessly* information through *surveillance* and *eavesdropping*. We model this characteristic using *Oracle Turing Machines* with the oracle having the role of information acquired "for free" based on surveillance (observations) and eavesdropping actions, without requiring computational effort. This model is, in some sense, based on the information that a Panopticon deduces using "free" information and, thus, we call it *deductive*. The focus is on the *semantics* of a Turing Machine, i.e. outcomes of operation, while the first model focuses on the *syntax*, i.e. definition, of a Turing Machine.

2 Definitions and Notation

In this section we briefly state the relevant definitions and notation that will be used in the subsequent sections. We, first, define a simple extension of a Turing Machine, following the notation in [6].

Definition 1 (Turing Machines). *A Turing Machine can be defined as a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where Q is a finite set of normal operation states, Γ is a finite set called the tape alphabet, where Γ contains a special symbol B that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the input alphabet, δ is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ called the transition function, $q_0 \in Q$ is a distinguished state called the start state, $F \subset Q$ is a set of final states.*

With $<M>$ we will denote the *encoding* or *code* of the Turing Machine.

One of the main outcomes of Turing's pioneering work [8] was that there exist problems that Turing machines cannot solve. The first, such, problem was the, so called, *Halting problem*:

The Halting Problem
Input: A string $x = <M, w>$ which is actually the encoding (description) of a Turing machine <M> and its input w.
Output: If the input Turing M machine halts on w, output True. Otherwise, output False.

The language corresponding to the Halting problem is $L_u = \{<M, w> | w \in L(M)\}$. In other words, the language L_u contains all possible *Turing machine-input* pair encodings $<M, w>$ such that w is accepted by M. This is why L_u is also called *universal language*. The language L_u was the first language proved to be non-recursive or undecidable by Turing in his seminal work [8].

In order to discuss Panopticons, we need an important variant of Turing machines, called *oracle* Turing Machines. Such a machine has a special tape on which it can write queries to which they obtain the answer instantaneously in one step, no matter what query it is. This type of Turing Machines was, first, discussed, briefly, by Turing himself in [9] under the name *O-machine*. Post's collaboration with Kleene in [7] resulted to the definition that is used today in computability theory.

Below, we give a formal definition of an Oracle Turing Machine:

Definition 2 (Oracle Turing Machine). *Let A be a language, $A \subseteq \Sigma^*$. A Turing machine with oracle A is a single-tape Turing machine with three special states $q_?, q_y$ and q_n. The special state $q_?$ is used to ask whether a string is in the set A. When the Turing machine enters state $q_?$ it asks: "Is the string of non-blank symbols to the right of the tape head in A?" The answer is provided by having the state of the Turing machine change on the next move to one of the states q_y or q_n. The computation proceeds normally until the next time $q_?$ is reached.*

With respect to notation, we denote by M^A the Turing machine M with oracle A. Also, a set (language) L is recursive with respect to A if $L = L(M^A)$ for some Turing machine M^A that *always* halts while two oracle sets (languages) are called *equivalent* if each of them is recursive in the other (see [6]).

3 The Panopticon Detection Problem and Our Approach

In Cohen's pioneering work (see [2,3]) a natural, *formal*, definition of a virus is provided based on Turing machines. Specifically, Cohen defined a virus to be a program, or Turing machine, that simply copies itself to other programs, or more formally, injects its transition function into other Turing machines' transition functions (see Definition 1) replicating, thus, itself indefinitely. Then, he proves that L_u reduces to the problem of deciding whether a given Turing Machine behaves in this way proving that detecting viruses is an undecidable problem.

Following Cohen's paradigm, we will propose two reasonable definitions of a Panopticon. A Panopticon is a Turing machine that when executed will demonstrate a specific, recognizable, behaviour particular to Panopticons manifested by the *execution* (not simply the existence) of a sequence of actions, e.g. it will publish secret information about an entity, it will download information illegally etc., actions that can be reflected by reaching, during its operation, particular states in a set Q_{pan}.

Definition 3 (Behavioural Panopticons). *A Behavioural Panopticon is an octuple*

$$M = (Q, Q_{pan}, \Sigma, \Gamma, \delta, q_0, B, F)$$

where Q is a finite set of normal operation states, Γ is a finite set called the tape alphabet, where Γ contains a special symbol B that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the input alphabet, δ is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ called the transition function, $q_0 \in Q$ is a distinguished state called the start state, $F \subset Q$ is a set of final states, and $Q_{pan} \subset Q$, $Q_{pan} \cap F = \emptyset$, is a distinguished set of states linked to Panopticon behaviour. We assume that transitions from states in Q_{pan} do not change the Turing Machine's tape contents, i.e. they are purely interactions with the external environment of the Turing Machine and can affect only the environment.

Beyond displayed behaviour, however, Panopticons can be reasonably assumed to also possess *deductive* powers, not directly visible or measurable. In other words, one type of such Panopticons may operate by gathering or computing totally new information, distinct from the information already known to it. We model this behaviour with the language S_1' defined later in this Section. Moreover, another type of Panopticons can take advantage of *easily* acquired, or even *stolen*, freely provided (in some sense) information. In other words, based on information the Panopticon acquires for free, in a sense, it deduces further information, perhaps expending some computational effort this time. We model the characteristic Panopticon action, i.e. *observation* or *surveillance*, using oracle Turing Machines, where the freely acquired information is modeled by the oracle set of the machine. Based on this information, the Turing machine deduces, through its normal computation steps, *further* information about its targets. This behaviour is modelled with the language S_2' defined later in this Section. Below, we describe both types of Panopticons, the ones based on S_1' and the ones based on S_2' since their common characteristicc is the *deduction* of new information from already known information.

Definition 4. *(Deductive Panopticons) A Panopticon is a Turing Machine that either by itself (language S_1') or based on observed or stolen information and, thus, acquired without expending computational effort to deduce or produce it (language S_2'), deduces (perhaps with computational effort) further information about entities.*

In the definition above, the Panopticon operating by itself, i.e. without oracles (language S_1'), is weaker (as we will show in what follows) than the one with oracles (language S_2') since the latter is allowed to obtain free advice or information, in the form of an oracle.

Naturally, many other deductive Panopticon definitions would be reasonable or realistic. Our main motivation behind the ones stated above was a balance of theoretical simplicity and plausibility in order to spark interest on the study on formal properties of Panopticons as well as the difficulty of detecting them algorithmically.

Based on the two Panopticon definitions we gave above, we can define the corresponding Panopticon detection problems. The aim of a Panopticon detection algorithm or Turing machine, is to take as input the encoding of another Turing machine and decide whether it is Panopticon or not based on the formal definition.

The Panopticon Detection Problem 1
Input: A description of a Turing machine (program).
Output: If the input Turing machine behaves like a Panopticon according to Definition 3 output True. Otherwise, output False.

More formally, if by L_b we denote the language consisting of Turing machine encodings $<M>$ which are Panopticons according to Definition 3, then we want to decide L_b, i.e. to design a Turing machine that, given $<M>$, decides whether $<M>$ belongs in L_b or not. Then (we omit the proof due to lack of space) the following can be proved:

Theorem 1. *(Impossibility of detecting behavioural Panopticons) The language L_b is undecidable.*

The Panopticon Detection Problem 2
Input: A description of a Turing machine (program).
Output: If the input Turing machine behaves like a Panopticon according to Definition 4 output True. Otherwise, output False (essentially, this problem asks to decide the languages S_1' and S_2').

Our approach is different for each of the two Panopticon models we propose since they are of a different nature, i.e. syntactic (for the behavioural model) vs. semantic (for the deductive model). For the behavioural model, we provide a simple adaptation of Cohen's pioneering formal model of a *virus* and prove a Panopticon detection impossibility result much like Cohen's result for virus detection. For the deductive model, we follow a completely different approach using Oracle Turing Machines and a technique that can be applied to prove undecidabililty results for this type of machines.

More specifically, in Chapter 8 of [6] a technique from [5] is presented that establishes a hierarchy of undecidable problems for Oracle Turing Machines. In particular, the technique targets the oracle set $S_1 = \{<M> | L(M) = \emptyset\}$, with $<M>$ denoting the encoding of Turing machine M, as we discussed before. Then, the sets $S_{i+1} = \left\{<M> | L(M^{S_i}) = \emptyset\right\}$ can be, recursively, defined and the following can be proved (see [5,6]):

Theorem 2. *The membership problem for TM's without oracles is equivalent to S_1 (i.e. L_u is equivalent to S_1).*

Theorem 3. *The problem of deciding whether $L(M) = \Sigma^*$ is equivalent to S_2.*

Our first contribution is to propose a plausible Panopticon model which incorporates the *information gathering and deduction* element of its behaviour (see Definition 4). More formally, let $N_i = \left\{L_1^i, L_2^i, \ldots, L_k^i\right\}$ be a set of recursively enumerable languages, for some fixed integer $k \geq 1$, such that $\emptyset \notin N_i$ for all i. Also, let $M_1^i, M_2^i, \ldots, M_k^i$ the Turing machines that, correspondingly, accept these languages. These Turing machines and their corresponding languages model the fixed, finitely many, information sets already known to the Panopticon. We, also, say that a set is *disjoint* from a collection of sets if it is disjoint from all the sets in the collection. We will, now, define the oracle set $S'_1 = \{<M>|L(M) \text{ is disjoint from } N_1\}$ ($<M>$ is the encoding of Turing machine M), and, recursively, in analogy with [5,6], the sets $S'_{i+1} = \left\{<M>|L(M^{S'_i}) \text{ is disjoint from } N_{i+1}\right\}$. The sets S'_1 and $S'_2 = \left\{<M>|L(M^{S'_1}) \text{ is disjoint from } N_2\right\}$ in particular, are central to our approach.

Based on this framework, in Sect. 4 we prove two theorems analogous to Theorems 2 and 3 on the undecidability of the problem of detecting a deductive Panopticon. The first one, Theorem 4, is focused on the weaker form of the deductive Panopticons, related to the set S'_1, while the more powerful one, based on oracle computation for "free" information gathering, related to the set S'_2, is handled by Theorem 5. In particular, in Theorem 4 we prove that L_u is equivalent to S'_1 and in Theorem 5 we prove that the problem of whether $L(M) = \Sigma^*$ is equivalent to S'_2.

Before continuing, we should remark that the essential element of the proposed definition of deductive Panopticons is that the oracle consultations model the "effortless", through surveillance, interception or eavesdropping, information gathering by Internet surveillance agencies and organizations. In this context, the sets S'_{i+1} define an infinite *hierarchy* of deductive Panopticons in which a Panopticon whose accepted language belongs in S'_{i+1} operates by consulting a (weaker) lower-level Panopticon whose language belongs in S'_i, with the weakest Panopticons being the ones whose accepted languages belong in S'_1. These last level Panopticons do not have oracle consultations.

4 Deductive Panopticons

In the following two theorems, we prove the undecidability of S'_1 and S'_2. Although their undecidability follows, directly, from *Rice's Theorem* (see [6]), the proofs we give below provide more insightful information as they place S'_2 in a *higher* undecidability level than S'_1.

Theorem 4. *The Halting Problem for Turing machines without oracles, i.e. L_u, is equivalent to S'_1.*

Proof. We first prove that given an oracle for S'_1 we can recognize L_u. We construct $M^{S'_1}$ such that given $\langle M, w\rangle$ constructs a Turing machine M' which operates as follows. It ignores its input and simulates, internally, M on w. M' accepts its input if M accepts w which means that $L(M') = \Sigma^*$ otherwise, i.e. if M does not accept w then M' does not accept its input and $L(M') = \emptyset$. Then, $M^{S'_1}$ asks the oracle whether $<M'> \in S'_1$. If yes, i.e. $L(M') = \emptyset$, then M does not accept w. If no, then $L(M^{'}) = \Sigma^*$ and, thus, M accepts w. We, thus, can recognize L_u.

Now, we show that given an oracle for L_u we can recognize S_1'. We will construct a Turing machine $M^{''}$ such that, given M, it constructs another Turing machine $M^{'}$ that operates as follows. $M^{'}$ ignores its own input and uses a generator of triples (i, j, l), $1 \leq l \leq k+1$, for simulating the lth Turing machine, M_l, with $M_{k+1} = M$, on the ith string lexicographically constructed for j steps. The triples are generated in increasing order of the sum $n = i + j + l$ of their components and for triples of equal component sum, in increasing i, then in increasing j (if the i components are equal), and finally in increasing l (if the i and j components are equal). Each time one of $M_1,\ M_2, \ldots, M_k$ accepts a particular input, this input is recorded on $M^{'}$'s second tape. Each time M_{k+1} accepts an input, it is also recorded on $M^{'}$'s second tape separately from the inputs accepted by $M_1,\ M_2, \ldots, M_k$. Then, $M^{'}$ checks (using the recorded inputs stored on its second tape) whether this M_{k+1} input, or one accepted previously by M_{k+1}, has been accepted by one of $M_1,\ M_2, \ldots, M_k$. If no, the process continues. If yes, $M^{'}$ stops the simulation and accepts its own input. Thus, $<M> \in S_1'$ if $L(M^{'}) = \emptyset$ since this means that $L(M)$ is disjoint from N_1 while $<M> \notin S_1'$ if $L(M^{'}) = \Sigma^*$, i.e. $M^{'}$ accepts all its inputs, ε in particular. Then, $M^{''L_u}$ may query its oracle set L_u for $\left\langle M^{'}, \varepsilon \right\rangle$. If the answer is yes then $M^{''}$ rejects $<M>$ which means that $<M> \in S_1'$, otherwise it accepts $<M>$ i.e. $<M> \notin S_1'$. Thus, S_1' is recognizable. □

Theorem 5. *The problem of deciding whether $L(M) = \Sigma^*$ is equivalent to S_2'.*

Proof. We first show that deciding whether $L(M) = \Sigma^*$ *is recursive in* S_2'. We construct $\hat{M}^{'S_2'}$ that takes as input a Turing machine M and constructs from it$\hat{M}^{S_1'}$, that is a Turing machine with oracle set S_1', that operates in the following way. It enumerates strings x over the alphabet Σ, and for each such string it uses oracle S_1' in order to decide whether M accepts x. This can be accomplished by constructing M' which ignores its input and simulates M on x. If M accepts x then M' accepts its input which means that $L(M') = \Sigma^*$ while $L(M') = \emptyset$ if M does not accept x. Then, $\hat{M}^{S_1'}$ asks the oracle whether $<M'> \in S_1'$. If the answer is yes, which means that M accepts x, then $\hat{M}^{S_1'}$ does not accept its input.

Thus, $\hat{M}^{S_1'}$ accepts its own input if and only if there is a string x *not* accepted by M. Consequently,

$$L(\hat{M}^{S_1'}) = \begin{cases} \emptyset, \text{ if } L(M) = \Sigma^* \\ \quad \Sigma^* \text{ otherwise.} \end{cases}$$

Now $\hat{M}^{'S_2'}$ asks its oracle S_2' whether $<\hat{M}^{S_1'}> \in S_2'$, i.e. whether $L(\hat{M}^{S_1'})$ is disjoint from all sets in N_2. If the answer is yes, then $L(\hat{M}^{S_1'}) = \emptyset$ and, thus, $L(M) = \Sigma^*$. If no, then $L(\hat{M}^{S_1'}) = \Sigma^*$ and, thus, $L(M) \neq \Sigma^*$. Thus, deciding whether $L(M) = \Sigma^*$ is recursive in S_2'. We show that S_2' is recursive in the problem of whether $L(M) = \Sigma^*$. If L_* contains the codes of the Turing machines accepting all their inputs, then we will prove that there exists a Turing machine $\hat{M}^{''L_*}$, i.e. a Turing machine with oracle set L_*, recognizing S_2'.

Given a Turing machine $M^{S'_1}$, we define the notion of a *valid computation* of $M^{S'_1}$ using oracle S'_1 in a way similar to the notion defined in [5,6]. A valid computation is a sequence of Turing Machine step descriptions, called *Instantaneous Descriptions* or *ID*, such that the next one follows from the current one after a computational (*not* oracle query) step, according to the internal operation details (i.e. transition function or program) of the Turing machine. Roughly, an ID describes fully the status of a Turing Machine computation at each time step, containing information such as tape contents, head position, and current state. However, if a query step is taken, i.e. the Turing machine $M^{S'_1}$ enters state $q_?$, and the next state is q_n, this means that $M^{S'_1}$ submitted a query to the oracle S'_1 with respect to whether some given Turing machine, say T, belongs to the set S'_1, receiving the answer no. In other words, the oracle replied that $<T> \notin S'_1$ or, equivalently, $L(T)$ is not disjoint from *all* sets in N_1. As evidence for the correctness of this reply from the oracle, we substitute the query step with a valid computation of the ordinary (i.e. with no oracle) Turing machine T that shows that a *particular* string from a language in N_1 is, also, accepted by T. If, however, after $q_?$ the state q_y follows, no computation is inserted. Intuitively, such a computation would be infinite. By definition, all valid computations conclude in a *halting*, i.e. acceptance state (see [5,6] for details).

We describe the operation of $\hat{M}''^{L_*}$ with $<M^{S'_1}>$ as input. Given $M^{S'_1}$, $\hat{M}''^{L_*}$ constructs M' which accepts all computations of $M^{S'_1}$ which show that they are not a Panopticon. We call these computations *non-Panopticon* computations and they are of two disjoint types: (i) invalid computations, i.e. computations which contain invalid successions of IDs, and ii) unsuccessful computations, i.e. computations which, although not invalid, they demonstrate that $M^{S'_1}$ is *not* a Panopticon.

M' interprets its inputs as computations of $M^{S'_1}$. Given such an input, M' first checks if the string is malformed (i.e. not of correct format) or when one step does not follow from the previous one according to the internals of the Turing machine $M^{S'_1}$, or when the inserted, non-oracle, computation in a $q_?$-q_n step is not valid. In all these cases M' accepts the input string as an invalid computation.

However, there is some difficulty in the $q_?$-q_y cases since, as we stated above, there is no obvious *finite* computation evidence for the correctness or not of the reply. Now the Turing machine M' must decide on its own whether the reply to each $q_?$-q_y query is correct. Let us assume there are $t \geq 1$ such queries in the examined computation (otherwise there are no $q_?$-q_y cases to check). Let, also, w be the input string to the computation of $M^{S'_1}$ that is checked by M' whether it is invalid, so as to accept it.

In particular, the reply q_y to the ith, $1 \leq i \leq t$, query means that the language recognized by T_i is disjoint from all the sets in N_1, i.e. $<T_i> \in S'_1$. Using a *round robin* technique similar to the *triples generation* technique described in the proof of Theorem 4, M' cycles, concurrently

- (Simulation A) over all the t $q_?$-q_y queries in the examined computation of $M^{S'_1}$, trying to locate a string accepted by a queried Turing Machine T_i and one of the Turing Machines $M_1^1, M_2^1, \ldots, M_k^1$ in S'_1.
- (Simulation B) over $M^{S'_1}$ and the Turing Machines $M_1^2, M_2^2, \ldots, M_k^2$ in S'_2, with the same input w, trying to discover whether w, which is accepted by the examined (by $M^{'}$) computation of $M^{S'_1}$, if valid, is, also, accepted by one of the Turing Machines $M_1^2, M_2^2, \ldots, M_k^2$ in S'_2.

As long as none of the above simulations concludes, $M^{'}$ continues the search. If one of them concludes, then $M^{'}$ stops the simulation and accepts its input string (which represents a computation of $M^{S'_1}$) since the computation it represents was either *invalid* (Simulation A concludes) or *unsuccessful* (Simulation B concludes). In other words, the computation was *a non-Panopticon computation.*

Based on the above, $L(M^{'}) = \Sigma^*$ if and only if $<M^{S'_1}> \notin S'_2$. Thus, $\hat{M}^{''L_*}$ can, now, ask its oracle whether $L(M^{'}) = \Sigma^*$ or not, deciding in this way S'_2 and, thus, detecting deductive Panopticons. $\square$

5 Conclusions

In this paper we addressed the problem of detecting Panopticons and their activity based on *Oracle Turing Machines.* Comparing Theorems 1, 4, and 5, Theorem 1 examines the detection of Panopticons based on the execution of *specific* visible or detectable actions, i.e. on a *behavioural level*, such as connecting to a server and sending eavesdropped information or sending an email to the unlawful recipient. Theorems 4 and 5 examine Panopticon detection not based on their visible behaviour but from what languages they may recognize, without having any visible clue of behaviour or actions, only their *descriptions* as Turing machines (i.e. programs or systems). These theorems, that is, examine the detection of Panopticons at a *metabehavioural level.* With respect to the difference between Theorems 4 and 5, we first observe that L_u is recursively enumerable but not recursive while the $\{<M>|L(M) = \Sigma^*\}$ language is *not* recursively enumerable (see, e.g., [6]). Although they are, both, not recursive (i.e. not decidable), their "undecidabilities" are of different levels, with the $\{<M>|L(M) = \Sigma^*\}$ language considered "more difficult" than L_u in restricted types of Turing machines (Panopticons). For example, the L_u language is decidable for Context-free Grammars (i.e. for Turing machines modeling Context-free Grammars) while the $\{<M>|L(M) = \Sigma^*\}$ language is still undecidable. Also, for regular expressions, the problem of deciding L_u is solvable efficiently (i.e. by polynomial time algorithms) while the $\{<M>|L(M) = \Sigma^*\}$ language has been shown, almost certainly, to require exponential time (in the length of the given regular expression) to solve (see, e.g., [6]). Therefore, a similar decidability complexity status is expected from S'_1 (deductive Panopticons without external advice) and S'_2 (deductive Panopticons with external advice in the form of an oracle) since they are equivalent to the languages L_u and $\{<M>|L(M) = \Sigma^*\}$ respectively. That is, when we consider more restricted definitions of Panopticons that render the detection problem decidable, then deciding which Panopticons belong in S'_1 is expected to be easier than deciding which Panopticons belong in S'_2.

In conclusion, we feel that the formal study of the power and limitations of massive surveillance establishments and mechanisms of today's as well as of the future Information Society can be, significantly, benefited from fundamental concepts and deep results of computability and computational complexity theory.

References

1. Bentham, J.: Panopticon or The Inspection House. Written as a Series of Letters in 1787
2. Cohen, F.: Computer viruses. Ph.D. thesis, University of Southern California (1985)
3. Cohen, F.: Computer viruses: theory and experiments. Comput. Secur. **6**(1), 22–35 (1987)
4. Foucault, M.: Discipline and Punish: The Birth of the Prison. Random House, New York (1977)
5. Hartmanis, J., Hopcroft, J.E.: Structure of undecidable problems in automata theory. In: Proceedings of 9th Annual IEEE Symposium on Switching and Automata Theory (SWAT 1968), pp. 327–333 (1968)
6. Hopcroft, J., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley series in Computer Science (1979)
7. Kleene, S.C., Post, E.L.: The upper semi-lattice of degrees of recursive unsolvability. Ann. Math. **59**, 379–407 (1954)
8. A.M. Turing: On Computable Numbers, with an Application to the Entscheidungs problem. Proc. London Math. Soc. **2**, 230–265 (1936–1937)
9. Turing, A.M.: Systems of logic based on ordinals. Proc. London Math. Soc. **45**(Part 3), 161–228 (1939)

Privacy-Preserving Contrastive Explanations with Local Foil Trees

Thijs Veugen[1,2](✉), Bart Kamphorst[1], and Michiel Marcus[1]

[1] TNO, The Hague, The Netherlands
{thijs.veugen,bart.kamphorst,michiel.marcus}@tno.nl
[2] CWI, Amsterdam, The Netherlands
http://www.tno.nl,http://www.cwi.nl

Abstract. We present the first algorithm that combines privacy-preserving technologies and state-of-the-art explainable AI to enable privacy-friendly explanations of black-box AI models. We provide a secure algorithm for contrastive explanations of black-box machine learning models that securely trains and uses local foil trees. Our work shows that the quality of these explanations can be upheld whilst ensuring the privacy of both the training data, and the model itself. An extended version of this paper is found at Cryptology ePrint Archive [16].

Keywords: Explainable AI · Secure multi-party computation · Decision tree · Foil tree

1 Introduction

The field of explainable AI focuses on improving the interpretability of machine learning model behaviour. Popular algorithms are the LIME [12] and SHAP [10] algorithms, which take a data point and its classification according to a trained machine learning model, and output the importance of each feature for that particular classification. The downside is that often a large number of features are used, which makes it hard to interpret. In a successful attempt at reducing the features in the explanation, Van der Waa et al. [15] created an algorithm called *local foil trees* that explains why someone was classified as class A instead of another class B, by providing a set of decisions rules that need to apply for that point to be classified as class B. This provides an increased understanding of the AI system [14], which can for instance be used to infer what can be done to change the classification, and is therefore actionable.

Our work focuses on creating a secure algorithm that provides the same functionality as the local foil tree algorithm in a setting where the black-box machine learning model needs to remain secret to protect the confidentiality of the machine learning model and the training data.

We assume that the black-box machine learning model cannot be revealed, because of commercial reasons, or its known leakage of sensitive training data [8,17,18]. This poses a new challenge for black-box explainable AI, because it is

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 88–98, 2022.
https://doi.org/10.1007/978-3-031-07689-3_7

not trivial to train a decision tree and extract an explanation when some of the inputs to the tree need to remain hidden.

There is a variety of cryptographic techniques that can be used to securely train models. When multiple organisations are involved, common techniques are secret sharing [3] and homomorphic encryption [11]. In this work, we address the aforementioned challenge and provide an algorithm that can produce contrastive explanations when the model is either secret shared, or homomorphically encrypted.

An additional challenge comes from the fact that explainable AI works best when rule-based explanations, as provided through the local foil tree algorithm, are accompanied by an example-based explanation, such as a data point that is similar to the user, but is classified as class B instead of A [14]. The use of a data point (having class B) from the sensitive training data would violate privacy in the worst way possible. As we will discuss in Sect. 3, we address this challenge using synthetic data.

In summary, we present a secure solution to explain AI, consisting of:

– A cryptographic protocol to securely train a binary decision tree when the target variable is hidden;
– An algorithm to securely generate synthetic data based on numeric sensitive data;
– A cryptographic protocol to extract a rule-based explanation from a hidden foil tree, and construct an example data point for it.

In the remainder of this introduction, we discuss related work. In the sections following after, we explain the local foil tree algorithm [15] and present a secure solution. Thereafter, we discuss the complexity of the proposed solution and share experimental results. Finally, we provide closing remarks in the conclusion.

1.1 Related Work

Our solution is based on the local foil tree algorithm by Van der Waa et al. [15], for which we design a privacy-preserving solution based on MPC. There is related work in the area of securely training decision trees, but these results are never applied to challenges in explainable AI. As we will elaborate on further in Sect. 3, we have a special setting where the feature values of the synthetic data to train the decision tree are not encrypted, but the classifications of these data points *are* encrypted. As far as we know, no training algorithm for such a setting has been proposed yet.

We mention the work of de Hoogh et al. [4], who present a secure variant of the well-known ID3 algorithm (with discrete variables). Their training data points remain hidden, whereas in our case that is not necessary. Furthermore, as the number of children of an ID3 decision node reflects the number of categories of the chosen feature, the tree decision is not completely hidden. Furthermore, Abspoel et al. [1] have implemented C4.5 and CART in the MP-SPDZ framework, but their solution is less efficient, because they work with encrypted feature

Table 1. Notation as used throughout the document. Some symbols are seen in the context of a certain point (node) within the decision tree, in which case they can be sub- or superscripted with l or r to denote the same variable in the left or right child node that originates from the current node.

A	Fact (class); classification of the user as indicated by the black-box
B	Foil (class); target class for contrastive explanation to the user
$\mathcal{B}$	Decision tree or, equivalently, foil tree
G_s	Gini index for split $s \in \{1, \ldots, \varsigma\}$
$\tilde{G}_s = N_s/D_s$	Adjusted Gini index for split $s \in \{1, \ldots, \varsigma\}$
k_A, k_B	Index of classes A and B, respectively
K	Number of classes
n	Number of available synthetic data points in a particular node
N	Number of synthetic data points $\lvert\mathcal{X}\rvert$
P	Number of features per data point
ς	Number of splits $\lvert\mathcal{S}\rvert$
$S_s = (p_s, t_s)$	Feature index $p_s \in \{1, \ldots, m\}$ and threshold t_s of split S_s, $1 \leq s \leq \varsigma$
$\boldsymbol{x}_i, \boldsymbol{x}_U$	Vector $(x_{i,1}, \ldots, x_{i,P})$ of feature values of synthetic data point i. With subscript U, it refers to the data point of the user
$\mathcal{X}$	Set of all synthetic data points $\boldsymbol{x}_i$, $i = 1, \ldots, N$
$\boldsymbol{y}_i$	Indicator vector $(y_{i,1}, \ldots, y_{i,K})$ of the class of data point i as indicated by the black-box
ξ_i	Bit that indicates whether data point i is available (1) or unavailable (0) in the current node

values. In a similar approach, Adams et al. [7] scale the continuous features to a small domain to avoid the costly secure operations, at the expense of a potential drop in accuracy.

The work of [9] presents a new class of machine learning models that are interpretable and privacy-friendly with respect to the training data. Our work does not introduce new models, but provides an algorithm to improve the interpretation of existing complex models that have been securely trained on sensitive data.

1.2 Notation

Due to the inherent complexity of both explainable AI and cryptographic protocols, we require many symbols in our presentation. These symbols are all introduced in the body of this paper; however, for the reader's convenience we also summarize the most important symbols in Table 1.

Sets are displayed in curly font, e.g. $\mathcal{X}$, and vectors in bold font, e.g. $\boldsymbol{x}_U$. The vector $\boldsymbol{e}_j$ represents the j-th elementary vector of appropriate, context-dependent length. The notation $(x \geq y)$ is used to denote the Boolean result of the comparison $x \geq y$. Any symbol between square brackets $[\cdot]$ represents a secret-shared version of that symbol. Finally, a reference to line y of Protocol x is formulated as line $x.y$.

Protocol 1. Foil-tree based explanation

```
Input: Data point $x_U$ that is classified as class $A$; foil class $B$
Output: Explanation why $x_U$ was not classified as the foil class
1: Obtain a classification for the user                              ▷ cf. Sect. 3.1
2: Prepare the synthetic data points for the foil tree               ▷ 3.2
3: Classify all synthetic data points through the black-box          ▷ 3.3
4: Train a decision tree                                             ▷ 3.4
5: Locate fact leaf (leaf node of $x_U$)                             ▷ 3.5
6: Determine the foil leaf (leaf node of class $B$ closest to fact leaf)   ▷ 3.6
7: Determine the decision node at which the root-leaf paths of the fact and foil leaf
   split                                                             ▷ 3.7
8: Construct the explanation (and provide an example data point).    ▷ 3.7
```

2 Explainable AI with Local Foil Trees

In this section we present the local foil tree method of Van der Waa et al. [15] and discuss the challenges when the black-box classifier needs to remain secret. We assume that this classifier returns a secret-shared classification.

If a user-supplied data point $\boldsymbol{x}_U$ is classified as some class A, our goal is to construct an explanation why $\boldsymbol{x}_U$ was not classified as another class B. The explanation will contain decision rules of the form that a certain feature of $\boldsymbol{x}_U$ is less (or greater) than a certain threshold value. An overview of the different steps is formalized in Protocol 1. Note that we deviate from Van der Waa et al. by providing an example data point in the final step. In each step of the protocol, we also refer to the section of our work where we present secure protocols for that step.

To train the decision tree, we adapt the CART algorithm [2] to work with secret-shared labels. We use the CART algorithm, because it generates binary trees. Other algorithms, such as ID3, generate non-binary trees, so their structure can reveal which feature is used in a node. The result of the adapted CART procedure is a binary decision tree whose decision rules and leaf classifications are secret-shared. As a consequence, we need a secure protocol for determining the position of a foil data point, and all nodes that are relevant for the explanation.

3 Secure Solution

In this section we describe the secure version of the local foil tree algorithm. In the rest of this work, we will refer to *training data* when we talk about the data used to train the black-box machine learning model and to *synthetic data* when we refer to the synthetically generated data that we use to train the foil tree.

The secure protocol generates N synthetic data points $\boldsymbol{x}_i$, $i = 1, \ldots, N$, with P features that each can be categorical, or continuous. To increase the efficiency of the secure solution, we make use of one-hot encoding to represent categorical values. We assume that the class $k \in \{1, \ldots, K\}$ of data point $\boldsymbol{x}_i$ is represented by a secret binary indicator vector $[\boldsymbol{y}_i] = ([y_{i,1}], \ldots, [y_{i,K}])$, such that $y_{i,k} = 1$, if data point $\boldsymbol{x}_i$ is classified as class k by the black-box, and $y_{i,k} = 0$, otherwise.

During the decision tree training, we maintain an indicator vector $\boldsymbol{\xi}$ of length N, such that $\xi_i = 1$, if and only if, the i-th synthetic data point is still present in this branch.

3.1 Classify User Data

We assume that the user is allowed to learn the black-box classification of her own data point $\boldsymbol{x}_U$, so this step is trivial. Without loss of generality, we assume that the user received classification A.

3.2 Generating Synthetic Data

In order to support example-based explanations in a privacy-preserving way, we generate synthetic data based on the sensitive training data. We use a simple algorithm that only requires the secure computation of the mean and standard deviation of the training data for each feature. To increase privacy guarantees, differential privacy [6] could be applied to the secure computation of the means and standard deviations.

We sample the synthetic data values from an interval around the values of data point x_U. For example, w.l.o.g., if feature f_i of x_U has value v_i, then we sample from the interval $[v_i - d_i, v_i + d_i]$ for some relatively small value d_i.

Constrained to this interval, we use the (normalized) normal distribution defined by the mean and standard deviation of the training data. This method ensures privacy, and provides synthetic data that are realistic and similar to x_U. We refer to the full paper [16] for more details.

3.3 Classify Synthetic Data

All synthetic training data points $\boldsymbol{x}_i$ can now be classified securely by the model owner(s). This results in secret-shared classification vectors $[\boldsymbol{y}_i]$. The secure computation depends on the model, and is beyond our scope.

3.4 Training a Decision Tree

In this section, we explain the secure CART algorithm that we use to train a secure decision tree, which is described in Protocol 2. We start with an empty tree, and all training data points are marked as available. First, the stopping criterion uses the number of elements of the most common class (line 2.8), and the total number of elements in the availability vector (line 2.7). The stopping criterion from line 2.10 is securely computed by

$$1 - (1 - [(n \leq \tau \cdot N)] \cdot (1 - [(n = n_{k^*})])$$

This stopping criterion is revealed afterwards to determine whether the algorithm should continue, or not.

If the stopping condition is equal to one, a leaf node with the secret-shared indicator vector of the most common class is generated. In order to facilitate the efficient extraction of a foil data point as mentioned at the start of Sect. 3, we also store the availability vector $\boldsymbol{\xi}$ in this leaf node. How this indicator vector is used to securely generate a foil data point is discussed in Sect. 3.8.

If the stopping criterion is not met, a decision node is created by computing the best split (lines 2.13–19) using the adjusted Gini indices of each split in $\mathcal{S}$. Normally, the best split is the split with the lowest Gini Index. However, for efficiency, we translate the problem of minimizing the Gini Index into a problem of maximizing the Adjusted Gini Index. The Adjusted Gini Index $\tilde{G}_s$ of a split S_s is given by

$$\tilde{G}_s = \frac{n^r \sum_{k=1}^{K} (n_k^l)^2 + n^l \sum_{k=1}^{K} (n_k^r)^2}{n^l \cdot n^r} =: \frac{N_s}{D_s}. \tag{1}$$

Here, n^l is the number of available data points in the left set that is induced by split S_s and n_k^l denotes the number of available data points in the left node with class k. The symbols n^r and n_k^r are defined analogously for the right set. We refer to our full paper [16] for more details. We note that we do not compute $\tilde{G}_s$ as one secure fixed-point number, but securely compute the numerator N_s and denominator D_s separately, and store both for efficiency reasons. Protocol 3 shows how the adjusted Gini index can be computed securely.

After determining the optimal split, an availability vector is constructed for each child based on this split in lines 2.21–22. For each synthetic data point, we check that it is present in the node, and whether it meets the splitting criterion. The entry-wise difference with $[\xi]$ then gives the availability vector for the right child. The CART algorithm is called recursively with the new availability vectors to generate the children of the decision node.

In protocol 2, the `max` subroutine securely computes the maximum value in a list using secure comparisons, and the `find` subroutine finds the secret-shared location of the maximum computed by `max` in the list that was input to `max`. The functions `max` and `find` are already implemented in MPyC. However, since we always use the two in conjunction, we implemented a slight variation. This variation and some comments on the convergence of this secure CART algorithm can be found in our full paper [16].

3.5 Locate the Fact Leaf

Once the decision tree has been constructed, we need to find the leaf that contains the fact class of $\boldsymbol{x}_U$. As the fact leaf will be revealed, the path from the root to the fact leaf will be revealed as well. Therefore, we can traverse the decision tree from the root downwards and reveal each node decision. First, the feature value that is relevant for the current decision node is securely determined through $[x_{U,p_{s*}}] = \sum_{p=1}^{P} [e_{p_{s^*},p}] \cdot [x_{U,p}]$. Second, the secure comparison $[(x_{U,p_{s^*}} \leq t_{s^*})]$ is performed and revealed. The result directly indicates the next decision node that needs to be evaluated. This process is repeated until a leaf is encountered: the fact leaf.

Protocol 2. `cart`

Secure CART training of a binary decision tree.

```
Input: Training set $\mathcal{X}$, split set $\mathcal{S}$, convergence parameter $\tau \in [0,1]$, secret-shared binary availability vector $[\boldsymbol{\xi}]$
Output: Decision tree $\mathcal{B}$
1: $\mathcal{B} \leftarrow \emptyset$
2: $N \leftarrow |\mathcal{X}|$
3: while $\mathcal{B}$ is not fully constructed do
4:     for $k = 1, \dots, K$ do
5:         $[n_k] \leftarrow \sum_{i=1}^{N} [y_{i,k}] \cdot [\xi_i]$        ▷ nr available data points per class
6:     end for
7:     $[n] \leftarrow \sum_{k=1}^{K} [n_k]$        ▷ nr available data points
8:     $[n_{k^*}] \leftarrow \mathtt{max}(([n_1], \dots, [n_K]))$
9:     $[\boldsymbol{e}_{k^*}] \leftarrow \mathtt{find}([n_{k^*}], ([n_1], \dots, [n_K]))$        ▷ indicates most common class
10:    if $[(n \leq \tau \cdot N)]$ or $[(n = n_{k^*})]$ then        ▷ branch fully constructed
11:        Extend $\mathcal{B}$ with leaf node with class indicator $[\boldsymbol{e}_{k^*}]$
12:    else        ▷ branch splits
13:        for $s = 1, \dots, \varsigma$ do
14:            $[G_s] \leftarrow \mathtt{adjusted\_gini}(S_s)$
15:        end for
16:        $[G_{s^*}] \leftarrow \mathtt{max}([\boldsymbol{G}])$
17:        $[\boldsymbol{e}_{k^*}] \leftarrow \mathtt{find}([G_{s^*}], [\boldsymbol{G}])$        ▷ indicates best split
18:        $[p_{s^*}] \leftarrow \sum_{s=1}^{\varsigma} [e_{s^*,s}] \cdot p_s$        ▷ feature of optimal split
19:        $[t_{s^*}] \leftarrow \sum_{s=1}^{\varsigma} [e_{s^*,s}] \cdot t_s$        ▷ threshold of optimal split
20:        $b \leftarrow$ decision node that corresponds with split $([p_{s^*}], [t_{s^*}])$
21:        $\boldsymbol{\xi}^l \leftarrow \mathtt{left\_child\_availability}(\mathcal{X}, [\boldsymbol{xi}], [p_*], [t_{s^*}])$
22:        $[\boldsymbol{\xi}^r] \leftarrow [\boldsymbol{\xi}] - [\boldsymbol{\xi}^l]$
23:        Extend $b$ to the left with result of $\mathtt{cart}(\mathcal{X}, \mathcal{S}, \tau, [\boldsymbol{\xi}^l])$
24:        Extend $b$ to the right with the result of $\mathtt{cart}(\mathcal{X}, \mathcal{S}, \tau, [\boldsymbol{\xi}^r])$
25:        Extend $\mathcal{B}$ with $b$
26:    end if
27: end while
28: Return $\mathcal{B}$
```

3.6 Locate the Foil Leaf

Since we know the fact leaf and the structure of the decision tree, we can create an ordered list of all tree leaves, starting with the closest leaf and ending with the farthermost leaf. We can traverse this list and find the first leaf that is classified as class B, without revealing the classes, but only whether they equal B or not, i.e. by revealing the Boolean $[(e_{k^*,k_B} = 1)]$ for every next leaf. This does not require any extra computations, as these vectors have already been computed and stored during the training algorithm. We use the number of steps between nodes within the decision tree as our distance metric, but as Van der Waa et al. [15] note, there are more advanced options.

Protocol 3. `adjusted_gini`
Compute the adjusted Gini index of a split.

```
    Input: Synthetic data set $\mathcal{X}$, vector of available transactions $\boldsymbol{\xi}$, split
    $(p_s, t_s) = S_s \in \mathcal{S}$
    Output: Encrypted numerator and denominator of adjusted Gini index
    $[\tilde{G}_s] = [N_s]/[D_s]$
 1: for i=1,...,N do
 2:     $\delta_i \leftarrow (x_{i,p_s} \leq t_s)$        ▷ 1 if data point meets split criterion, else 0
 3: end for
 4: $[n] \leftarrow \sum_{i=1}^{N}[\xi_i], \quad [n^l] \leftarrow \sum_{i=1}^{N} \delta_i \cdot [\xi_i], \quad [n^r] \leftarrow [n] - [n^l]$
 5: $[n_k] \leftarrow \sum_{i=1}^{N}[y_{i,k}] \cdot [\xi_i], \quad [n_k^l] \leftarrow \sum_{i=1}^{N} \delta_i \cdot [y_{i,k}] \cdot [\xi_i], \quad [n_k^r] \leftarrow [n_k] - [n_k^l]$
 6: Return $[N_s] \leftarrow [n^r]\sum_{k=1}^{K}([n_k^l])^2 + [n^l]\sum_{k=1}^{K}([n_k^r])^2$ and $[D_s] \leftarrow [n^l] \cdot [n^r]$
```

3.7 Construct the Explanation

As the structure of the decision tree is known, we can identify the lowest common node between the fact and foil leaf without secure computations. The relevant nodes for the explanation lay on the path between the lowest common node and the foil leaf, as they explain what changes need to happen for x_U to end up in the foil leaf. The feature and threshold in the nodes on this path are revealed *only* to the user, as it could possibly leak information about the values we are trying to protect. These pairs of features and thresholds can be interpreted as rules, e.g., $(f_i, t_i) \implies f_i < t_i$ if it is a left child, and $f_i \geq t_i$ otherwise. For each rule, the user determines whether it applies to $\boldsymbol{x}_U$. For instance, if a rule states that $x_{U,i} \geq 3$, and $\boldsymbol{x}_U$ already satisfies this rule, then it is not relevant for the explanation. After this filter is applied, the remaining rules are combined where applicable. For example, if one rule requires $x_{U,i} \geq 3$ and another rule requires $x_{U,i} \geq 4$, we take the strictest rule, which in this case is $x_{U,i} \geq 4$.

3.8 Retrieving a Foil Data Point

We now show how to securely compute a synthetic data point that is classified as class B and ends up in the foil leaf node, which we call a foil data point.[1] We can retrieve the binary availability vector $\boldsymbol{\xi}^{foil}$ of the foil leaf, as this was stored while training the foil tree. A protocol for retrieving a foil data point is presented in Protocol 4, which returns the first foil data point of the synthetic data set.

It is important that the foil data point is only revealed to the user, and not to the computing parties, since the foil data point can leak information on the classifications of the synthetic data points according to the secret-shared model, which are the values we are trying to protect. In practice this means that all computing parties send their shares of the feature values in vector $\boldsymbol{s}$ to the user, who can then combine them to reconstruct the secret values.

[1] Note that it is possible for samples in a foil leaf to have a classification different from B, so care needs to be taken in determining the foil sample.

Protocol 4. `retrieve_foil`
Retrieve foil data point

```
    Input: Availability vector $[\boldsymbol{\xi}]$ of the foil leaf, class index $k_B$
    Output: Foil data point $\boldsymbol{s}$
1: $[\varepsilon] \leftarrow [0]$                ▷ flips to [1] when a foil data point is found
2: for $i = 1, \ldots, n$ do
3:     $[\delta_i] \leftarrow (1 - [\varepsilon]) \cdot [\xi_i] \cdot [y_{i,k_B}]$
4:     $[\varepsilon] \leftarrow [\varepsilon] + [\delta_i]$
5: end for
6: for $p = 1, \ldots, P$ do
7:     $[s_p] \leftarrow \sum_{i=1}^{N} [\delta_i] \cdot [x_{i,p}]$
8: end for
9: Reveal $\boldsymbol{s}$ to the user
```

4 Security

We use the MPyC platform [13], which is known to be passively secure. The computing parties jointly and securely train the foil tree and produce an explanation, which is revealed to the user. The machine learning model is out of scope, we simply assume that the computing parties can securely obtain secret-shared classifications of the synthetic data, without any party learning the classifications.

During the protocol, the computing parties will learn the data point x_U of the user, its class A, and the foil class B, together with the average and variance of each feature in the training data, which are used to generate synthetic data set $\mathcal{X}$. Furthermore, the (binary) structure of the decision tree, including the fact leaf, foil leaf, and therefore also the lowest common node, will be revealed. Other than this, no training data or model information will be known to the computing parties.

The explanation, consisting of a feature index and threshold for each node on the path from lowest common node to the foil leaf, and the foil data point s, is revealed only to the user.

5 Experiments

We implemented our secure foil tree algorithm in the MPyC framework [13]. In our experiments, we ran MPyC with three parties, and used secure fixed point numbers with a 64-bit integer part and 32-bit fractional part. For the secret-shared black-box model, we secret-shared a neural network with three hidden layers of size 10 each. We used the iris data set [5] as our training data for the neural network (using integer encoding for the target variable) and generated three synthetic data sets based on the iris data set of sizes 50, 100 and 150 respectively.

Table 2 shows the results of our performance tests. We report the timing in seconds of our secure foil tree training algorithm under ‘Tree Training’, for the

Table 2. Performance results (timing in seconds) of our algorithms in MPyC.

N	Tree training			Explanation			Data Point			Accuracy
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	
50	20.396	19.594	21.158	0.033	0.027	0.041	0.157	0.112	0.219	0.96
100	94.455	93.133	95.234	0.061	0.058	0.062	0.277	0.269	0.361	0.89
150	130.575	129.681	131.327	0.050	0.038	0.052	0.404	0.387	0.425	0.91

explanation construction under ‘Explanation’, and for the extraction of the data point under ‘Data Point’. For each of these, we report the average timing, the minimum and the maximum that we observed. The column ‘Accuracy’ denotes the accuracy of the foil tree with respect to the neural network. We do not provide any performance results on the training algorithm or classification algorithm of the secret-shared black-box model, since our solution is model-agnostic.

We see that the accuracy does not necessarily increase when we use more samples. A synthetic data set size of 50 seems to suffice for the iris data set, and shows performance numbers of less than half a minute for the entire algorithm.

6 Conclusion

We presented the first cryptographic protocol that is able to explain black-box AI models that are trained by sensitive data, in a privacy-preserving way. The explanation is constructed by means of local foil trees. After generating synthetic data close to a fact data point, a binary tree is securely computed to find the so-called fact and foil leaves. Using both fact and foil leaf, an explanation of the AI model is constructed that explains to the user why they were classified as the fact class, and not as the foil class. We additionally provide a synthetic data point from the foil leaf to strengthen the explanation.

Our solution hides the classification model and its training data, in order to provide explanations towards users without leaking commercially or privacy sensitive data. We implemented our solution with MPyC on the iris data set with different sizes of synthetic data sets. With 50 samples, we achieved an accuracy of 0.96 within half a minute.

Acknowledgements. The research of this paper has been done within the FATE project, which is funded by the TNO Appl.AI program (internal AI program). We additionally thank Jasper van der Waa for his helpful comments and suggestions.

References

1. Abspoel, M., Escudero, D., Volgushev, N.: Secure training of decision trees with continuous attributes. Priv. Enhanc. Technol. **2021**(1), 167–187 (2021)
2. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. Wadsworth (1984)

3. Cramer, R., Damgård, I., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press (2015)
4. de Hoogh, S., Schoenmakers, B., Chen, P., op den Akker, H.: Practical secure decision tree learning in a teletreatment application. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 179–194. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_12
5. Dua, D., Graff, C.: UCI machine learning repository (2017)
6. Dwork, Cynthia: Differential privacy. In: Bugliesi, Michele, Preneel, Bart, Sassone, Vladimiro, Wegener, Ingo (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_1
7. Adams, S., et al.: Privacy-preserving training of tree ensembles over continuous data, CoRR abs/2106.02769 (2021)
8. Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: an end-to-end case study of personalized warfarin dosing. In:Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014. USENIX Association, pp. 17–32 (2014)
9. Harder, F., Bauer, M., Park, M.: Interpretable and differentially private predictions. In: The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI. AAAI Press, pp. 4083–4090 (2020)
10. Lundberg, S.M., Lee, S.-I: A unified approach to interpreting model predictions. In: Annual Conference on Neural Information Processing Systems. Advances in Neural Information Processing Systems, vol. 30, pp. 4765–4774 (2017)
11. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
12. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016. ACM, pp. 1135–1144 (2016)
13. Schoenmakers, B.: MPyC - Secure Multiparty Computation in Python. https://github.com/lschoe/mpyc
14. van der Waa, J., Nieuwburg, E., Cremers, A.H.M., Neerincx, M.A.: Evaluating XAI: a comparison of rule-based and example-based explanations. Artif. Intell. **291**, 103404 (2021)
15. van der Waa, J., Robeer, M., van Diggelen, J., Brinkhuis, M., Neerincx, M.: Contrastive explanations with local foil trees, CoRR abs/1806.07470 (2018)
16. Veugen, T., Kamphorst, B., Marcus, M.: Privacy-preserving contrastive explanations with local foil trees. IACR Cryptology ePrint Archive, no. 360, pp. 1–20 (2022)
17. Yang, Z., Zhang, J., Chang, E.C., Liang, Z.: Neural network inversion in adversarial setting via background knowledge alignment. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security CCS. ACM, pp. 225–240, November 2019
18. Zhang, Y., Jia, R., Pei, H., Wang, W., Li, B., Song, D.: The secret revealer: generative model-inversion attacks against deep neural networks. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, pp. 250–258, June 2020

Timing Leakage Analysis of Non-constant-time NTT Implementations with Harvey Butterflies

Nir Drucker(✉) and Tomer Pelleg

IBM Research - Haifa, Haifa, Israel
drucker.nir@gmail.com

Abstract. Harvey butterflies and their variants are core primitives in many optimized number-theoretic transform (NTT) implementations, such as those used by the HElib and SEAL homomorphic encryption libraries. However, these butterflies are not constant-time algorithms and may leak secret data when incorrectly implemented. Luckily for SEAL and HElib, the compilers optimize the code to run in constant-time.

We claim that relying on the compiler is risky and demonstrate how a simple code modification, naïve compiler misuse, or even a malicious attacker that injects just a single compiler flag can cause leakage. This leakage can reduce the hardness of the ring learning with errors (R-LWE) instances used by these libraries, for example, from 2^{128} to 2^{104}.

Keywords: NTT · Harvey's butterflies · Constant-time code · Compiler optimizations · Ring-LWE · Side-channel attacks

1 Introduction

Constant-time implementations are today considered a requirement for cryptographic libraries that provide production-level code. Commonly, an implementation is considered "constant-time" if code branches and memory access patterns are independent of secret information. Efficiently achieving this property is not always easy, and the literature is full of examples that exploit optimized code that does not run in constant-time, for example, [12,14,22].

The number-theoretic transform (NTT) algorithm is used by many cryptographic implementations to achieve fast polynomial multiplications. Some examples include post-quantum schemes such as Kyber [24], NTTRU [21], and Dilithium [11], or homomorphic encryption (HE) libraries such as SEAL [18], HELib [15], Palisade [27], and HEAAN [7]. Despite the performance benefits provided by NTT, it is still the bottleneck in many of these implementations, which makes it an ideal target for optimization. The list of works that deal with NTT optimizations in hardware and software is large. In this work, we analyze the constant-time property in the optimizations of [16,19] that are also used by SEAL, HElib, and other software [3,16,17] and hardware [10,23] implementations. Specifically, we consider Harvey butterflies [16] that involve branches for performing fast modular reduction and therefore are not considered constant-time algorithms. Nevertheless, some constant-time implementations for them exist, such as the implementation of [19] [Sect. 5] and the vectorized implementations of [3].

SEAL uses the following C macro to perform the modular reduction branch of Harvey's butterfly:

```c
#define SEAL_COND_SELECT(cond,if_true,if_false) (cond ? if_true:if_false
    )
```

and states that "This is a temporary solution that generates constant-time code with all compilers on all platforms." We tested this claim on Linux for all the supported compilers "Clang++ (≥5.0) or GNU G++ (≥6.0)" and observed that the Assembly code generated by the compiler indeed always used a conditional move (`CMOV`) instruction.

The same ternary pattern used in `SEAL_COND_SELECT` is used in NTL [26], an optimized mathematical library that HElib uses for its NTT implementation. However, unlike SEAL, NTL also includes a branch-less implementation that is controlled by the flag `NTL_AVOID_BRANCHING`. The reason for including the flag seems to be related to performance and not to security. This is indicated in the code comment:

> "On some modern machines, this is usually faster and NTL uses this non-branching strategy. However, on other machines (modern x86's are an example of this), conditional move instructions can be used in place of branching, and this code can be faster than the non-branching code. **NTL's performance-tuning script will figure out the best way to do this.**"

or in the comment

> "With this option, branches are replaced at several key points with equivalent code using shifts and masks. It may speed things up on machines with deep pipelines and high branch penalties."

As a final example, we consider the Palisade code [27], which does not seem to implement Harvey's butterflies but still uses a simple if-else code.

Despite requests (e.g., [5]) to include a built-in directive-API in GCC that will force compilers to use a conditional move, it does not yet exist. Consequently, SEAL's assumption could be wrong in new compilers and OSs. Without continuous integration tests to test this assumption, secret information can be leaked. In fact, this may already happen today. The SEAL function `multiply_plain_normal` performs multiplication of plaintext by HE ciphertext, with cases where the plaintext is a secret as indicated therein: "Optimizations for constant/monomial multiplication can lead to the presence of a timing side-channel in use-cases where the plaintext data should also be kept private". This function uses the macro `SEAL_COND_SELECT`, but its translation to Assembly involves branches, which contradicts the original comment.

A side-channel analysis of the NTT algorithm in the context of ring learning with errors (R-LWE) was presented, for example, by [22]. Specifically, this attack relies on a Hamming-weight leakage model, where the data for the model was collected from real traces using electromagnetic (EM) measurements. In contrast, our attack targets high-end CPUs (e.g., x86-64), where collecting EM data is rarely possible. Instead, we rely only on timing differences and specifically the binary knowledge of whether a branch was taken or not. This knowledge can be collected for example when the code is called from within Intel®SGX®by using the SGX-step framework [25]. Another difference between our work and [22] is that our attack focuses on NTT implementations with Harvey butterflies, as in the case for SEAL and HElib code. The work of [22] assumed the following modular reduction operation $a \pmod{q} = a - q\left\lfloor \frac{a}{q} \right\rfloor$, which allowed them to collect leakage information from the variable-time `DIV` instruction on Cortex-M4F. This knowledge allowed them to report a full key recovery attack on NTT. In contrast, we are restricted to a smaller leakage and therefore report only partial key extraction. Still, this partial key extraction can lead to a reduction in the hardness of the R-LWE instances and should be taken into account by security researchers who evaluate potential risks for using a certain implementation.

Our Contribution. This work identifies places in the code of SEAL, HElib and other HE libraries that depend on the compiler for generating a constant-time code. We show why this assumption is risky and analyze the security loss caused by the potential leakage in the key generation code of SEAL, which uses NTT with Harvey butterflies. Our analysis shows that if the generated Assembly is not a constant-time code, we can extract more than 9% of the secret key, which reduces the hardness of the R-LWE instance by more than 10%. For example, from 2^{128} to 2^{104} security estimation.

Organization. The document is organized as follows. Section 2 provides some background and describes our notation. We describe the risk of depending on the compiler in Sect. 3. Section 4 analyzes the leaked data in the case of a sparse secret and reports our results. We conclude in Sect. 6.

2 Background and Notation

Let $\mathbb{F}_q$ be a finite field of characteristic q, where q is prime. The residue classes of $\mathbb{F}_q$ are represented as elements from $Z \cap [0, q)$ and are ordered according to their integer values. The elements in the polynomial quotient ring $\mathcal{R}_q = \mathbb{F}_q[X]/(X^N + 1)$ are polynomials of a degree at most $N-1$ with residue class coefficients in $\mathbb{F}_q$ represented as integers, where $q \equiv 1 \pmod{2N}$ and N is a power of two. We may refer to a polynomial $a = \sum a_i x^i$ by its coefficients i.e., $a = (a_0, \ldots, a_{N-1})$. For a specific platform, we denote its word-size with β. For example, for typical CPUs $\beta = 2^{32}$ or $\beta = 2^{64}$. We use $\wedge$ to denote the binary logical-and operator and $u \Rightarrow v$ to denote that a boolean statement u implies another boolean statement v.

2.1 Distribution of an HE Secret Key

Modern HE schemes rely on the R-LWE assumption [20]. The plaintext, key, and error domains are the polynomials ring $\mathcal{R}_q$, where the keys and errors are randomly derived from the χ_{key}, χ_{err} distributions, respectively. Let $R_q^u \subset \mathcal{R}_q$ be the set of all polynomials from $\mathcal{R}_q$ with coefficients in $\{0, \pm 1\}^N$, then the commonly used options for χ_{key} are the

1. uniform distribution over R_q^u
2. uniform distribution over R_q^u with Hamming weight h, for $h > 0$
3. distribution over R_q^u, where each coefficient has a probability $\frac{\alpha}{2}, \frac{\alpha}{2}, 1-\alpha$ of being $+1, -1, 0$, respectively

The χ_{err} distribution is commonly a Gaussian distribution. We target the residue number system (RNS) variant of CKKS [6] and its key generation method, where a secret key s is sampled from χ_{key}. The original CKKS variant [6] sets χ_{key} according to Option 2 with $h = 64$. In contrast, SEAL implements Option 1 and HElib implements Option 3 with $\alpha = 0.5$. In SEAL, the function `generate_sk` calls the function `sample_poly_ternary` and in HElib, the function `SecKey::GenSecKey` calls `sampleSmallBounded`. We demonstrate the potential leakage on SEAL and set χ_{key} according to Option 1, which simplifies the attack computations. We conjecture that similar exploits can be generated for the other distributions. After generating the secret key s, the SEAL `generate_sk` function invokes the NTT algorithm on s. This behavior is common to other libraries as well.

2.2 NTT

The NTT algorithm is a variant of the fast Fourier transform (FFT) algorithm over $\mathcal{R}_q$. It receives a polynomial $a = (a_0, \ldots, a_{N-1}) \in \mathcal{R}_q$ and a fixed N's primitive root of unity ω as inputs; it outputs $\tilde{a} = (\tilde{a}_0, \ldots, \tilde{a}_{N-1}) \in \mathcal{R}_q$, where $\tilde{a}_i = \sum_{j=0}^{N-1} a_j \omega^{ij}$. The inverse function $a = InvNTT_\omega(\tilde{a})$ is given by $a_i = \frac{1}{N} \sum_{j=0}^{N-1} \tilde{a}_j \omega^{-ij}$. The powers of ω are called twiddles.

Appendix A presents one variant of the NTT and inverse-NTT (InvNTT) algorithms as specified in [19]. These algorithms are implemented in different libraries such as SEAL. Variants of these algorithms are available in NTL and inherently in HElib. The main bottleneck of these algorithms is the Cooley-Tukey (CT) [8] and Gentleman-Sande (GS) [13] butterflies, respectively. These are implemented in SEAL using Harvey butterflies [16], which we present in Algorithm 1. For brevity, we use $\texttt{ShoupModMul}(t, \omega, \omega', q) = \omega t - q \left\lfloor \frac{\omega' t}{\beta} \right\rfloor$ to denote Shoup's multiplication [16], which performs a lazy reduction and leaves the output in the range $[0, 2q-1]$. Here, β is a fixed global parameter, and $\omega' = \left\lfloor \frac{\omega \beta}{q} \right\rfloor$ is a precomputed value.

Algorithm 1. Harvey's Butterflies [16]

```
   Global parameters: A word-size $\beta$, a modulus $q < \frac{\beta}{4}$; $\omega \in \mathbb{F}_q$; $\omega' = \left\lfloor \frac{\omega\beta}{q} \right\rfloor < \beta$
   Input: $0 \leq x, y < 4q$
   Output: $x = x + \omega y$, $y = x - \omega y \pmod{4q}$
1: procedure HarveyFwdButterfly($x, y, \omega, \omega', q, \beta$)
2:     if $x \geq 2q$ then $x = x - 2q$
3:     $t$ =ShoupModMul($y, \omega, \omega', q$)
4:     return $(x + t,\ x - t + 2q)$
   Input: $0 \leq x, y < 2q$
   Output: $x = x + y$, $y = \omega(x - y) \pmod{2q}$
1: procedure HarveyInvButterfly($x, y, \omega, \omega', q, \beta$)
2:     $x' = x + y$
3:     if $x' \geq 2q$ then $x' = x' - 2q$
4:     $t = x - y + 2q$
5:     $y'$ =ShoupModMul($t, \omega, \omega', q$)
6:     return $x', y'$
```

3 Compiler Optimizations

The compiler's decision to use a conditional move or branch depends on the penalty that the compiler believes a program hits when it executes the specific branch. Controlling this penalty and observing the changes in the compiler output is possible using the compiler's target flag `-mbranch_cost=`x, where x is in $\{0, \ldots, 5\}$. Indeed, when setting $x \leq 2$ and compiling the SEAL code, the output Assembly does not include conditional moves.

The reasons that led the compiler to add conditional moves to begin with, are performance-related and not security-related. Thus, other flags may affect its decision. For example, the following GCC optimization flags `-O0`, `-fno-if-conversion`, `-fno-if-conversion2`, `-fno-tree-loop-if-convert`, and `-fno-tree-loop-if-convert-stores` may turn off this optimization.

Someone may inadvertently compile a cryptographic library using the above flags, for example in debug mode. It can also be the case that an adversary intentionally injects these compilation flags in order to convert the code be non constant-time. Putting these two options aside, it is still interesting to explore some examples where the compiler simply does not know how to compute the branch penalty and thus even when using the default optimization mode, it uses branches because this is its default behavior. Consider the next example:

```c
uint64_t foo(uint32_t *a) {
    uint64_t i = 0;
    while (i < 5) {
        i = (i >= a[i]) ? i+2 : i;
    }

    return i;
}
```

Here, the compiler does not know the content and size of a and thus the generated Assembly (using Clang-10) is

```
 1     0:    31 c0                    xor    %eax,%eax
 2     2:    eb 12                    jmp    16 <foo+0x16>
 3     4:    66 2e 0f 1f 84 00 00     nopw   %cs:0x0(%rax,%rax,1)
 4     b:    00 00 00
 5     e:    66 90                    xchg   %ax,%ax
 6    10:    48 83 f8 05              cmp    $0x5,%rax
 7    14:    73 0e                    jae    24 <foo+0x24>
 8    16:    8b 0c 87                 mov    (%rdi,%rax,4),%ecx
 9    19:    48 39 c8                 cmp    %rcx,%rax
10    1c:    72 f2                    jb     10 <foo+0x10>
11    1e:    48 83 c0 02              add    $0x2,%rax
12    22:    eb ec                    jmp    10 <foo+0x10>
13    24:    c3                       retq
```

which involves the `jb` branch on line 10, instead of a conditional move. Another, perhaps simpler example, is the function

```
uint64_t foo2(uint32_t a, uint32_t b) {
    return (a > b ? a : b);
}
```

which when compiled with GCC-9 uses conditional moves (`cmovb`).

```
  44:    39 fe                    cmp    %edi,%esi
  46:    0f 42 f7                 cmovb  %edi,%esi
  49:    89 f0                    mov    %esi,%eax
  4b:    c3                       retq
```

However, once we perform a simple modification to it

```
uint64_t foo3(uint32_t a, uint32_t b) {
    if (b < 100000) return b;
    return (a > b ? a : (a < 2*b ? b : a));
}
```

the output assembler involves branches.

```
  54:    39 f7                    cmp    %esi,%edi
  56:    77 10                    ja     68 <foo3+0x18>
  58:    8d 04 36                 lea    (%rsi,%rsi,1),%eax
  5b:    39 f8                    cmp    %edi,%eax
  5d:    76 09                    jbe    68 <foo3+0x18>
  5f:    89 f0                    mov    %esi,%eax
  61:    c3                       retq
  62:    66 0f 1f 44 00 00        nopw   0x0(%rax,%rax,1)
  68:    89 f8                    mov    %edi,%eax
  6a:    c3                       retq
```

Note that the logic in `foo3` is the same logic as in `foo2`. The false condition of the first ternary operator should be considered only when $a \leq b$; in which case it also follows that $a \leq 2b$. In this example, for the logic to stay the same, it is important that the $2 \cdot b$ operation does not result in an integer overflow. To accommodate this, we added the first `if` statement to bound b. Another example involves lookup tables where the key is a secret, as in [9].

The above examples show that even a small code modification may break SEAL's assumption as already happened for the function `multiply_plain_normal`. Therefore, even if a code is currently compiled with a conditional move, it is important to understand the consequences of a compilation mistake that leads to leaking secret information.

4 Exploiting NTT over Secret Keys

We already saw how a simple code modification or a simple malicious injection of a compilation flag can result in a variable time implementation that may leak secret information. In this section, we provide an example that exploits an NTT implementation that may leak information. Specifically, we target the NTT transformation that SEAL applies to every secret key. We focus on this scenario for two reasons. First, the distribution of the key is over polynomials with small coefficients in $\{-1, 0, 1\}$, hereafter denoted small polynomials. Second, this example uses the forward NTT of [19], which involves a smaller number of twiddles in its first few iterations. This is in contrast to analyzing the inverse NTT of [19], which is more complex since it involves a different twiddle for every butterfly.

Although we focus on the SEAL code, the methods we apply here should work with minor modifications on other key distributions, butterflies, or NTT implementations. For example, we could have performed the same analysis to extract data on the encryption error polynomial, which is derived from a somewhat wider distribution, i.e., its coefficients are sampled from a discrete Gaussian distribution with mean zero and standard deviation $\sigma \approx 3.2$.

In this demonstration, we analyze, measure, and accumulate the leakage knowledge after every NTT iteration. We use analytical methods for the first and second iterations and empirical methods for the other iterations, in which the number of options to analyze grows exponentially. Figure 1 shows the first few iterations of the NTT and InvNTT algorithms over polynomials of degree $N = 16$. The yellow ovals (x) demonstrate values that go through the branch $x > 2q \ ? \ x \ : \ x - 2q$ from which we attempt to extract information.

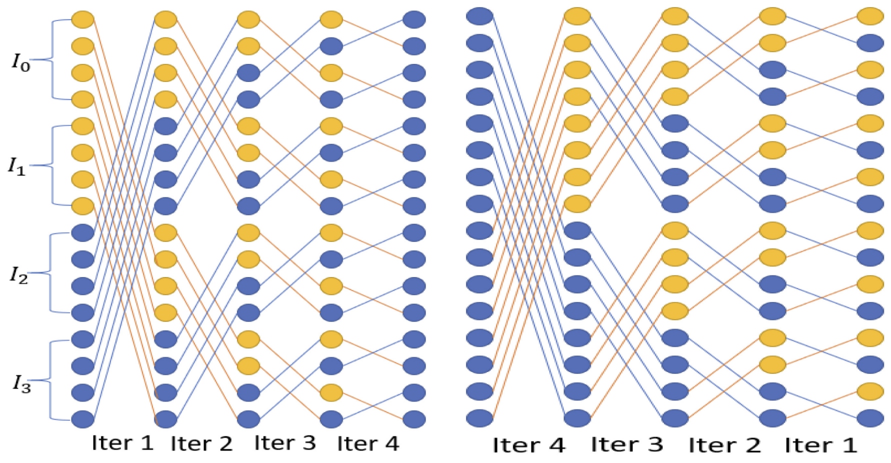
Observation 1. *For a small polynomial input, no information is leaked from the branches of the first NTT iteration.*

Proof. The NTT inputs are always in $\{0, 1, q-1\} < 2q$ thus, the Harvey butterfly branch is never taken, and the branch leakage is independent of the input. $\square$

Second Iteration. For the second iteration, we first define the variable

$$s = ShoupModMul(\mathbf{q - 1}, \omega, \omega', q)$$

and observe some of its properties in Lemmas 1, 2. Note that $s \neq 0$ and $s \neq q$ by definition.



(a) NTT (b) InvNTT

Fig. 1. The first few iterations of the NTT (panel a) and invNTT (panel b) algorithms over polynomials of degree $N = 16$. Yellow ovals (x) demonstrate values that will go through the branch $x > 2q$ `?` x `:` $x - 2q$ in a subsequent iteration.

Lemma 1. *For a given set of parameters β, ω, q of the NTT algorithm (Algorithm 3) if $\omega\beta \pmod q < \frac{(q-\omega)}{(q-1)}\beta$ then $s < q$ otherwise $q \leq s < 2q$*

Proof. First, we compute

$$\begin{aligned}\frac{(q-1)\omega'}{\beta} &= \frac{(q-1)\left\lfloor\frac{\omega\beta}{q}\right\rfloor}{\beta} = \frac{q\left\lfloor\frac{\omega\beta}{q}\right\rfloor}{\beta} - \frac{\left\lfloor\frac{\omega\beta}{q}\right\rfloor}{\beta}\\ &= \frac{\omega\beta - (\omega\beta \pmod q)}{\beta} - \frac{\omega\beta - (\omega\beta \pmod q)}{\beta q}\\ &= \omega - \left\lceil\frac{\omega\beta + (q-1)(\omega\beta \pmod q)}{\beta q}\right\rceil. \end{aligned} \tag{1}$$

Let $\eta = \frac{\omega\beta + (q-1)(\omega\beta \pmod q)}{\beta q}$, where for $0 \leq \eta < 1$ we have

$$\omega\beta \pmod q < \frac{(\beta q - \beta\omega)}{(q-1)} = \frac{(q-\omega)}{(q-1)}\beta$$

as in our assumption. We obtain the first claim by

$$\begin{aligned} s = \texttt{ShoupModMul}(q-1, \omega, \omega', q) &= \omega(q-1) - q\left\lfloor\frac{\omega'(q-1)}{\beta}\right\rfloor\\ &\underset{\texttt{Eq (1)},\ 0\leq\eta\leq1}{=} \omega(q-1) - q(\omega-1) = q - \omega < q \end{aligned}$$

When $1 < \eta < 2$ we have $\left\lfloor\frac{(q-1)\omega'}{\beta}\right\rfloor = \omega - 2$ and $q \leq s = 2q - w < 2q$. Note that $\eta < 2$; otherwise $s > 2q$, which contradicts Harvey's proof [16][Theorem 1]. $\square$

Lemma 2. *For a positive integer γ, when $q < \frac{\beta}{\gamma}$ and $\omega < \frac{\gamma-1}{\gamma}q + \frac{1}{\gamma}$, it follows that $s < q$.*

Proof. Lemma 1 states that $s < q$ when

$$\omega\beta \pmod q < \frac{(q-\omega)}{(q-1)}\beta$$

but $\omega\beta \pmod q < q < \frac{\beta}{\gamma}$, which implies that $s < q$ at least for

$$\frac{\beta}{\gamma} < \frac{(q-\omega)}{(q-1)}\beta$$

which after rearrangement becomes

$$\omega < \frac{(\gamma-1)}{\gamma}q + \frac{1}{\gamma}$$

$\square$

Example 1. For HarveyFwdButterfly, $q < \frac{\beta}{4}$. Thus, $\omega < 0.75q + 0.25 \Rightarrow s < q$.

Example 2. For $\beta = 2^{64}$ and $q < 2^{32}$, it holds that $\omega < \frac{2^{32}-1}{2^{32}}q + \frac{1}{2^{32}} \Rightarrow s < q$.

The last example emphasizes that in SEAL, where the implementation uses 64-bit scalars ($\beta = 2^{64}$), for primes that are orders of magnitude smaller than 2^{64} we will rarely encounter the $q < s < 2q$ case.

Theorem 1. *Given the NTT parameters $q, \omega, \omega', \beta$ and $N = 2^m$, and $k \leq 2^{m-1}$, for the input variable $s =$`ShoupModMul`$(q-1, \omega, \omega', q)$ and the branches br_k and $br_{k+2^{m-1}}$, Algorithm 2 returns a list of the possible coefficients at position k and $k + 2^{m-1}$ after applying `HarveyFwdButterfly` on a small polynomial.*

Proof. Let $x_k, x_{k+2^{m-1}}$ be the inputs to HarveyFwdButterfly and denote by $x'_k, x'_{k+2^{m-1}}$ its output. Table 1 shows the possible outputs of the `HarveyFwdButterfly` after the first NTT iteration.

Table 1. Possible x'_k (left) and $x'_{k+2^{m-1}}$ (right) values.

	x_k		
$x_{k+2^{m-1}}$	0	1	$q-1$
0	0	1	$q-1$
1	ω	$\omega+1$	$\omega+q-1$
$q-1$	s	$s+1$	$s+q-1$

	x_k		
$x_{k+2^{m-1}}$	0	1	$q-1$
0	$2q$	$2q+1$	$2q+q-1$
1	$2q-\omega$	$2q-\omega+1$	$2q-\omega+q-1$
$q-1$	$2q-s$	$2q-s+1$	$2q-s+q-1$

Algorithm 2. Second iteration exploit

```
    Input: $s$ (see text), branches parameters $br_k$, $br_{k+2^{m-1}}$.
    Output: Possible input pairs for the coefficients at position $k$ and $k+2^{m-1}$
 1: procedure SecondIterExploit($s$, $br_k$, $br_{k+2^{m-1}}$):
 2:     $out = \{0,1,q-1\} \times \{0,1,q-1\}$
 3:     if $br_k$ is taken: then
 4:         return $\{(q-1,q-1)\}$
 5:     if $br_{k+2^{m-1}}$ is taken then
 6:         $out = \{(0,0),(1,0),(q-1,0),(q-1,1)\}$
 7:         if $s < q$ then
 8:             $out = out \cup \{(q-1,q-1)\}$
 9:     if $br_{k+2^{m-1}}$ is not-taken then
10:         $out = \{(0,1),(0,q-1),(1,1),(1,q-1)\}$
11:         if $s > q$ and $br_k$ is unknown then
12:             $out = out \cup \{(q-1,q-1)\}$
13:     return $out$
```

First, recall that $\omega \notin \{1, q-1\}$ as there are no primitive roots of unity for primes bigger than 3 and that $0 < \omega, s < 2q$. Then, by looking at the table, we see that the only case for b_k to be taken ($x'_k > 2q$, Step 3) is when $(x_k, x_{k+2^{m-1}}) = (q-1, q-1)$ and $q < s < 2q$. We continue the analysis assuming that br_k is not taken or is unknown. Here,

$$\begin{aligned}
(x_k, x_{k+2^{m-1}}) \in \{(0,0),(1,0),(q-1,0),(q-1,1)\} &\Rightarrow (x'_{k+2^{m-1}} \geq 2q)\\
&\Rightarrow b_{k+2^{m-1}} \texttt{ is taken}\\
(x_k, x_{k+2^{m-1}}) \in \{(0,1),(0,q-1),(1,1),(1,q-1)\} &\Rightarrow (x'_{k+2^{m-1}} < 2q)\\
&\Rightarrow b_{k+2^{m-1}} \texttt{ is not taken}\\
((x_k, x_{k+2^{m-1}}) = (q-1,q-1)) \wedge (s < q)&\\
&\Rightarrow b_{k+2^{m-1}} \texttt{ is taken}\\
((x_k, x_{k+2^{m-1}}) = (q-1,q-1)) \wedge (q < s < 2q)&\\
&\Rightarrow (b_{k+2^{m-1}} \texttt{ is not taken}) \wedge (b_k \texttt{ is unknown})
\end{aligned}$$

The correctness of the algorithm follows. □

4.1 Extracted Leakage After the Second Iteration

Algorithm 2 provides us with a way to reduce the number of options for the inputs x_k and $x_{k+2^{m-1}}$ of the NTT algorithm. Theorem 2 summarizes the expected leakage and Fig. 2 illustrates it. The theorem uses the following four intervals $I_0 = [0, \frac{N}{4})$, $I_1 = [\frac{N}{4}, \frac{N}{2})$, $I_2 = [\frac{N}{2}, \frac{3N}{4})$, $I_3 = [\frac{3N}{4}, N)$.

Theorem 2. *Let the input to Algorithm 3 be a small polynomial and let* $0 \leq \rho \leq 1$ *be the percentage of the extracted branches (distributed uniformly).*

1. *The probability that a coefficient (x_k, $k \in I_0 \cup I_2$) has only one option when $q < s < 2q$ is $P_1 = \frac{\rho}{9}$.*
2. *The probability for a coefficient (x_k) to have only two options is*

$$P_2 = \begin{cases} \frac{4\rho}{9} & (s < q) \wedge (k \in I_0 \cup I_2) \\ \frac{4\rho^2}{9} & (q \leq s < 2q) \wedge (k \in I_0) \\ \frac{3\rho^2+5\rho}{9} & (q \leq s < 2q) \wedge (k \in I_2) \end{cases}$$

Proof. We define the following events:

$$E_1 := (br_{k+2^{m-1}} \texttt{ is not taken }) \mid (0 < s < q)$$
$$E_2 := (br_k \texttt{ is taken}) \mid (q < s < 2q)$$
$$E_3 := (br_k \texttt{ is not taken}) \wedge (br_{k+2^{m-1}} \texttt{ is not taken }) \mid (q < s < 2q)$$
$$E_4 := (br_k \texttt{ is not taken}) \wedge (br_{k+2^{m-1}} \texttt{ is taken }) \mid (q < s < 2q)$$
$$E_5 := (br_k \texttt{ is unknown}) \wedge (br_{k+2^{m-1}} \texttt{ is not taken }) \mid (q < s < 2q)$$

where Algorithm 2 outputs

$$\begin{aligned} \{(0,1),(0,q-1),(1,1),(1,q-1)\} &\texttt{ for } E_1, E_3 \\ &\Rightarrow x_k \in \{0,1\} \texttt{ and} \\ &x_{k+2^{m-1}} \in \{1, q-1\} \\ \{(0,0),(1,0),(q-1,0),(q-1,1)\} &\texttt{ for } E_4 \\ &\Rightarrow x_{k+2^{m-1}} \in \{1, q-1\} \\ \{(0,1),(0,q-1),(1,1),(1,q-1),(q-1,q-1)\} &\texttt{ for } E_5 \\ &\Rightarrow x_{k+2^{m-1}} \in \{1, q-1\} \\ \{(q-1,q-1)\} &\texttt{ for } E_2 \\ &\Rightarrow x_k = x_{k+2^{m-1}} = q-1 \end{aligned}$$

The coefficients of small polynomials are uniformly distributed and independent of ρ. Thus, the above events will happen with probability

$$Pr(E_1) = \frac{4\rho}{9} \quad Pr(E_2) = \frac{\rho}{9} \quad Pr(E_3) = Pr(E_4) = \frac{4\rho^2}{9} \quad Pr(E_5) = \frac{5\rho(1-\rho)}{9}.$$

At the second iteration we only perform branches over inputs at positions $I_0 \cup I_2$ (see illustration in Fig. 1), which limits the leakage to the coefficients x_k, $k \in I_0 \cup I_2$ of the original input polynomial to the NTT algorithm. When $q < s < 2q$, the probability that a coefficient (x_k, $k \in I_0 \cup I_2$) has only one option is $P_1 = Pr(E_2) = \frac{\rho}{9}$.

The probability of a coefficient having two options is

$$P_2 = \begin{cases} Pr(E_1) = \frac{4\rho}{9} & (s < q) \wedge (k \in I_0 \cup I_2) \\ Pr(E_3) = \frac{4\rho^2}{9} & (q \leq s < 2q) \wedge (k \in I_0) \\ (Pr(E_3) + Pr(E_4) + Pr(E_5)) = \frac{3\rho^2+5\rho}{9} & (q \leq s < 2q) \wedge (k \in I_2) \end{cases}$$

□

Let X_1, X_2 denote the number of coefficients with only one or two options, respectively, for a given ρ. In addition, for a polynomial of degree N and $\rho = 1$, we define $Y_1 = \mathbb{E}(X_1)/N$, $Y_2 = \mathbb{E}(X_2)/N$ to be the portion of coefficients we identified to have a reduced number of options (one or two, respectively).

Corollary 1. *For an input of a small polynomial of degree N, and a fixed ρ,*

$$\mathbb{E}(X_1) = \begin{cases} 0 & s < q \\ \frac{\rho}{18}N & q < s < 2q \end{cases}$$

$$\mathbb{E}(X_2) = \begin{cases} \frac{2\rho}{9}N & s < q \\ \left(\frac{1}{4} \cdot \frac{4\rho^2}{9} + \frac{1}{4} \cdot \frac{3\rho^2+5\rho}{9}\right) N = \frac{7\rho^2+5\rho}{36}N & q < s < 2q \end{cases}$$

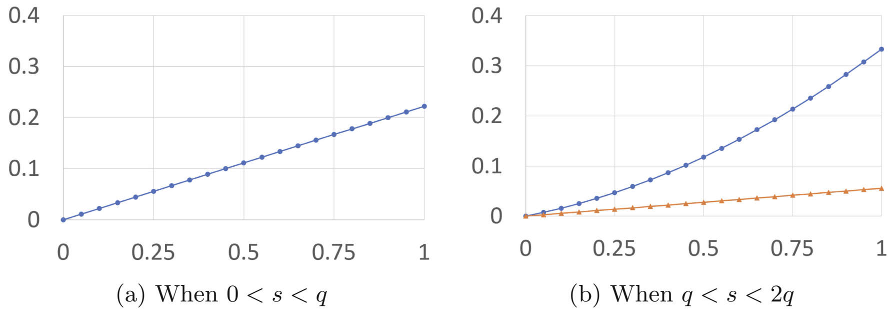


Fig. 2. Second iteration leakage for a given branch extraction probability ρ. The orange triangles show the portion of coefficients (out of N) that are completely identified (Y_1). The blue dots show the portion of coefficients (out of N) that now have only two options out of three (Y_2). (Color figure online)

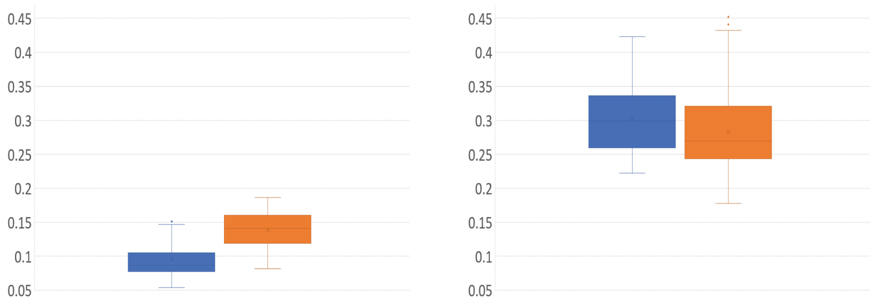
Third Iteration. The input to the third iteration is the output of the second iteration, which we can view as the output of a radix-4 NTT, where every branch depends on a coefficients-quartet of the original polynomial x_k, $x_{k+2^{m-2}}$, $x_{k+2 \cdot 2^{m-2}}$, $x_{k+3 \cdot 2^{m-2}}$ for $k \in I_0$. For every such quartet we may know two branch results from the second iteration. In addition, at the third iteration, for half the quartets $0 \leq k < 2^{m-3}$, we add the knowledge of all four branches, while for the other half, we learn nothing.

Using an analytical approach, as we did for analyzing the leakage at the second iteration, is more complex because the number of cases increases exponentially. For example, in the second iteration, we only had two cases where we needed to consider whether or not $s < q$.

To assess the complexity of analyzing the third iteration, we performed an exploratory empirical experiment, which revealed more than 80 cases that need to be analyzed. We note that, unlike post-quantum schemes where a prime is usually fixed by design, many HE implementations generate the required prime numbers on-the-fly and according to the users' needs. For these reasons, we decided to use an empirical approach to demonstrate the potential data leakage from the third iteration. We stress that this can be fine-tuned when the set of primes is predefined (fixed).

Our experiment involves a program, which for a given modulo N and a prime q, computes the minimal $2N$ primitive root of unity ω. It then executes the first three iterations of the NTT, brute-forcing over all possible quartet inputs. The program generates a branching table as we did for the second iteration (Table 2) and outputs the answers to the following questions:

1. What is the probability of this branch combination occurring, assuming uniform distribution of the original coefficients?
2. How many coefficients does it reduce to one or two options (Y_1, Y_2)?



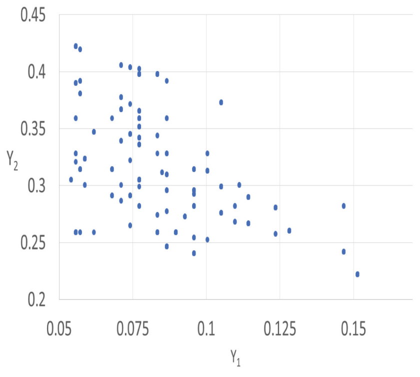
(a) Y_1 (b) Y_2

Fig. 3. Third and fourth iterations leakage extraction (Y_1, Y_2). Blue and orange box-plots show the distribution results when using information up to the third and fourth iteration, respectively. (Color figure online)

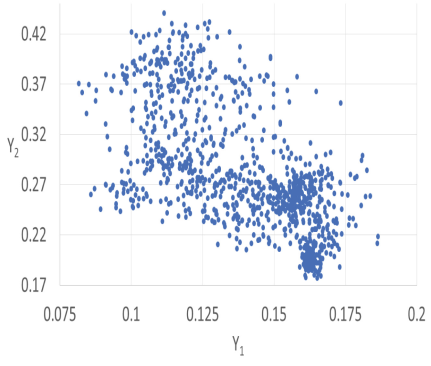
We repeated the above process for the fourth iteration, where every branch affects eight inputs at a time.

For our experiments we generated $1,000$ NTT-friendly primes using the code from Appendix B, and fixed $N = 2^{15}$. Figure 3 shows the portion of coefficients we identified as having a reduced number of options (one or two). As these numbers depend on the values of q and N, we use box-plots to demonstrate the resulting distributions. The left and right box-plots show the distributions after the third and fourth iterations, respectively. As expected, at the fourth iteration, Y_1 increases on average while Y_2 decreases. This phenomenon is also demonstrated in Fig. 4, where we see a negative correlation between Y_1 and Y_2.

Moving into Deeper Iterations. It is possible to extend the search to deeper iterations. However, the number of options that our program checks grows super-exponentially. In the ith iteration, it considers 2^{i-1} coefficients per one of the



(a) Third Iteration



(b) Fourth Iteration

Fig. 4. Correlation between Y1 and Y2, using the information from the third and fourth iterations. The number of dots indicates the number of cases to consider for the experimented primes.

two tables, with three options per coefficient. Thus, the total number of options to consider is $3^{2^{i-1}}$. For example, for $i = 5$ and $i = 6$ the number of options per table to consider is $\sim 2^{25}$ and $\sim 2^{50}$, respectively.

4.2 Hardness of RLWE Instances After the Leakage

Modern HE libraries follow the HE standard [1] when considering security parameters. The standard, in turn, uses the *LWE estimator* [2] to compute the relevant values. We followed the standard and re-evaluated the security guarantees provided when using the same values of N and $\log_2 q$. Specifically, we used the command

```
n = 2048; q = 2^54; alpha = 8/q; m = 2*n
estimate_lwe(n, alpha, q, secret_distribution=(-1,1),
             reduction_cost_model=BKZ.sieve, m=m)
```

from [2], which is designed for SEAL. To assess the security impact, we only considered the coefficients that we completely extracted by replacing n with $n - nY_1$ in the script above. Table 2 summarizes the results. For the second iteration, we used $Y_1 = \frac{\rho}{18}$ from Theorem 2. For the third and forth iterations, we first evaluated the entropy per experiment using the equation

$$\frac{(Y_2 + \log_2 3 \cdot (1 - Y_1 - Y_2))}{\log_2 3}$$

and then chose the pairs (Y_1, Y_2) that yielded the minimal and maximal entropy per iteration. Specifically, we used $(0.0555556, 0.259259)$ and

$(0.146605, 0.2824075)$ for the third iteration, and $(0.08916325, 0.2458845)$ and $(0.17332525, 0.35162325)$ for the fourth iteration. It can be observed that when including the leakage from the fourth iteration, the security estimate drops from 2^{128} to around $2^{104} - 2^{115}$, an 18% gap. Taking Y_2 into consideration will result in an even higher estimation for the drop in security.

Table 2. Estimated security level of HE instances after combining data from our extracted leakage. For our baseline we use N and q for 2^{128} classical bits security from [1].

N	$\log_2 q$	Security estimation		
		2nd iter.	3rd iter.	4th iter.
1,024	27	2^{124}	$2^{111} - 2^{124}$	$2^{107} - 2^{119}$
2,048	54	2^{121}	$2^{109} - 2^{121}$	$2^{105} - 2^{117}$
4,096	109	2^{120}	$2^{107} - 2^{120}$	$2^{103} - 2^{115}$
8,192	218	2^{120}	$2^{107} - 2^{120}$	$2^{104} - 2^{115}$
16,384	438	2^{120}	$2^{108} - 2^{120}$	$2^{104} - 2^{115}$
32,768	881	2^{120}	$2^{108} - 2^{120}$	$2^{104} - 2^{115}$

5 Responsible Disclosure

The paper was disclosed to the HEAAN, HELib, Palisade and SEAL teams before its publication.

6 Conclusions

In this work, we identified a potential vulnerability that can occur in almost all HE libraries, where the NTT algorithm and specifically the Harvey butterflies can be compiled naïvely or with a malicious intention to have a non constant-time Assembly. While there is, no immediate vulnerability when the libraries are used as intended, we explained how a simple code modification, a simple misuse of the library (e.g., compiling and using it in debug mode), or even a malicious adversary that messes with the library setup can flip the situation. Specifically, we evaluated the potential leakage of a secret from the NTT implementation of SEAL and showed the hardness degradation of the R-LWE instances used therein. In fact, the degradation can be further reduced by considering more NTT layers or taking into account the coefficients that are limited to two values.

While it is not always possible to control the compiler results, we recommend that library owners either use hand-written Assembly for critical code paths or modify the code so it won't use branches. Such code exists, for example, in HElib when compiled with the flag `NTL_AVOID_BRANCHING` or in the vectorized implementation of [3,4].

We hope that this paper will increase the awareness of cryptographic libraries owners to the implications of trusting modern compilers that are designed to bring optimized binaries without always taking security aspects into consideration.

A NTT Algorithms

Algorithms 3 and 4 are the forward and inverse NTT algorithms from [19], respectively.

Algorithm 3. CT radix-2 NTT [19]

```
    Input: $a \in \mathcal{R}_q$, $N$ a power of 2, $q$ a prime satisfying $q \equiv 1 \pmod{2N}$, $\psi_{rev}$, which holds the powers of $\psi$ in bit-reversed order.
    Output: $\tilde{a} = NTT_\psi(a)$ in bit-reversed order.
 1: procedure CT RADIX-2 NTT($a, N, q, \psi_{rev}$)
 2:     $t = N$, $\tilde{a} = a$
 3:     for ($m = 0$; $m < N$; $m = 2m$) do
 4:         $t = t/2$
 5:         for $i = 0$; $i < m$; $i$++ do
 6:             $w = \psi_{rev}[m + i]$
 7:             for ($j = 2it$; $j < (2i + 1)t$; $j$++) do
 8:                 $(X, Y) = (\tilde{a}_j, \tilde{a}_{j+t}w)$
 9:                 $(\tilde{a}_j, \tilde{a}_{j+t}) = (X + Y, X - Y) \pmod{q}$
10:     return $\tilde{a}$
```

Algorithm 4. Gentleman-Sande (GS) Radix-2 InvNTT [19]

```
    Input: $\tilde{a} \in \mathcal{R}_q$, $N$ a power of 2, $q$ a prime satisfying $q \equiv 1 \pmod{2N}$, $\psi^{-1}_{rev}$, which holds the powers of $\psi^{-1}$ in bit-reversed order.
    Output: $a = InvNTT(\tilde{a})$ in bit-reversed order.
 1: procedure GENTLEMAN-SANDE (GS) RADIX-2 INVNTT($\tilde{a}, N, q, \psi^{-1}_{rev}$)
 2:     $t = 1$, $a = \tilde{a}$
 3:     for ($m = N/2$; $m > 0$; $m \gg= 1$) do
 4:         for ($i = 0$; $i < m$; $i$++) do
 5:             $w = \psi^{-1}_{rev}[m + i]$
 6:             for $j = 2it$; $j < (2i + 1)t$; $j$++) do
 7:                 $(X, Y) = (a_j, a_{j+t})$
 8:                 $(a_j, a_{j+t}) = (X + Y, w(X - Y)) \pmod{q}$
 9:         $t = 2t$
10:     for ($j = 0$; $j < N$; $j$++) do
11:         $a_j = a_j \cdot N^{-1}$
12:     return a
```

B Generating the Primes

For reproduction purposes, we provide the SageMath script we used to generate the primes in Sect. 4.

```
import numpy as np
import math
import random
# Generate a random prime p in [n+1, n^3+1],
#  where p = 1 mod n
def primeGen(n):
    i=0;
    while True:
        x = randrange(n^2);
        if is_prime( x*n +1):
            return x*n+1
random.seed(120)
primes = [primeGen(2^16) for i in range(1000)]
```

References

1. Albrecht, M., et al.: Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. https://homomorphicencryption.org/standard/
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
3. Boemer, F., Kim, S., Seifu, G., de Souza, F.D., Gopal, V.: Intel HEXL: accelerating homomorphic encryption with Intel AVX512-IFMA52. Technical report (2021). https://eprint.iacr.org/2021/420
4. Bradbury, J., Drucker, N., Hillenbrand, M.: NTT software optimization using an extended Harvey butterfly. Technical report (2021). https://eprint.iacr.org/2021/1396
5. GCC bugs: [Bug c++/98801] New: Request for a conditional move built-in function (2021). https://www.mail-archive.com/gcc-bugs@gcc.gnu.org/msg676288.html
6. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jacobson Jr., M.J. (eds.) Selected Areas in Cryptography - SAC 2018, pp. 347–368. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-10970-7_16
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
8. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of Complex Fourier Series. Math. Comput. **19**(90), 297–301 (1965). https://doi.org/10.2307/2003354

9. Daan, S.: LLVM provides no side-channel resistance (2019). https://dsprenkels.com/cmov-conversion.html
10. Dai, W., Sunar, B.: cuHE: a homomorphic encryption accelerator library. In: Pasalic, E., Knudsen, L.R. (eds.) BalkanCryptSec 2015. LNCS, vol. 9540, pp. 169–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_11
11. Ducas, L., et al.: CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation (2017). https://pq-crystals.org/dilithium/data/dilithium-specification.pdf
12. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: exploiting branch tracing against StrongSwan and electromagnetic emanations in microcontrollers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1857–1874, CCS 2017. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134028
13. Gentleman, W.M., Sande, G.: Fast Fourier transforms-For fun and profit. In: AFIPS Conference Proceedings - 1966 Fall Joint Computer Conference, AFIPS 1966, pp. 563–578 (1966). https://doi.org/10.1145/1464291.1464352
14. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 359–386. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_13
15. Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_31
16. Harvey, D.: Faster arithmetic for number-theoretic transforms. J. Symbolic Comput. **60**, 113–119 (2014). https://doi.org/10.1016/j.jsc.2013.09.002
17. Jung, W., et al.: HEAAN demystified: accelerating fully homomorphic encryption through architecture-centric analysis and optimization (2020)
18. Laine, K.: Simple encrypted arithmetic library 2.3.1. Technical report, Microsoft, WA, USA (2017). https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf
19. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_8
20. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 1–35 (2013). https://doi.org/10.1145/2535925
21. Lyubashevsky, V., Seiler, G.: NTTRU: truly fast NTRU using NTT 2019. IACR Trans. Cryptographic Hardware Embed. Syst. **2019**, 180–201 (2019). https://doi.org/10.13154/tches.v2019.i3.180-201
22. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_25
23. Sadegh Riazi, M., Laine, K., Pelton, B., Dai, W.: HEAX: an architecture for computing on encrypted data. In: International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS, pp. 1295–1309 (2020). https://doi.org/10.1145/3373376.3378523
24. Schwabe, P., et al.: CRYSTALS-KYBER (2020). https://pq-crystals.org/kyber/

25. Van Bulck, J., Piessens, F., Strackx, R.: SGX-Step: a practical attack framework for precise enclave execution control. In: 2nd Workshop on System Software for Trusted Execution (SysTEX), pp. 4:1–4:6. ACM, October 2017. https://doi.org/10.1145/3152701.3152706
26. Victor, S.: NTL - a library for doing numbery theory - version 11.5.1, commit 91acd5b3a7df709c0d8bf88a99a24bc340dc34f7 (2021). https://github.com/libntl/ntl
27. Yuriy, P., Kurt, R., Gerard, R.W., Dave, C.: PALISADE Lattice Cryptography Library, commmit d76213499af44558170cca6c72c5314755fec23c (2021). https://gitlab.com/palisade/palisade-release

Predicting the Direction of Changes in the Values of Time Series for Relatively Small Training Samples

Sergey Frenkel(✉)

Federal Research Center "Computer Science and Control", Russian Academy of Sciences, Moscow, Russia
fsergei51@gmail.com

Abstract. Among the tasks of prediction, the task of predicting the signs of increments (direction of change) of the time series process is singled out separately. The essential difference of this problem from the prediction of values (ordinates) of time series implementations is the weak correlation of increments, which, from the point of view of the classical theory of time series forecasting, leads to certain difficulties. First of all, this refers to a non-stationary process, the prediction of which requires constant retraining of the parameters of the predictors used (for example, the weight coefficients of the neural network) over relatively short time intervals, which leads to time costs. This may be unacceptable, for example, when using the prediction of traffic characteristics in telecom networks, when predicting the direction of the gradient in optimization problems, etc.

The paper proposes the use of some results of the theory of random processes for the estimated fast prediction of increments with acceptable accuracy. The proposed procedure of the fast prediction is a simple heuristic rule for predicting the sign of the increment of two neighboring values of a random sequence.

Keywords: Prediction · Time series · Networks traffic

1 Introduction

Among the tasks of forecasting, the task of predicting signs of increments (direction of change) of random time series is singled out separately [1, 2]. Under time series is understood as a sequence of numerical data that occur in successive order over some period of time. The mathematical model of prediction in this case are mainly various models of random processes theory [3], and the increment is $x_{t+1} - x_t$, where x_{t+1}, x_t are the time series members (or ordinates of the corresponding random process) in the neighbor moments t, t+1.

Let's call such data and prediction models probabilistic. As the analysis of publications shows, the main consumer of models, methods and tools for Machine Learning based (ML) forecasting the sign of changes is the field of financial predictions ("financial mathematics" [4]) and econometrics. Within these models, problems are solved due to a combination of the properties of predictability used in the modeling of mathematical

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 118–134, 2022.
https://doi.org/10.1007/978-3-031-07689-3_9

objects (for example, random processes [3]) and specific assumptions and properties of financial trading, for example, the postulated relationship between the dependence on the volatility of asset returns and the dependence on the sign of asset returns (and, hence sign predictability) [2]. For example, they use the fact that volatility and higher-order conditional moments of return are related when predicting the sign.

At the same time the question of the predictability of the direction of change (that is sign) in the ordinate of the process is interesting for many problems not related to financial and economic issues, and for which there are no analogues of the theoretical results of financial mathematics.

An example of such tasks of sign prediction of increments is the optimal real-time correction of changes in the amount of resources that a cloud service provider can allocate to serve its customers, depending on the direction (decrease or increase) of the evolution of the state indicators (traffic volume, for example) of the network [5].

Another example is when attacks on a network are performed in a way that cannot be detected by anti-virus software, and analysts must rely on analysis of changes in network traffic (volume of traffic), changes in service activity, or direction of changes in traffic intensity entries in the log file.

Packet count reversal sign prediction can be useful to prevent packet loss under threat of buffer overflow [9]. The question of predicting the sign of the increment of random variables is also interesting in the problems of studying brain activity [1].

This paper considers the possibility of the sign increment prediction without using any theoretical assumption and conceptions from financial area.

Within the framework of probabilistic models, prediction in modern literature [6, 17] is understood as the computation of the conditional probability $\gamma(x_{t+1}|x_1, x_2,\ldots, x_t)$, defined on all elements of the sequence (time series), all members of which take these values from a domain A (binary, natural, numerical etc.) for which the natural condition is satisfied $\sum_{a\in A}\gamma(x_{t+1} = a|x_1, x_2,\ldots, x_t) = 1$. The function $\gamma()$ is called a *predictor* [2].

Note, that the predictor is also called software products (for example, some modules of cloud MS AWS Azure ML, Google Cloud ML, etc.), which are used for prediction.

In the following, these software products will be referred to as “Prediction tools” (PT), which may be a subsystem of Intelligent Decision Support (IDS) [7], and use the word “predictor” in both senses with the necessary explanations, if this does not follow directly from the context.

In many modern tools based on Machine Learning (ML), the formation of a forecast for data values at the moments in the future is performed as a search for internal patterns and relationships without using any classical mathematical procedures (Markov models or other models of mathematical probability theory) or lying in their basis of formal theories. Training is performed on examples rather than a mathematical model, although using various mathematical tools such as linear or logistic regression in intermediate stages - for example, algorithms based on search trees and boosting (XGB etc.), neural networks [8]. These circumstances are often supplemented by the opacity of the transformation of the input data (i.e., the construction of features), which makes it difficult to assess the impact of certain properties of the input data on the result.

This is why the a prior estimation of how a specific prediction tool is effective from the point of view of a PT user is difficult and ambiguous.

At the same time, although most of the modern ML tools do not explicitly use probabilistic models, indirectly certain statistical properties, primarily indicators of the dependence of past and future values, affect the effectiveness of the forecast. Therefore, it is interesting to consider the possibility of these characteristics in the field of Predictability for predicting changes in processes of various nature that meet only the mathematical constraints of Predictability using only directly measured characteristics of the random processes under consideration, and not those that follow from theoretical and empirical representations of a certain subject area.

Such a consideration makes sense for operational decision-making systems, for essentially non-stationary operation modes, when it is not possible to consider long sequences for predictor training. The fact is that for highly non-stationary time series, the learning process is associated with constant retraining (in a certain sliding window). At the same time, parameters (for example, weight coefficients of neural network predictors) updated as a result of training should be constantly calculated. This indicates possible large time costs even on the most expensive computing equipment (GPU, clusters) [10]. Therefore, one of the questions is to predict on relatively small stationary sequences, so that the sequence (x_1, x_2,..., x_t) in the expression $\gamma(x_{t+1}|x_1, x_{2,}..., x_t)$, can be considered as stationary.

To connect the above with our problem of predicting the sign of a time series members increments, we point out that this problem has a certain specificity compared to predicting their values themselves, for which the existing methods and tools for predicting time series were mainly developed. One of the manifestations of this is that we cannot directly use the results of predicting the value of the time series x_{i+1} (the amplitude of the corresponding random process) using well-established tools (for example, ARIMA) to predict the sign of the difference $x_{i+1} - x_i$. Firstly, this is due to the fact that the main time series prediction algorithms are tuned to the criterion of minimizing the root-mean-square loss function, which is not sensitive to changes in the sign of increments. Secondly, this is due to the complex non-linear nature of the dependence between the amplitude of the time series changes and the sign of the increments [1]. Further, the correlations between the differences $x_{i+1} - x_i$ themselves are very weak, which makes it difficult to predict within the framework of the binary sequence model, where 0 and 1 would correspond to the signs of the differences.

In the paper, we consider possible approaches to predicting the sign of the change in the increments of a random process, for example, the distribution of extrema of a stationary random process [11]. The parameters of the probability distribution function of the time series are also considered as possible characteristic of predictability of the sign of the increment. We study one of the previously proposed models for predicting the [12] sign from the point of view of its suitability for real data, for example, predicting the sign of changes in corporate network traffic volume. This model of behavior of the sign of the increment of a continuous random process is supplemented by a purely binary model of predicting the sign from samples of relatively small length.

The goal of this consideration is to study the possibility of predicting only by directly observed and measured data characteristics, without using computationally complex tools such as Artificial Neural Networks or boosting, which may be unacceptable, for example, for operational control in networks. On the other hand, by using simple data characteristics and receiving some a pro pry understanding about theie intrinsic predictability, one can try to improve the performance of complex instruments. Therefore, we consider of the sample autocorrelation function along with other directly estimated characteristics which are similar to the frequency of increments sign change on a training interval as a possible preliminary characteristics of possibility of forecasting implementation for management of a specific system.

The use of these models is illustrated by comparison with known prediction algorithms present in Python libraries. It is shown that such easily calculated characteristics of random processes as the value of the autocorrelation function in the first lag and the proportion of positive increments in the considered interval can give a priori estimates of the predictability of the sign, i.e. play the role of a characteristic of predictability.

2 On the Predictability of the Increments Sign of Random Sequences and Processes

Since the most natural and most used and discussed in the prediction community are probabilistic data models, first of all we will consider the possibility of using the main models of random theory to solve problems of predicting signs in modern prediction algorithms. First of all, in accordance with the approaches to Predictability accepted in the theory of random processes (see also [9]), we will say that predicting the sign of the difference (increment) $x_{t+1} - x_t = \Delta_{t+1}$ of the original process $X = x_1, \ldots, x_{t+1}$ at time t+1 is possible if the prehistory $\Im_t$ somehow affects the conditional expectation $E(sign(\Delta_{t+1})/\Im_t))$. Let us clarify that $\Im_t$ corresponds to filtration in the theory of random processes [3]. A filtration $\Im_s \subseteq \Im_t$ on a probability space $(\Omega, \Im_t, P)$ is a set (collection) of sub-sigma-algebras satisfying whenever $s \leq t$. As usual in the practical application of the theory of random processes, sigma-algebra is understood as possible combinations of events associated with the implementation of the process, combined with the help of known set-theoretic operations. Accordingly, at a fixed t, $\Im_t$ is the set of possible events occurring up to and including time t. Some examples of such events are listed in the following definition.

Definition 1. The change in the sign of the increment of values in random data is predictable (and the data has the Predictability property) if $E(sign(\Delta_{t+1})/\Im_t) \neq E(sign(\Delta_{t+1}))$, where $\Im_t$ is a part of that probabilistic space, on which a random process considered defined, which corresponds to the observed values x_t, or $x_{t-k}, \ldots, x_t$, or, say, to the regression estimate of the conditional mean x_{t+1}, etc.

From the point of view of the approach to forecasting adopted in mathematical statistics, this means that we can express, say, the probability of a positive increment as $Pr(\Delta_{t+1} > 0|\Im_t) = E(I(\Delta_{t+1} > 0)|\Im_t)$, where I() is the indicator function, E() (as before, and in the future) is the signature of mathematical expectation (according to the distribution of values Δ_t).

Assume, for example, that the source process is normal, and that a normal approximation of the conditional distribution is used (and about which the developer or user of PT/IDS might have considerations about the acceptability of such an approximation).

If we assume that the probability distribution of increments really has a good normal approximation with respect to the conditional variance, with the mathematical expectation determined over the entire region of (time) increments of the process under consideration, i.e., the distribution $F(\Delta_{t+1}|\Im_t)$ can be approximated by normal distribution $N(a, \sigma^2(t + 1|t))$, where:

$a = E(\Delta)$, $\Delta = \{\Delta_{1,\ldots,}\ \Delta_t,\ \Delta_{t+1}\}$ are the increments over the entire area of consideration of the random sequence $X = \{x_{t-k}, \ldots, x_{t,\ldots}\}$,

$\sigma^2(t + 1|t) = E(\Delta - E(\Delta))^2$ is conditional variance of the increments over time in which events from $\Im_t$ occur, in particular, over the entire time interval (t−k,…,t) on which the values $x_{t-k}, \ldots, x_t$ are observed.

Then it is easy to show [2] that the probability of a positive sign:

$$\Pr(\Delta_{t+1} > 0) = \Phi(a/\sigma(t + 1|t)),$$

where Φ is the standard normal distribution function.

It is clear from above, that zero mean will make the sign unpredictable (at least for the normal distribution of the increment) in the sense of the definition given above.

For an arbitrary distribution F from [13] it can be obtained that the minimum estimate of the forecast error Δ'_{t+1} relative to the true sign (Δ_{t+1}) of the estimated "loss function":

Loss $(\Delta_{t+1,}\ \Delta'_{t+1}) = E_t(I(\Delta_{t+1} > 0) - \Delta'_{t+1|t})^2$ is achieved by the estimate:

$$\Delta'_{t+1|t} = EI(\Delta_{t+1} > 0|\Omega_t) = P(\Delta_{t+1} > 0|W_t) = 1 - F(-a_{t+1|t}/\sigma(t + 1|t))$$

where Ω_t is a set of events related to Δ_t. We repeat, the conditional expression (…|t) means the calculation of the observed values up to the moment t inclusive.

Consequently, the dynamics of change in variations will affect the forecast of the sign in all cases when the conditional mean is not equal to zero. Then the sign of the increment is predictable, even if the conditional mean is unpredictable at zero expectation.

Note also that if the distribution of F is skewed, then the sign can be predicted even if the mean is zero: in this case, the time-varying skewness can be the determining factor in predicting the sign.

But for a non-zero conditional mean, even if the distribution is symmetric about the conditional mean and the conditional mean is constant by assumption (in contrast to the t-dependent variation $\sigma(t + 1|t)$, the sign of the increment is predictable from the point of view of the above definition.

So, conclusions about the predictability of a sign can only be based on knowledge of the distribution law of time series modeled by a random process with this law, and on the parameter of the distribution, namely, average, variance, skew coefficient etc.

For further, we note that the considered approach to determining the predictability of the sign of increments has a number of contacts with another sign prediction paradigm based on the consideration of a binary sequence corresponding to signs. For example, it is natural to assume that the value of the new binary sequence is 0 if $x_t - x_{t-1} \leq 0$, and 1 in the case of a positive increment, and use this binary sequence to predict the corresponding statistics of this binary sequence. At the same time, obviously, this statistics corresponds to the statistics of changes in increment signs or extrema.

In the most general form, the concept of predictability for binary sequences can be represented in the following form.

Definition 2. A sequence s_1, s_2, …s_n from some distribution S is predictable for a predictor that implements some polynomial algorithm A in complexity if for each $1 \leq i \geq n$ and any polynomial algorithm for estimating the next value rejects any statistical test [14] for inequality

$$\left|\mathrm{Prob}\left\{A\left(s_1^{i-1}\right) = s_i - 1/2\right\}\right| \leq O(\nu(n))$$

where $A\left(s_1^{i-1}\right)$ is an event consisting in observing a segment of the sequence up to the moment i,

$O(\nu(n))$ denotes a function decreasing faster than any polynomial in n.

Let's emphasize, however, that the measure of predictability must allow the choice of the most suitable prediction algorithm-and-tool from the point of view of a given quality criterion. Therefore, it is impractical to focus on type of probabilistic distribution of the random process and the increments, the evaluation of which is a very difficult problem when choosing prediction tools. In view of this, we will consider simpler and more practical approaches to assessing predictability when choosing prediction tools.

3 On the Use of the Random Processes Conceptions to Predict the Sign

Let us first of all consider the possibility of using the properties of the local structure of random processes that model the predicted data. As such a characteristic, consider the extrema of a random process.

3.1 Change of Sign and Local Extremes

It is possible to formulate the problem of predicting the sign of the change in s_i as predicting the appearance of a local extremum of the implementation of the corresponding random process at time t.

Indeed, it is easy to see that for any interval [t−2, t]:

if $x_{t-1} = \max_{[t-2,\, t]}(X)$ then $\mathrm{sign}(x_t - x_{t-1}) =$ "−".
if $x_{t-1} = \min_{[t-2,\, t]}(X)$ then $\mathrm{sign}(x_t - x_{t-1}) =$ "+".
if at time t−1 the process reaches neither maximum nor minimum, then the sign of the increment $x_t - x_{t-1}$ at time t coincides with the sign of the increment $x_{t-1} - x_{t-2}$.
if at time t−1 the process reaches neither maximum nor minimum, then the sign of the increment $x_t - x_{t-1}$ at time t coincides with the sign of the increment $x_{t-1} - x_{t-2}$.

Accordingly, the probability distribution of the number of negative and positive increments coincides with the distribution of the number of corresponding local extrema. Since the predictor for us, as stated in the Introduction, the prediction is in any case associated with the calculation of the conditional distribution $\gamma(x_{t+1}|x_1, x_2,\ldots, x_t)$, the conditional probabilities of the occurrence of maxima and minima with known past observed values can be used to predict.

In the theory of extrema of random sequences and processes, by now, a huge number of results have been accumulated on the probability distribution of the number of local extrema over finite intervals [t_0, t], mainly for Gaussian stationary random processes. In addition, results were obtained on the conditions (probabilities) of the absence of extrema at a certain level (values of x_j) [11]. In other words, these are the conditions for a monotonous change in the values of a random process on a given interval, and, accordingly, the absence of a sign change.

The first one gives a simple sufficient condition for a one-parameter random process not to have, almost certainly, critical points at a certain specified level of the data $x_1, \ldots x_t$ values. The second result states that under mild conditions, a Gaussian process defined on a quite general parameter set, with probability one does not have local extrema at a given level [11].

An analysis of the models for the distribution of extrema shows the possibility of using characteristics that are directly calculated from samples of random processes, which, within the framework of the considered process models, allow us to see the nature of their influence on the prediction accuracy. For example, the mathematical expectation of the number of local extrema (maxima and minima) depends on the spectral characteristics of the random processes under consideration, which, under broad assumptions about the probabilistic model of the process, are associated both with the characteristics of the autocorrelation function (which does not require any model of the process behavior to calculate), and with the frequency of change of maxima and minima [11].

However, in reality, all these results can only be used for stationary Gaussian processes, which, as well as for the approach considered above [2], is a serious limitation, since, in this case, the prediction turns out to be dependent on the data model, and the associated assumptions about the probabilistic properties data, which is desirable to avoid. At the same time, most of the results are asymptotic in nature (both in time and in terms of the values (amplitudes) $x_{i.}$

Let us therefore consider how these model-independent characteristics of a random process can be used to predict the sign of increments.

3.2 Impact of Time Series Autocorrelation on the Sign Prediction

It is well known (and intuitively clear) the influence of correlation properties on the prediction of numerical values (amplitudes) of random processes (ARIMA, etc.) [15]. For example, the prediction error for the simplest known linear forecast for a stationary time series y is $\varepsilon = (1 - \rho^2)\mathrm{var}(y)$, where the correlation coefficient is $\rho = \mathrm{ACF}(1)$, ACF(1) means the autocorrelation function in the lag 1.

In [12] the question is raised whether it is possible to predict the sign of the difference of neighboring distributed observations $x_1, x_2, \ldots$, with a probability of correct prediction $>1/2$ (i.e., more than with the probability of a decision on the outcome of a fair coin toss).

Consider a sequence of centered random variables $y = \{y_i = x_i - (Ex_i)\}, i = 1, \ldots t \ldots$, with zero mean and probability density distribution P(y). Note that centering is often used in the practice of predicting various processes to eliminate the trend. This is necessary to work with models of stationary processes, and the selected trend is used as additional information [1]. If we assume independence of y_i, then the conditional expectation of its increments $y_{i+1} - y_i$ at the last observed value of y_i:

$$E(y_{i+1} - y_i | y_i) = E(y_{i+1} | y_i) - E(y_i | y_i) = -y_i.$$

Indeed, $E(y_i | y_i) = y_i$, and since the mathematical expectation is zero and all increments $\{y_i\}$ are independent $E(y_{i+1} | y_i) = 0$, which implies $E(y_{i+1} - y_i | y_i) = -y_i$.

Accordingly, considering the conditional mean as a sign predictor, the following rule is proposed for predicting the sign of the difference $y_{i+1} - y_i$:

$$\operatorname{sign}(y_{i+1} - y_i) = -\operatorname{sign}(y_i) \tag{1}$$

In [12] it is stated that uncorrelated increments are sufficient to fulfill the indicated sign relation, which is not always true, although it can be used in practice. Indeed, on the one hand, in real processes, the dependence often takes place due to a non-stationary trend, which is eliminated by centering $y_i = x_i - Ex_i$. However, most of the real data, such as changing traffic volumes, changes in the number of requested IP addresses, etc. somehow connected with a random change (decrease-increase) of some factors, and can be represented by random walk processes that are close to processes with independent increments [1].

It is possible to give more subtle mathematical reasoning on this matter [1].

The fraction of successful predictions according to (1) for sufficiently long (sequences proposed to be assessed as

$$R = 1/2 + (1 - F(0))F(0) \tag{2}$$

where:

$$F(x) = \int_{-\infty}^{x} dy P(y)$$

$y = y_{i+1} - y_i$; recall that P() is the corresponding probability distribution function (pdf).

Obviously, $(1 - F(0))\, F(0)) \leq 1/4$, because F(0) as a probability <1, and the maximum is reached at $F(0) = 1/2$.

Thus, under certain situations, it is possible to achieve R up to 0.75, which is a very good result from the point of view of forecasting practice.

If the analytical model (expression) of the probability density P(y) is known, then F() can be expressed as an ordinary integral (and not Stieltjes one, as in [12]).

In what follows, we will call rule (2) SC - (“Sign criterion-based” or CS - predictor).

Formulas (1) and (2) mathematically express, at first glance, a paradoxical result, that when guessing the sign of the increment of independent random variables from the previous value of the centered process, one can get an average success rate of 0.75. The formal explanation is that difference is a differential operator known to introduce short-range correlations (reflected by autocorrelation function of the time series) into uncorrelated process [16], corresponding to (1).

It is possible, however, to give (1) a certain elementary probabilistic substantiation. As it is easy to see, the truth of (1) depends on combinations of 3 conditions for y_{t+1}, y_t (all possible relations ">0" "0<" "$y_{t+1}< >y_t$") in total such conditions are $2^3 = 8$, namely, it will be true in the cases (considering that zero values is impractical because of measurement practice):

$$\begin{array}{ll} y_{t+1} > 0,\ y_t > 0, & y_{t+1} < y_t \\ y_{t+1} < 0,\ y_t > 0, & |y_{t+1}| > y_t \\ \ldots\ldots\ldots\ldots\ldots\ldots & |y_{t+1}| < y_t \\ y_{t+1} > 0,\ y_t < 0, & |y_{t+1}| < y_t \\ \ldots\ldots\ldots\ldots\ldots\ldots & |y_{t+1}| > y_t \\ y_{t+1} < 0,\ y_t < 0, & |y_{t+1}| < y_t \end{array}$$

And it will be false if :

$$\begin{array}{l} y_{t+1} > 0,\ y_t > 0,\ y_{t+1} > y_t \\ y_{t+1} < 0,\ y_t < 0,\ |y_{t+1}| < |y_t| \end{array}$$

Assuming that all of them are equally probable, we obtain the a priori probability of fulfilling the "true" inequality 0.75!

Of course, the assumption of equiprobability may not be completely satisfied. If the probability of true combination is p_1, and false one is p_2, then the probability of correct prediction is (Bayesian formulas):

$$P_{corr} = 0.75p_2/(0.75p_1 + 0.25p_2)$$

That is if p_1 even less then p_2 in two times, $P_{corr} > 0.6$, that is enough form the point of view of current prediction practice. However, there is no reason to believe that 2 combinations leading to incorrect predictions will occur on some interval in several times more often than leading to correct.

We will review the experimental evidence and give some explanations based on it that such a simple prediction method can be used for arbitrary data sets.

Note that the authors [12] use a slightly different result (obtained from the same model), but they consider the sign of the increment as probably opposite to the past increment, not to the centered last value of the predicted time series, i.e. if $\delta y_t = y_t - y_{t-1}$. They are trying to predict the change in δy_t with the strategy that $\delta y_{t+1} = y_{t+1} - y_t$ predicts the opposite sign as

$$\text{sign}(\delta y_{t+1}) = -\text{sign}(\delta y_t) \tag{3}$$

Will call it as "delta-sign" (DS)-predictor.

They demonstrate it proposing to use the above considerations about predicting the sign of the increment as the opposite sign of the last observation in relation to daily closing quotes q(t) and, estimating price fluctuations. This is based on some assumptions of financial mathematics, primarily on the "efficient market hypothesis" [18], according to which market prices should only respond to new information (i.e., the latest data), and random walk patterns that are consistent (say, daily) the price returns of liquid organized markets with essentially approximately symmetrical distributions.

Experiments were performed with prediction by (1) and (3) for a network traffic data, for which the assumptions about the random walk and the nature of the dependence on the previous values are obviously not fulfilled. The results obtained indicate a significant advantage of (1) ("*SC-predictor*").

Let us consider a possible practical application of rule (1), which we will call the *SC-predictor* (Sign criterion-based) for real data sets for which it is difficult to test the hypothesis of sample independence, for example, due to the small sample size, according to which a statistical test could be performed, but the empirical autocorrelation function can be computed.

4 Ways to Use SC as a Predictor of the Increments Sign

Consider two conditions for using the SC predictor following from the previous section.

First of all, one of the signs of the possibility of using (2) is the uncorrelated (or extremely weak correlation) of successive differences in the values of the considered time series (process) with a tendency to negative correlation of the values $y_{i+1} - y_i$ and y_i, since the "anti-correlation" of two random variables means a tendency to change in the opposite direction.

It is intuitively clear that this should be expressed in terms of the frequencies (probability) of fulfilling the equality (1), as it has been shown in the previous Section.

Considering that our task is to study the possibility of using directly measured characteristics of the time series for prediction (albeit preliminary) and to evaluate the predictability of this time series based on them, for a better understanding of the possibility of using the predictor (1), it would be desirable to consider the study of the use of the sample autocorrelation function along with other directly estimated characteristics.

Set the sign of each y increment to binary 0 if it's negative, and 1 if it's positive.

Suppose that the resulting sequence of zeros and ones is a sequence of independent Bernoulli trials. Let "0/1loss" (or simply "0/1") be the measure of predictor user losses from prediction errors of the form "zero is predicted instead of the true value" and vice versa, equal to the proportion of incorrect predictions on a sequence of length n, i.e.

$$e_n = E_p\left(\left(\sum\nolimits_{i=1,n} I(b_t \neq x_t)\right)/n\right)$$

where $p = \text{Prob}(x_t = 1)$ - Bernoulli distribution parameter, I() - indicator function, b_t, x_t - predicted and true value, respectively.

It is known [17] that the optimal prediction rule (predictor) according to the criterion of minimum loss with respect to the measure e_n is the following:

$b_t = 1$, if $\text{Prob}(1) > 1/2$
$b_t = 0$, if $\text{Prob}(1) < 1/2$
and "rejection of the forecast" in case $\text{Prob}(0) = \text{Prob}(1) = 1/2$

In this case, as it is easy to see, the average losses are equal to the error probability $1 - p$, with $p > 1/2$, and p, if 0 is considered "success", which can be expressed as

$$L_p = \min(p, 1 - p)$$

Since we are dealing with relatively small sample sizes, which makes testing hypotheses about probabilistic characteristics inefficient, it is more practical to consider not probability estimates, but the frequencies of the corresponding events, in particular "1" (n_1) and "0" (n_0), $(n_1 + n_0) = M$ is the length of the sequence.

In this case, we consider all measured and calculated in the window of length M as a characteristic of only this segment of the data set without assumptions about its belonging to a stochastic ensemble of sequences. Accordingly, we draw conclusions about the effectiveness of the predictor only on this time interval, and therefore get rid of the need to evaluate the confidence intervals of probabilistic estimates.

Accordingly, the rule will be changed as:

1 if $n_1 > n_0$,
0 if $n_0 > n_1$,

Prediction is not performed if $n_1 = n_0$, and the loss (optimal according to the 0/1loss criterion) of incorrect predictions

$$L_n = \min(n_0, n_1) \tag{4}$$

That is the wrong predicted points in corresponding time interval is $\min(n_0, n_1)$. Let's call the considered prediction rule as "0/1" predictor.

The possibility of sharing SC and 0/1 predictors, the algorithms of which use only directly measured data characteristics, was experimentally evaluated using sets of records obtained both from Internet sources and sources devoted to the study of changes in the volume of traffic in networks [19], and from measurements using the IBM QRadar-NETflow tool of the number of so-called. "flows of events" for traffic in a network that implements a geographically distributed hybrid high-performance computing cluster of the Russian Academy of Sciences at different times of the day [10] (Fig. 1).

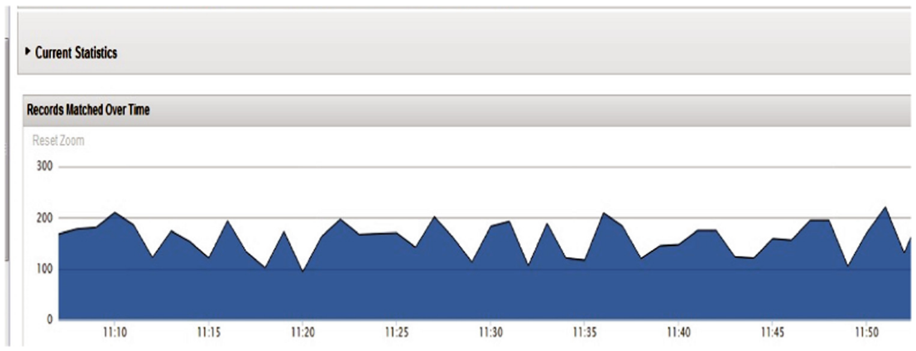


Fig. 1. "Flow" record snippet examples (from IBM QRadar NETflow). The abscissa axis is the time of day, the ordinate axis is the number of recorded event streams.

For example, for two plots of selected records, considered as a time series, the centered realizations (a, b) looks like this (Fig. 2).

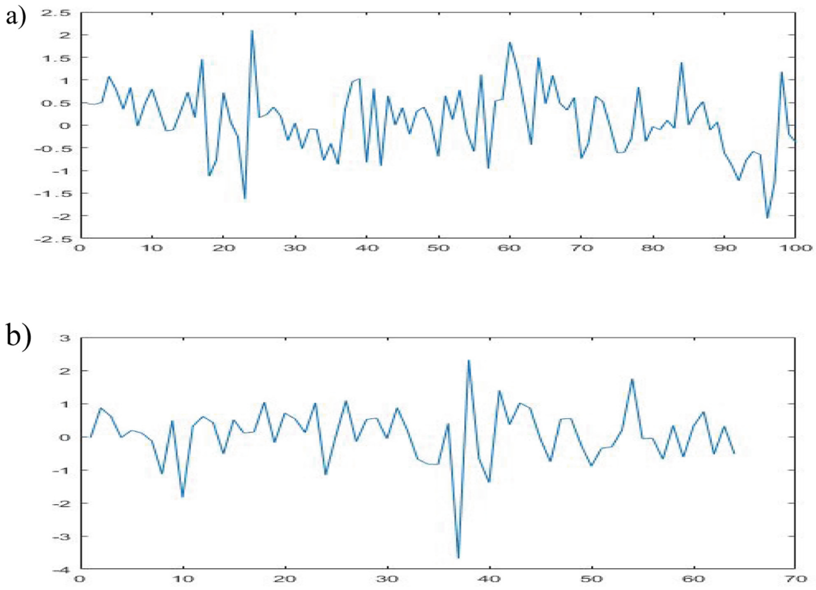


Fig. 2. Examples of centered records.

Autocorrelation function of increments of a centered process for the implementation of Fig. 2(a) (Fig. 3):

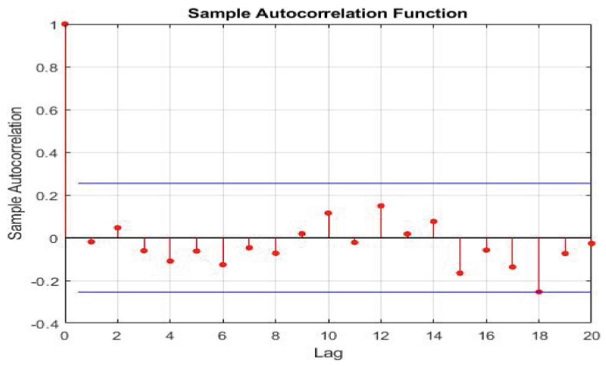


Fig. 3. Autocorrelation function for an example from Fig. 2(a).

The autocorrelation function in lag 1(ACF(1)) between successive increments is near zero, with a slight shift to the negative area (autocorrelation function ACF(1) = −0.04). The number of increments for which $y = \mathrm{sign}(y_{i+1} - y_i) = -\mathrm{sign}(y_i)$ is equal to 40 (that is $\#(\text{sign}(y_{i+1} - y_i) = -\text{sign}(y_i)) = 40$). Therefore, out of 63 increments at 64 time points, in 40 cases it is possible to correctly predict the signs of the increments if the sign of each next increment is predicted as a negation of the current one.

At the same time, the estimate of the proportion of positive increments (represented as binary 1) on the sample under consideration is $P = 31/61$ and, therefore, using the optimal 1/0 predictor, namely, assigning 1 (positive change) to all increments, we will make a mistake at 32 points, and not at 21, as when using the SC–predictor.

Note that if in this sample we consider the binary sequence as a segment of the Bernoulli sequence, with the parameter $p = 32/63$, then with a high degree of confidence we can talk about its randomness, and this is naturally reflected in the close-to-zero correlation of the original sequence of differences.

For another sample from the same dataset (shifted by about 4 months in terms of recording time), with a plot of differences (Fig. 2(b)):

The SC predictor gives 38 correct predictions over 64 records ($R = 0.6$). In this case, $n_1 = 35$, i.e. predicting for all points of the time interval "1" (i.e., minimizing the error of the 1/0loss type) we will get 3 more errors than using the SC predictor. That is these are already comparable, and 0/1 type predictor can be considered in the decision making about way of prediction along with SC type. The value of the autocorrelation function $ACF(1) = -0.21$. According to the rules accepted in statistics, random variables, the correlation (correlation coefficient) of which is less than 0.7, are considered as weakly correlated, since the probability of a strictly unidirectional or multidirectional change (for negative correlations) of these variables is extremely small, regardless of the type of probability distribution. Therefore, for us, the main indicator of the applicability of the SC predictor is precisely the extremely weak correlation, and the frequency of zeros - ones of the binary representation. Therefore (under the above assumptions about the properties of the random process) the following rule for predicting the value $\text{sgn}(x_{t+1} - x_{t+1})$ at the point t can be proposed.

SC-0/1-procedure:

– for time points $t_i \in \{t - M, t\}$, where M is the size of the observation window, i.e. the number of values of the time series available for observation up to the moment t+1, we perform the prediction of the signs of the increment, as

$$y_i = -\text{sign}(x_{t-M+i})$$

– Compute $\#SC = \Sigma_i I(\text{sign}(y_{i+1}) = \text{sign}(x_{t-M+i+1} - x_{t-M+i}))$
  where I is an indicator function, $i \in t_i \in \{t - M, t\}$).
– calculate the number of units n_1 in the window M; $n_0 = M - n_1$.
– if $(\#SC - \max(n_1, n_0)) > k_1$, $-k_1 = 1, 2, \ldots$- some natural number, chosen as the difference threshold between the accuracy of predictions, then we accept the forecast:

$$\text{sign}(y_{t+1} - y_t) = -\text{sign}(y_t)$$

Use 1/0 predictor if $\#SC < \max(n_1, n_0)$, and $\max(n_1, n_0) > k_2 = 1, 2, \ldots$

Otherwise, or one should abandon the prediction of the next value beyond the window, or start working with accessible ML-predictors (e.g., ANN, XGB, etc.).

Let us explain the meaning of the comparison #SC and max(n_1, n_0). We can consider the 0/1 predictor as trivial, in the sense that it makes it possible to predict the next value, and often with a high probability of correct prediction, if the proportion of 1 (or zeros) is significantly greater than $>1/2$. It does not use any information about the sequence other than the specified proportion of zeros and ones. Therefore, if the difference between SC and 0/1 is small, say 5% relative to the length of M, then we can say that SG works as a trivial predictor, and if the requirements for the quality of the forecast are very high, one should try some ML predictor. Note also, that if the loss function used gives large (inappropriate) value for a small number k, this may also be a reason to think that using a predictor close to trivial in its properties does not make sense, and try using more complex ML predictors. However, our experience shows that SC and sometimes 0/1 predictors are not worse than MLs (Table 1).

The same consideration is for the k_2 threshold. If the loss min(n_0, n_1) (4) is inappropriate, say the fraction estimated errors is $>k_2/M$ and it is very high from some informal viewpoint, the predictor is rejected. The considered window M corresponds to the training sequence of any modern predictor used in practice, when prediction is performed based on the results of predictions based on predictions inside the corresponding window, and checking their results (usually, the modes of preparing the predictor for predicting an unknown value are called "training" and "testing"). In this case, it is assumed that the main data properties (statistical characteristics, structure) do not change over a short period of time outside the training window (in particular, in one step, at time t+1 as in the described procedure).

Thus, the essence of this procedure is that predictions are made only using directly performed calculations - an increment at neighboring times and the number of positive/negative signs ("1/0").

The choice of the thresholds value k_1, k_2 is determined by the specifics of the problem and the goals of the prediction.

If from the point of view of the user of the forecast (it does not matter whether it is a real decision maker or a program that forms some control based on the forecast made and formalized estimates of its reliability and usefulness) it is unacceptable, then either the forecast is abandoned altogether, or more complex predictors are used, the use of which, however, may require significant hardware and time resources.

Table 1 illustrates the possibility of using the SC-1/0 procedure to predict the signs of increments over relatively short intervals, and how the uncorrelated of the increments reflects the possibility of using the sign of the current value to predict the sign of the increment. Natural numbers in the predictor columns mean the number of correct predictions, n_1 is the number of ones among 65 values of the binary transformation of the original record (according to Sect.) Obviously, $n_1 = 33$ means that according to the 0/1 prediction rule there would be $65 - 32 = 33$ correct predictions, and 32 incorrect (0s).

We can see, that the DS predictor (Sect. 3, suggested in [12]) is worse essentially that SC, because of reasons, considered above (the assumption from financials mathematics in not fairly for arbitrary time series).

Table 1 also shows the results of prediction by well-known self-learning-based predictors from the Python libraries for the same data intervals, using training on a segment of 66 records.

Default predictor hyperparameters were used and training was performed on default function parameters.

Table 1. Characteristics of the SG-0/1 procedure in comparison with self-learning based predictors.

SC	n1	ACF(1)	Length	MLP	XGB	LSTM	DS
41	32	−0.04	64	35	43	25	36
37	35	−0.25	64	30	46	31	32
53	53	−0.12	90	48	42	45	44
61	54	0.09	100	56	51	59	52
56	50	−0.22	100	59	51	60	53

5 Discussion and Conclusion

The choice and use of modern software tools for predicting the values of time series in various subject areas requires significant financial, hardware and time resources, since the quality of predictions depends on various properties of the series that are not explicitly reflected in predictor models. At the same time, modern predictors, such as XGB, MLP, SVM, LSTM etc. use a large set of options specified by parameters and hyperparameters (choice of which may take a long time), e.g., type of activation function (say, Sigmoid or RELU).

In addition, two levels of predictor optimization are required - by model (Sigmoid or RELU, Linear regression of logit etc.) and by parameters of each model, which can be very time consuming. Also as follows from the above, certain properties of time series can prevent high quality prediction. Therefore, it is desirable to have simple means for a preliminary assessment of possible predictability.

Such a pre-estimation procedure may make sense for operational decision-making systems in the conditions of the need to make predictions about significantly non-stationary data streams, when it is impossible to consider long sequences for predictor training. For highly non-stationary time series, the learning process is associated with enumeration of parameters (for example, neural network coefficients) if necessary, retraining of models caused by the above non-stationarity, which can be time-consuming even on the most expensive computing equipment (GPU, clusters) [10].

We can consider SC and 1/0 as indicators of predictability at some subsequence $x_{t-M}, x_{t-M+1}, \ldots, x_t$, from which it may be necessary to predict the increment $X_M = x_t - x_{t+1}$ at the next moment (interval). If we use a number series model, then as such a criterion, we can use the estimation of correlations of increments and prediction of the sign of increments according to the SC procedure in the M-window considered in the previous section, namely, the proportion of correct forecasts in the time interval immediately preceding t+1, and which is an estimate of the conditional mathematical expectation of a correct forecast. If the assessment result satisfies the consumer of the forecast (a real or virtual user, for example, a program that decides on the type of forecast based on this assessment), then he accepts it.

Otherwise, the conditional expectation of the 0/1 predictor is estimated, and if the result exceeds the proportion (or number) of correct SC, then its prediction is used. If the resulting estimate also does not satisfy the conditions, the issue of using professional PT from the set of available predictors, for example, as a cloud resource, is decided.

If we consider a subset X^M as a binary sequence obtained from the increment signs (see above) of length $|X^M| - 1$, it is possible to check two aspects of predictability: the degree of difference from randomness, and the proportion of correct predictions achieved by the 0/1 predictor under the assumption that the sequence in question is a sequence Bernoulli.

Benefit from the proposed technique may consist in the fact that in a fairly large number of cases (if the prediction must be performed on a fairly long time interval (T_1, T_2) so that the sequences mentioned above.

In fact, the proposed procedure is a simple heuristic rule for predicting the sign of the increment of two neighboring values of a random sequence [12], formulated as follows: predict the sign of the increment of a centered sequence opposite to the sign of the last observed value. It is shown that the probability of correctness of such a prediction will be $>1/2$. There are cases when this rule obviously does not work (Subsect. 3.2), but it is considered specifically for cases where a quick preliminary assessment of the predictability of specific data sets is needed, when the main criterion for predictive effectiveness is the proportion of correct predictions. There are two reasons for this rule.

One comes from elementary probability-theoretic ideas, naturally under the assumption of independence of the centered sequence. The second justification I would call “elementary-probabilistic”, since it is based simply on an estimate of the number of chances of this heuristic rule being fulfilled. These justifications somehow explain the somewhat paradoxical conclusion that the sign of a sequence of independent increments can be predicted better than by simply tossing a coin.

The considerations given in the article show the presence of significant scientific and practical potential in the study of the influence of characteristics directly calculated from data sets, with respect to which it is required to make certain predictions, on the efficiency of prediction algorithms. These characteristics are the parameters and characteristics of the probabilistic models that model the data.

References

1. Assaf Almog, A., Garlaschelli, D.: Binary versus non-binary information in real time series: empirical results and maximum-entropy matrix models. New J. Phys. **16**(9), 093015 (2014)
2. Christoffersen, P., Diebold, F.: Financial asset returns, direction-of-change forecasting, and volatility dynamics. Manage. Sci. **52**(8), 1273–1287 (2006)
3. Bosq, D., Nguyen, H.T.: A Course in Stochastic Processes. Stochastic Models and Statistical Inference. Kluwer, Dordrecht (1996)
4. Pliska, S.R.: Introduction to Mathematical Finance: Discrete Time Models. Blackwell, Maldon, Mass (1997)

5. Mozo, A., Ordozgoiti, B., Gómez-Canaval, S.: Forecasting short-term data center network traffic load with convolutional neural networks. PLoS ONE **13**(2), e0191939 (2018). https://doi.org/10.1371/journal.pone.0191939
6. Lysyak, A.S., Ryabko, B.Y.: Time series prediction based on data compression methods. Probl. Inf. Transm. **52**(1), 92–99 (2016). https://doi.org/10.1134/S0032946016010075
7. Hodgea, V., Krishnanb, R., Austina, J., Polakb, J., Jackson, T.: Short-Term Prediction of Traffic Flow Using a Binary Neural Network. Neural Comput. Appl. **25**(7–8), 1639–1655 (2014)
8. Chen, A., Law, J., Aibin, M.: A survey on traffic prediction techniques using artificial intelligence for communication networks. Telecom **2**, 518–535 (2021). https://doi.org/10.3390/telecom2040029
9. Shimall, T.: Traffic Analysis for Network Security: Two Approaches for Going Beyond Network Flow Data, 16 September 2016. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.513.7546&rep=rep1&type=pdf
10. Volovich, K.I., Denisov, S.A., Shabanov, A.P., Malkovsky, S.I.: Aspects of the assessment of the quality of loading hybrid high-performance computing cluster. In: 5th International Conference on Information Technologies and High-Performance Computing, ITHPC 2019. CEUR Workshop Proceedings, 16–19 September 2019, vol. 2426, pp. 7–11 (2019)
11. Leadbetter, M.R., Lindgren, G., Rootzén, H.: Extremes and Related Properties of Random Sequences and Processes. Springer, New York (1983). https://doi.org/10.1007/978-1-4612-5449-2
12. Sornette, D., Andersen, J.V.: Increments of uncorrelated time series can be predicted with a universal 75% probability of success. Int. J. Mod. Phys. **11**(4), 713–720 (2000)
13. Andersen, T.G., Bollerslev, T., Christoffersen, P.F., Diebold, F.X.: Volatility and correlation forecasting. In: Elliot, G., Granger, C.W.J., Timmermann, A. (eds.), Handbook of Economic Forecasting, pp. 778–878. North-Holland, Amsterdam (2006)
14. Lavasani, A., Eghlidos, T.: Bit test for evaluating pseudorandom sequences. Comput. Sci. Eng. Electr. Eng. **16**(1), 19–33 (2009)
15. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control. Wiley, New York (2008)
16. Rabiner, L.R., Rader, C.M.: Digital Signal Processing. IEEE Press Selected Reprint Series, 1 December 1994
17. Feder, M., Merhav, N., Gutman, M.: Universal prediction of individual sequences. IEEE Trans. Inf. Theor. **38**(4), 887–892 (1992)
18. Campbell, J.Y., Lo, A.W., MacKinlay, A.C.: The Econometrics of Financial Markets. Princeton University Press (1997)
19. Hasanov, V., Bayramov, H., Mehdiyev, H.: Prediction of network traffic based on neural-fuzzy analysis of changes in the volume of IP-packets. In: 9th International Conference on Theory and Application of Soft Computing, Computing with Words and Perception, ICSCCW 2017, pp. 438–445 (2017)

Machine-Learning Based Objective Function Selection for Community Detection

Asa Bornstein[1(✉)], Amir Rubin[1,2], and Danny Hendler[1,2]

[1] Ben-Gurion University of the Negev, 8410501 Beersheba, Israel
asabor@post.bgu.ac.il

[2] Department of Computer Science, Ben-Gurion University of the Negev, 8410501 Beersheba, Israel

Abstract. NECTAR, a Node-centric ovErlapping Community deTection AlgoRithm, presented by Cohen et al., chooses dynamically between two objective functions which to optimize, based on the network on which it is invoked. It was shown that this approach outperforms six state-of-the-art algorithms for overlapping community detection. In this work, we present NECTAR-ML, an extension of the NECTAR algorithm that uses a machine-learning based model for automating the selection of the objective function, trained and evaluated on a dataset of 15,755 synthetic and 7 real-world networks. Our analysis shows that in approximately 90% of the cases our model was able to successfully select the correct objective function. We conducted a competitive analysis of NECTAR and NECTAR-ML. NECTAR-ML was shown to significantly outperform NECTAR's ability to select the best objective function. We also conducted a competitive analysis of NECTAR-ML and two additional state-of-the-art multi-objective evolutionary community detection algorithms. NECTAR-ML outperformed both algorithms in terms of average detection quality. Multi-objective evolutionary algorithms are considered to be the most popular approach to solve multi-objective optimization problems and the fact that NECTAR-ML significantly outperforms them demonstrates the effectiveness of ML-based objective function selection.

Keywords: Community detection · Complex networks · Machine learning · Overlapping community detection · Supervised learning

1 Introduction

Social networks tend to exhibit community structure: They are partitioned to sets of nodes called communities (a.k.a. clusters), each of which relatively densely-interconnected, with relatively few connections between different communities. Revealing the community structure underlying complex networks is

This work was supported in part by the Cyber Security Research Center at Ben-Gurion University.

the focus of intense research (see e.g. [1,9,17,19,28]). While research focus was initially on detecting disjoint communities, in recent years there is growing interest also in the detection of overlapping communities, where a node may belong to several communities.

Many community detection algorithms are guided by an objective function that provides a quality measure of the clusterings they examine in the course of their execution (see e.g. [2,10,13,26,32]). Since exhaustive-search optimization of these functions is generally intractable (see e.g. [4,31]), existing methods search for an approximation of the optimum and employ heuristic search strategies. A key example of such heuristic is Blondel et al.'s algorithm [2], also known as the Louvain method (LM). The method aims to maximize the *modularity* objective function [34] as it employs a greedy local search heuristic that iterates over all nodes, assigning each node to the community it fits most (as quantified by modularity). Modularity assumes disjoint communities. Which objective functions should be used for overlapping community detection? According to what criteria should they be chosen? NECTAR, a Node-centric ovErlapping Community deTection AlgoRithm - presented in 2016 by Cohen et al. [7], generalized Blondel et al.'s method, so that it can be applied also to the overlapping case. NECTAR chooses dynamically between two objective functions which to optimize, based on the network on which it is invoked. The selection is made between Q^E, an extension of modularity for overlapping communities [5], and WOCC - Weighted Overlapping Community Clustering, an extension of WCC (Weighted Community Clustering) [29] for overlapping community detection. NECTAR selects which objective function to use based on the rate of closed triangles (out of all possible triangles) in the graph. This approach, as shown by Cohen et al. [7], outperforms six state-of-the-art algorithms for overlapping community detection.

In this work, we extend NECTAR, by taking into account multiple features of the input graph and using an ML-based classification model. This model selects an objective function which maximizes the quality of the clusters computed by NECTAR. Since there are several commonly-used metrics that can be used to quantify this quality, we generate an ML model per each such metric.

We analyzed 3,933 synthetic and 7 real-world networks, measuring the quality of our models, aiming to dynamically select, based on the properties of the graph at hand, which objective function should be used. Our analysis shows that in approximately 90% of the cases our model was able to successfully select the correct objective function to maximize the desired metric.

1.1 Our Contributions

We present NECTAR-ML, a node-centric overlapping community detection algorithm which employs a machine learning model for the selection of the objective function. To train our models, we generated a dataset of 15,755 synthetic networks, of various sizes and properties[1]. We conducted a competitive analysis of NECTAR

[1] The dataset was generated using a parallel computing framework we developed, available at https://github.com/asaborn/GenericMultiTasking.

and NECTAR-ML. NECTAR-ML was proven to be superior over NECTAR as it significantly outperformed NECTAR's ability to select the best objective function out of Q^E and WOCC. In addition, we have conducted an extensive competitive analysis of NECTAR-ML and two additional state-of-the-art multi-objective community detection algorithms. NECTAR-ML outperformed also both these algorithms in terms of average detection quality.

The rest of this paper is organized as follows. We survey key related work in Sect. 2. We present the NECTAR-ML algorithm in Sect. 3. We report on our experimental evaluation in Sect. 4. We conclude in Sect. 5.

2 Related Work

Blondel et al.'s algorithm [2], a.k.a. the Louvain method, is a widely-used disjoint community detection algorithm. It is based on a simple search heuristic that seeks to maximize *modularity* [34]—a global objective function that estimates the quality of a graph partition to disjoint communities. Chen et al. extended the definition of modularity to the overlapping setting [5] and their extended definition is denoted Q^E. Yang and Leskovec [39] observed that objective functions that are based on triadic closure—identification of nodes which close a triangle with other nodes in the graph, provides the best results when there is significant overlap between communities. Weighted Community Clustering (WCC) [29] is such an objective function, defined only for disjoint community structures.

Cohen et al. presented NECTAR [7], an overlapping community detection algorithm, that generalized Blondel et al.'s algorithm so it can be applied also to networks possessing overlapping community structure. They presented WOCC—Weighted Overlapping Community Clustering, a generalization of WCC to the overlapping case. In addition to WOCC, NECTAR also employed the Q^E objective function. A unique feature of NECTAR is that it chooses dynamically whether to use WOCC or Q^E, depending on the structure of the graph at hand. NECTAR's decision as to which of the above objective functions to choose is based solely on the average number of closed triangles per node in the graph. If this average is below a certain threshold, Q^E will be employed, otherwise WOCC will be used. This approach, as shown by Cohen et al. [7], provided good separation between communities with high overlap (on which WOCC, in most cases, is superior) and low overlap (on which extended modularity, in most cases, is superior). An extensive experimental evaluation was conducted, comparing NECTAR and six other state-of-the-art overlapping community detection algorithms. The evaluation was done using both synthetic and real-world networks with ground-truth. The evaluation of the clusterings output by the algorithms was made using the commonly-used metrics ONMI [21], Omega Index [8] and Average F1 score [38]. NECTAR outperformed all other algorithms in terms of average detection quality. It was ranked first (on average) for both synthetic and real-world networks, leading in 33 out of 96 of synthetic networks and was best for one real-world network and second-best for another.

Another approach for the problem of approximating the optimum of the objective function is by employing evolutionary algorithms (EAs) (see e.g. [6,11–14,20,25–27,30,32,33,35]). EAs are a class of optimization heuristics methods inspired by biological evolution. Candidate solutions to the optimization problem play the role of individuals in a population, and a fitness function is used to determine the quality of the solutions. A common approach is to consider the community detection problem as a single-objective optimization problem (see e.g. [13,26,32]). However, it is plausible to assume that a community should have dense intraconnections and sparse interconnections, implying that two objectives should be optimized simultaneously in community detection, i.e., maximizing internal links and minimizing external links [6,11,20]. Therefore, the community detection problem can also be modelled as a multi-objective optimization problem (MOP). Multi-objective EAs (MOEAs) are considered to be the most popular approach to solve MOP and indeed, several MOEAs [12,14,25,27,30] have been proposed for community detection.

A unique feature of such algorithms is that they find a set of optimal solutions instead of a single solution in one run, as each solution corresponds to a partition of the given network. Most of the existing works focus on developing MOEAs for detecting nonoverlapping communities, but there are also a few works (see e.g. [33,35]) that focus on detecting overlapping communities. Unfortunately, MOEAs' scalability on large-scale real-world networks is limited, causing them to be impractical for graphs containing over 10,000 nodes.

Among the few MOEAs which are able to detect overlapping communities, we focus on two leading algorithms: The Maximal Clique based Multi Objective Evolutionary Algorithm (MCMOEA), introduced by Wen et al. [35], and the Evolutionary Multiobjective Optimization based Fuzzy Method (EMOFM), introduced by Tian et al. While in [33] three approaches of EMOFM are presented, in this paper, we use its best approach in terms of performance and run time compared to the other two, denoted EMOFM-DK. DK stands for the use of Diffusion Kernel similarity for measuring the distance between nodes in the graph [18]. Both the MCMOEA and the EMOFM-DK algorithms use two exact metrics for quantitatively comparing the quality of overlapping communities obtained by different approaches on the benchmark networks, i.e., the extended modularity Q^E and the generalized normalized mutual information (gNMI) [21]. Similarly to ONMI, which is used as an evaluation criteria in our work, gNMI is also a version of Normalized Mutual Information (NMI), suitable for overlapping clustering evaluation.

We emphasize that the main difference between NECTAR's approach to that of MOEA algorithms is the fact that NECTAR is striving to select the best objective function among multiple objective functions for the optimization problem, while the MOEA algorithms are simultaneously optimizing multiple objective functions. Another important difference between the two approaches is that NECTAR was designed to handle large-scale networks, while MOEA algorithms are capable of processing networks comprised of at most a few thousand nodes.

3 The NECTAR-ML Algorithm

The high-level pseudo-code of the NECTAR-ML algorithm is given by Algorithm 1 with modified lines w.r.t. NECTAR coloured in light blue. As described in Sect. 2, NECTAR's decision as to which objective functions to choose, is based on the average number of closed triangles per node in the graph. NECTAR-ML optimizes this selection using a machine learning model (lines 5–7 in Algorithm 1). It extracts features from the graph G and provides them as input to the model, which then predicts which of the two objective functions is best to use.

`NECTAR` proceeds in iterations, in each of which, it iterates over all nodes $v \in V$ (in some random order), attempting to determine the set of communities to which node v belongs such that the objective function is maximized. Parameter $\beta \geq 1$ is used by NECTAR (as well as by NECTAR-ML) to determine the number of communities to which a node should belong in a dynamic manner. A detailed description of NECTAR can be found in the full version of this paper [3].

3.1 Learning a Model for Objective Function Selection

For a given graph, we can either use WOCC or Q^E as our objective function. Thus, we need to construct a binary ML classifier which, given a network, selects between the two. Towards this goal, we constructed a large labeled dataset of networks for the training and evaluation of our model. A detailed description of the data collection process is provided later. As for model evaluation, since our key goal is for the model to select an objective function that will optimize a specific quality metric, we selected to use the Accuracy and Recall scores per class as the performance evaluation metrics of the model. We consider several supervised ML algorithms for generating the model, including decision trees, deep learning and linear classifiers. We use the default threshold of 0.5 to select which objective function to use. Our training dataset is imbalanced. We therefore use the Balanced Accuracy metric, which is the average of TPR and TNR. In order to improve the models' accuracy, we tune the hyperparameters according to the average of the Balanced Accuracy metric over 5-folds of the data.

Data Generation. Lancichinetti, Fortunato and Radicchi et al. [22] introduced a set of benchmark graphs (named the LFR benchmark) that provide heterogeneity in terms of node degree and community size distributions, as well as control of the degree of overlap between ground truth communities. LFR is widely used in the field of overlapping community detection research (see e.g. [7,15,16,23,33,35–37]). We used LFR to generate a dataset of 15,755 networks, by using the following parameters: The number of nodes, n, is in the range 10K–65K; average node degree, k, is set to 10, 20, 40, 60 or 80; The number of overlapping nodes, O_n, is 10%, 25%, or 50% for networks with an average node degree of 10, 20 or 40 and 75% for networks with an average node degree of 60 or 80. The number of communities an overlapping node belongs to, O_m, is set to $\{2, \ldots, 10\}$ (networks with low average degree are parameterized only with the lower portion of this range). The mixing parameter for the topology, mut, is set to values from $\{0.1, \ldots, 0.5\}$ (networks with low average degree are parameterized only with the lower portion of this range). Maximum node degree $maxK$ is set to 50 for networks with low average degree and to 100 or 120 for networks with higher degrees. The full list of parameter values can be found in the full version of this paper [3]. For each combination of parameters we generated two distinct graph instances. The features we extracted for classification are:

1. GCC (Global clustering coefficient): $\frac{3 \times NumberOfTriangles}{NumberOfTriplets}$, where a triplet consists of three nodes that are connected by either two (open triplet) or three (closed triplet) undirected edges. The expression $3 \times NumberOfTriangles$ can also be referred as to the total number of closed triplets in the graph, as one triangle contains three closed triplets.
2. ACC (Average clustering coefficient): $\frac{\sum_{u \in G} C_u}{NumberOfNodes}$, where Cu is defined as
$$Cu = \begin{cases} \frac{2 \times |E_{v,w}|}{k_u \times (k_u - 1)}, & \text{if } k_u > 1 \\ 0, & \text{otherwise,} \end{cases}.$$
where $E_{v,w}$ is the set of edges among node u's neighbours and k_u is the degree of u
3. $RatioOfNodesInTriangle$: $\frac{NumberOfNodesInTriangles}{NumberOfNodes}$.
4. $AverageNodeDegree$: $\frac{2 \times NumberOfEdges}{NumberOfNodes}$.
5. $AverageTrianglesRate$: $\frac{NumberOfTriangles}{NumberOfNodes}$.

As for the labels, we invoked NECTAR on each network, once using the WOCC objective function and once using the Q^E objective function with 10 different values of $\beta \in \{1.01, 1.05, 1.09, 1.1, 1.2, 1.3, 1.4, 1.6, 1.8, 2.0\}$. Using the ground truth of the those networks, we evaluated NECTAR's outputs per network and β value in terms of ONMI [21], Omega Index [8] and Average F1 score [38] which served as the evaluation criteria in [7] for the NECTAR algorithm. A full description of those metrics can be found in the full version of this paper [3].

We then label the networks as follows. For each metric type and network, the objective function which provides the best metric score, over all β values, is selected as the label of that network for that metric. In addition, a network is also labeled, in the same manner, according to the average of the three metrics. Therefore, four different labels are generated (one per metric and a 4th for their average scores). It is possible that for some metric, the gap in scores between the result of NECTAR, when invoked using one objective function to the other, is very small. We would like the model to reflect the fact that, in these cases, it is not that important which objective function is selected. Consequently, we weighted each of the networks according to the gap between the two results, according to Eq. (1):

$$Weight(G, m) = \frac{|WOCCScore_m(G) - ModScore_m(G)|}{\max(WOCCScore_m(G), ModScore_m(G))}, \tag{1}$$

where G denotes the network and m denotes the metric type. $WOCCScore_m(G)$ is the best metric score over all β values for the network, using the WOCC objective function. $ModScore_m(G)$ is the equivalent term for the Q^E objective function case. The labels distribution, with and without consideration of the networks weight, for a total of 15,755 synthetic networks indicates that all metric types present a significant class imbalance (Approximately 80% labeled as Q^E and 20% as WOCC). The full distribution data can be found in the full version of this paper [3].

Training and Evaluation. Synthetic Networks: We generated NECTAR-ML models and evaluated each of them according to the following metrics: ONMI [21], Omega Index [8], Average F1 score [38] and their average value. All comprised datasets, one per metric, were partitioned into a training set and a test set in the same manner: Networks containing 10K–50K vertices, which constitute 75% of the dataset (11,822/15,755), were used as the training set while networks containing 55K-65K vertices which constitute 25% of the dataset (3,933/15,755), were used as the test set. The purpose of this setting is to validate that the models are robust, in terms of their ability to scale to the size of the network, as large scale networks share similar attributes, regardless of their size. This will imply that the models possess the ability to infer, with high accuracy, the suitable objective function for networks that are larger than those used for their training. Oversampling is used in training to handle data imbalance.

Algorithm 1 NECTAR-ML algorithm pseudo-code.

```
1  const maxIter ← 20                                    /* max iterations */
2  const $\alpha$ ← 0.8                                  /* merge threshold */
3  Procedure NECTAR-ML($G$=<$V$,$E$>, $\beta$){
4  /* Extract features from Graph                                        */
5  $features$ ← ExtractFeatures(G)
   /* Predict obj. function                                              */
6  $objFunc$ ← model.predict($features$)
7  use $objFunc$ as the objective function
8  Initialize communities
9  $i \leftarrow 0$                              /* number of extern. iterations */
10 repeat
11     $s \leftarrow 0$                                /* number of stable nodes */
12     forall $v \in V$ do
13         $C_v \leftarrow$ communities to which $v$ belongs
14         Remove $v$ from all the communities of $C_v$
15         $S_v \leftarrow \{C \in \mathcal{C} \mid \exists u : u \in C \wedge (v,u) \in E\}$
16         $D \leftarrow \{\Delta(v,C) \mid C \in S_v\}$
17         $C'_v \leftarrow \{C \in S_v \mid \Delta(v,C) \cdot \beta \geq max(D)\}$
18         Add $v$ to all the communities of $C'_v$
19         if $C'_v = C_v$ then
20             $s$++
21     end
22     merge($\alpha$)                                 /* merge communities */
23     if merge reduced number of communities then
24         $s \leftarrow 0$
25     $i$++
26 until $(s = |V|) \vee (i = maxIter)$
```

Table 1. Performance of selected ML algorithms in terms of balanced accuracy (BA) on synthetic networks dataset. The 5th column presents averaged BA results for 5-folds cross-validation runs. The remaining columns present the BA and recall per class over the test-set.

Metric	Best algorithm	Hyperparameter type	Opt. value	BA 5-folds	BA test set	Recall - Q^E test set	Recall - WOCC test set
ONMI	*GBDT*	Number of estimators	400	0.880	0.881	0.806	0.956
		Max. depth	4				
		Min. samples split	3				
		Min. samples leaf	2				
		Learning rate	0.1				
Average F1	*GBDT*	Number of estimators	400	0.884	0.883	0.838	0.928
		Max. depth	4				
		Min. samples split	3				
		Min. samples leaf	2				
		Learning rate	0.1				
Omega-Index	*ExtraTrees*	Number of estimators	300	0.905	0.907	0.842	0.971
		Max. depth	25				
		Min. samples split	2				
		Min. samples leaf	2				
Metrics-Average	*ExtraTrees*	Number of estimators	300	0.901	0.899	0.844	0.954
		Max. depth	40				
		Min. samples split	2				
		Min. samples leaf	2				
Overall average				**0.893**	**0.893**	**0.833**	**0.952**

We considered several supervised ML algorithms and hyperparameters for all models (The full list of algorithms can be found in the full version of this paper [3]). To select the best hyper-params, we use 5-fold cross validation, keeping the ratio between classes equal in each fold. Each trained classifier C_m was applied to the corresponding test set fold. Table 1 summarizes the performance of the models in the setting described above. It presents the ML algorithms and hyperparameters values which obtained the best balanced accuracy results per metric. The balanced average achieved over the validation and test sets for all metrics is close to 90% with a low std of less than 0.01. This indicates that the models are not overfitted. In addition, the recall for both WOCC and Q^E is high.

Real World Networks: To validate the quality of our model, we evaluated it on real world networks from the Stanford Large Network Dataset Collection [24]. Five undirected, unweighted networks from three different domains were considered:

1. The **Co-product purchasing network (Amazon).**
2. The **Co-publishing network (DBLP).**
3. 3 **Social networks**: LiveJournal, Friendster and Youtube.

A summary of the properties of these graphs is presented in Table 2.

Similarly to the synthetic networks, the labels of the real-world networks were assigned by applying NECTAR to each real-world network, applying both the WOCC and Q^E objective functions with the 10 different values of β. Then, each network is labeled for the best objective function per metric. Table 3 presents

Table 2. Properties of real-world networks used.

Sub-graphs			
Network	Number of vertices	Number of edges	Number of clusters
Amazon	16,716	48,739	1,517
DBLP	93,432	335,520	4,961
Youtube	39,841	224,235	4,771
LiveJournal	84,438	1,521,988	4,703
Friendster	220,015	4,031,793	4,914
Full graphs			
DBLP	317,080	1,049,866	13,477
Amazon	334,863	925,872	75,149

Table 3. Ground truth labels and weights for real world networks. For each real world network, the best objective function and network weight are presented.

Metric	ONMI		Average F1		Omega-Index		Metrics average	
Sub-Graphs	Best	Weight	Best	Weight	Best	Weight	Best	Weight
Amazon	Q^E	0.074	Q^E	0.050	Q^E	0.273	Q^E	0.100
Youtube	Q^E	0.235	Q^E	0.171	Q^E	0.401	Q^E	0.192
DBLP	Q^E	0.012	Q^E	0.008	Q^E	0.226	Q^E	0.045
LiveJournal	WOCC	0.038	WOCC	0.020	Q^E	0.619	Q^E	0.032
Friendster	WOCC	0.003	WOCC	0.0002	Q^E	0.204	Q^E	0.056
Full graphs								
DBLP	Q^E	0.071	Q^E	0.014	Q^E	0.695	Q^E	0.027
Amazon	Q^E	0.127	Q^E	0.075	Q^E	0.259	Q^E	0.128

the labels for each metric and real-world network, including the network weight according to Eq. (1). It can be seen that the extended Modularity objective function is the preferable choice in 85% of the cases (24/28) as the models for the ONMI and Average F1 metrics contain WOCC labels for the LiveJournal and Friendster networks while the other metrics models contain only Q^E labels.

Table 3 reveals the motivation behind splitting the analysis per metric, as some users might have different needs in evaluating the clustering results. For instance, from Table 3, a user who chooses to use the Metrics Average model with NECTAR-ML for the Friendster or LiveJournal social networks will indeed receive better results for this metric, but if the user is only interested in the ONMI or Average F1 metric scores, it would be best to use the ONMI or Average F1 models for that purpose.

In order to evaluate the models' performance on the real world networks, each classifier C_m was trained using the selected hyperparameters on the whole synthetic networks dataset (10K–65K) and then applied on the real-world networks. Figure 1 presents the weighted hit/miss results of the predictions of all classifiers. The values in the heatmap represent networks weight and signify the level of importance in selecting the objective function. A hit is represented by a positive

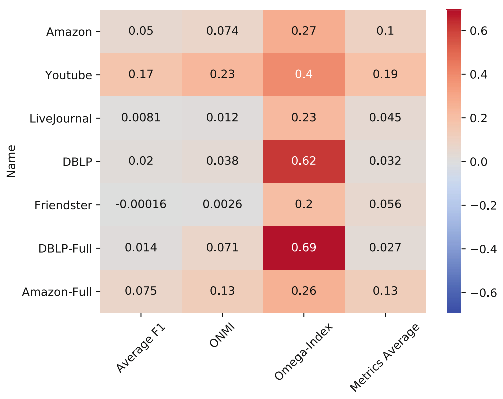


Fig. 1. A heat-map of the weighted hit/miss results on real world networks using the classifiers per metric.

value of the network weight and therefore coloured in red. A miss is represented by a negative value of the network weight and therefore coloured in blue. Weights close to zero are coloured in grey. By examining the weights values, we can see that the classifiers presented a high level of accuracy on both low and high weight values. It can also be seen that there is a single miss, by the Average F1 classifier, whose significance is very low (the weighted value is practically zero). Table 4 summarizes the averaged results of the experiments conducted on the real-networks. The network weights are calculated using Eq. (1) and presented in Table 3. The 2nd column presents the averaged balanced accuracy results per metric while the remaining two columns present the averaged recall results for each of the objective functions. Recall values for WOCC are missing for the Omega-Index and Metrics-Average since no WOCC ground truth labels exist for those metrics. The balanced accuracy weighted average of all metrics reaches nearly 100%.

4 Experimental Evaluation

This section includes a comparison of NECTAR-ML to NECTAR and MOEA algorithms on synthetic and real world networks.

Table 4. Averaged test set results for real world networks with respect to the networks weights.

Metric	Balanced accuracy	Recall Q^E	Recall WOCC
ONMI	1.000	1.000	1.000
Average F1	0.990	1.000	0.980
Omega-Index	1.000	1.000	—
Metrics average	1.000	1.000	—
Weighted average	**0.999**	**1.000**	**0.999**

4.1 Competitive Analysis to NECTAR

In this section we present a competitive analysis between NECTAR-ML and NECTAR. We analyse the performance of the two methods on the test sets of both synthetic and real world networks, presented in Sect. 26. We compare the accuracy provided by each algorithm in selecting the best objective function for each of the test set networks. The specific model, per metric type, was applied on the test set networks and its objective function predictions are compared to NECTAR's dynamic selections according to the $AverageTriangleRate$ threshold value.

Synthetic Networks: Figure 2 presents the competitive analysis of Cohen et al.'s NECTAR[7] and NECTAR-ML algorithms using the average metrics for labelling. Competitive analysis figures using ONMI, Omega-index and average F1 metrics follow similar trends and are presented in the full version of this paper [3]. In the figure, each cell in the heatmap represents a certain LFR configuration which is comprised out of four variables, k O_n, O_m and mut. The networks which match the cell's configuration are given a calculated weight according to Eq. (1). The value of the cell is calculated as follows. First, a total sum variable is initialized to 0. Then, for each network, If NECTAR-ML selects the objective function correctly while NECTAR does not, the calculated weight value of the network is added to the total sum. If NECTAR selects the objective function correctly while NECTAR-ML does not, the calculated weight value is subtracted from the sum. If both algorithms are wrong/correct, no value is added nor subtracted from/to the sum. Since there are multiple networks per configuration, in order to normalise the value between -1 to 1, the summed value is divided by the number of networks which comprised the sum. The cell's color and intensity, with respect to the cell's value, are defined by the colour palette which is placed at the right of the heatmap. Therefore, a red colour indicates that for a specific configuration, NECTAR-ML provided better selections than NECTAR while a blue colour indicates the opposite. A grey colour indicates that both algorithms provided similar results. It can be seen that, except for only a few minor cases, NECTAR-ML is on average, equal to or better than the NECTAR algorithm. NECTAR-ML's better selections of the objective functions are more noticeable for the configurations containing O_n values equal to 50% and 75% of the number of total nodes. In addition, we can see that this pattern is being strengthened with the increase of k and O_m values, meaning that NECTAR-ML presents supremacy over NECTAR especially in dense networks, as it can be seen from the bottom table in the figure.

Real-World Networks: Figure 3 presents the competitive analysis of the NECTAR [7] and NECTAR-ML algorithms, with respect to all metric types. As in Fig. 2, the colour of the cell, represent a hit/miss of NECTAR-ML vs NECTAR while its opacity is controlled by the network's weight. The Amazon and Amazon-Full networks shared the same algorithms selections for all metric cases and are therefore omitted from the heatmap. It can be seen that NECTAR-ML's selections are equal or better than NECTAR, apart from one selection

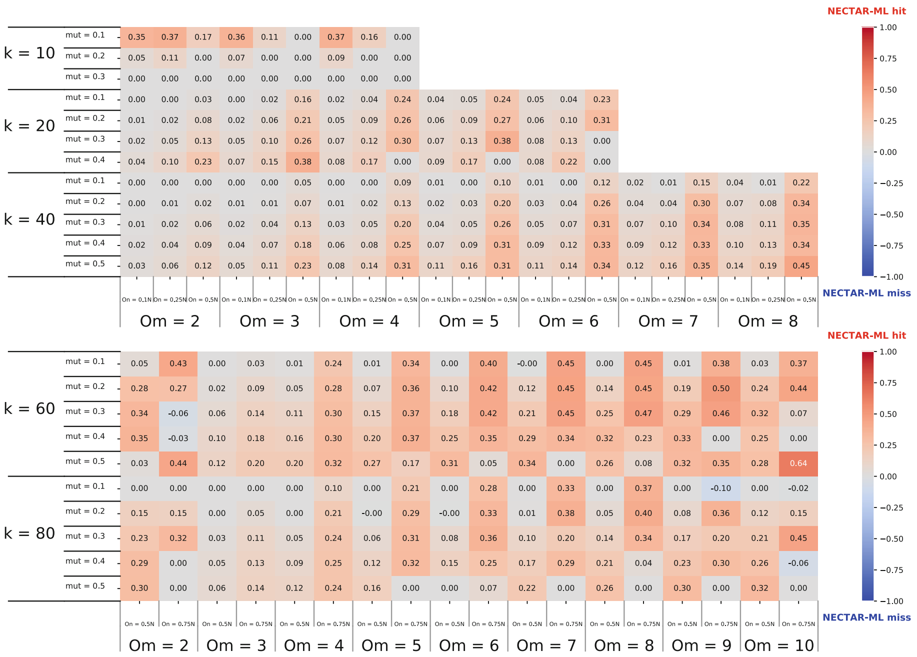


Fig. 2. NECTAR vs NECTAR-ML competitive analysis on synthetic networks using the average metrics model. The comparison was made using the 55K–65K test set networks.

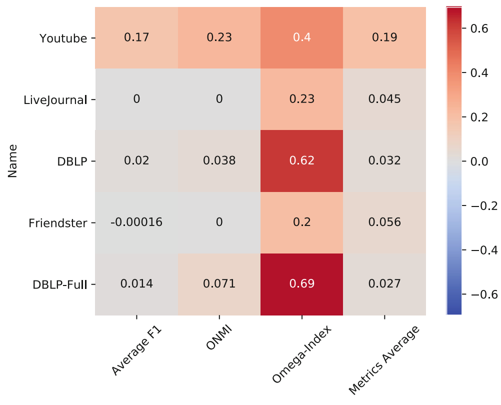


Fig. 3. NECTAR vs NECTAR-ML competitive analysis on real world networks per metric.

whose importance is very low, as the weighted value of this network is practically zero (Average F1 metric over the Friendster network). NECTAR-ML's superior objective function selections are reflected both on low and high weights values of the networks and therefore positions NECTAR-ML as significantly superior to NECTAR over real-world networks, for all four metrics.

4.2 Competitive Analysis to MOEA Algorithms

In this section we analyse and describe the results of an extensive competitive analysis we have conducted between NECTAR-ML and the two MOEA algorithms presented in Sect. 2. First, all three algorithms are compared using the same benchmark of networks due to the network size limitation of the EMOFM-DK algorithm. Then, NECTAR-ML is compared to the MCMOEA algorithm over large scale networks.

NECTAR-ML vs MOEAs: To evaluate correctly EA algorithms, a suitable benchmark was selected, which reflects a common ground of all algorithms. For this, the following considerations were taken into account. First, due to the runtime limitation of EMOFM-DK, the number of total nodes selected for the comparison was set to 1000. This network size is much smaller than the large scale networks we used for evaluating NECTAR-ML, but it enabled us to compare the algorithms properly. Second, the evaluation of the clusterings output by the algorithms was made using the commonly-used metrics, ONMI [21], Omega Index [8] and Average F1 score [38]. Since ONMI and gNMI are versions of NMI and both are suitable for overlapping clustering evaluation, as stated in Sect. 2, we decided to continue to use the ONMI metric, in order to use it consistently all throughout this work. Third, evolutionary multiobjective optimization based approaches can obtain multiple solutions of overlapping community structures in a single run, while NECTAR-ML is producing only a single solution per objective function and β value (10 values of β are used). To mitigate this gap, we decided to compare MOEAs best score per metric, selected from the entire set of runs and clustering outputs to NECTAR-ML's best score per metric, retrieved using the predicted objective function over the β values. Finally, the number of runs for a MOEA algorithm to be executed is set to 30, which is the number of runs used by Wen et al. and Tian et al. in their competitive analysis. Both MOEA algorithms run with their default parameters values. We created a dataset of 785 synthetic networks for the competitive analysis. Specifically, the LFR parameters used by [36] can be found in the full version of this paper [3]. The parameters, which were used in both MCMOEA [35] and EMOFM-DK [33] competitive analysis to other algorithms, are contained in this set of parameters. Due to the fact that NECTAR-ML's learning based model was designed for large scale networks and the analysis of significantly smaller ones present a different view on the classes and features distributions, the model was re-trained using smaller network sizes. The network sizes which were selected for the training phase were 100, 300, 500, 1400, 1600, 1800 and 2000. All networks sizes share the same configurations as the 1000 network size configurations. The labels distribution, for a total of 5,918 synthetic networks, with and without consideration of the networks weight, can be found in the full version of this paper [3].

Table 5 presents the results of the model evaluation on the test data for all metric types. The 1st column presents the balanced accuracy results for the 5-folds cross-validation runs. The maximum standard deviation for the folds is less than 0.014. The remaining columns present the results for the networks test set. The 2nd column present the balanced accuracy result and the last two columns

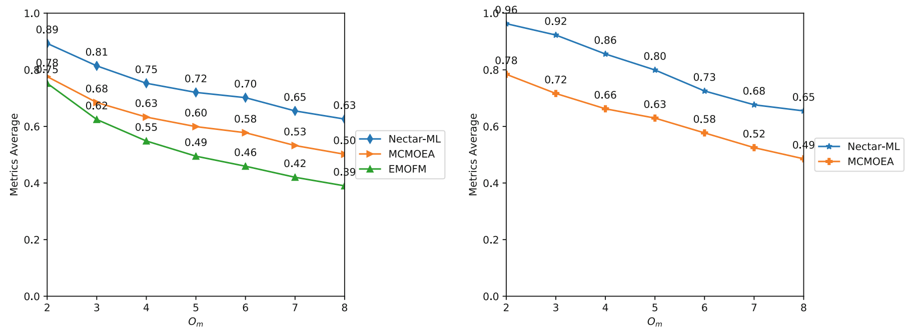
Table 5. Synthetic networks results for the classifiers on networks of size 1000.

Metric	Balanced accuracy 5-Folds	Balanced accuracy test set	Recall - Q^E test set	Recall - WOCC test set
ONMI	0.887	0.920	0.928	0.911
Average F1	0.922	0.924	0.936	0.912
Omega-Index	0.876	0.897	0.910	0.883
Metrics average	0.917	0.933	0.935	0.931
Overall average	**0.901**	**0.919**	**0.927**	**0.909**

present the recall results for each of the objective functions. All models are using the default threshold value of 0.5 to select between the objective functions. As can be seen, the balanced average achieved over the test set of the networks of size 1000 is close to 92%. Also, the 5-folds cross validation results present similar results and have a low standard deviation of less than 0.014 for all metrics. This indicates that the models are not overfitted.

Using our dedicated framework, we were able to process, for the EMOFM-DK and MCMOEA algorithms, approximately 9,891,000 and 1,177,500 clustering outputs respectively, as there were 785 compared networks, 30 runs per network, 420 clustering outputs for EMOFM-DK, and 50 clustering outputs for MCMOEA per run. A timeframe of 48 h was given for the MOEA algorithms to complete their runs per network. Figure 4a presents the average performance of the algorithms in terms of average value of the three known metrics for all 785 networks. As can be seen in Fig. 4a, NECTAR-ML outperformed both algorithms, while MCMOEA is the second-best, with a score lower than NECTAR-ML's by approximately 17%. EMOFM-DK comes last with a score lower than NECTAR-ML's by approximately 30%. Competitive analysis figures related to the ONMI, Omega-index and average F1 metrics follow the same trend and can be found in the full version of this paper [3].

NECTAR-ML vs. MCMOEA: We created a dataset of 195 synthetic networks for the large-scale networks competitive analysis. The selected LFR parameters extend the networks configuration used in MCMOEA [35], for the performance analysis of large-scale networks, as follows: The total number of nodes, n, is 10K, the average node degree, k, is set to 20. The number of overlapping nodes, O_n, is set to 10%, 25%, 50%. The number of communities an overlapping node belongs to, O_m, is set to $\{2, \ldots, 8\}$. The mixing parameter for the topology, mut, is set to values from the range $\{0.1, \ldots, 0.5\}$. The maximum node degree $maxK$ is set to 50. Using the same platform used for the 1K competitive analysis, we were able to produce MCMOEA clustering outputs for the dataset. As for NECTAR-ML, since this dataset contains network configurations which are contained in the training set, the model was re-trained with these networks excluded. Then, the model was applied on the dataset. Figure 4b presents the average performance of the algorithms in terms of average value of the three known metrics for all 195 networks.



(a) Obtained by the three approaches on 785 networks

(b) Obtained by NECTAR-ML and MCMOEA algorithms on 195 large-scale networks

Fig. 4. Average performance of the average metrics value as a function of O_m.

As can be seen in Fig. 4b, NECTAR-ML outperformed MCMOEA, which obtained an averaged score lower than NECTAR-ML's by approximately 22%. Competitive analysis figures related to The ONMI, Omega-index and average F1 metrics follow the same trend and can be found in the full version of this paper [3].

5 Conclusion

We introduced NECTAR-ML, an extension of the NECTAR algorithm that uses a machine-learning based model for automating the selection of the objective function by leveraging features based on the input graph. To train the model, we created a dataset of 15,755 synthetic networks, with various sizes and properties. We analyzed 3,933 synthetic and 7 real networks, measuring the quality of our models, aiming to dynamically select, based on the properties of the graph at hand, which objective function should be used. Our analysis shows that in approximately 90% of the cases our model was able to successfully select the correct objective function to maximize the desired metric.

NECTAR-ML was proven to be superior over NECTAR as it significantly outperformed NECTAR's ability to select the best objective function out of Q^E and WOCC. In addition, we have conducted an extensive competitive analysis of NECTAR-ML and two additional state-of-the-art multi-objective algorithms based on the MOEA approach for the multiobjective optimization problem (MOP). NECTAR-ML outperformed both algorithms in terms of average detection quality. Multiobjective EAs (MOEAs) are considered to be the most popular approach to solve MOP and the fact that NECTAR-ML significantly outperforms them demonstrates the effectiveness of ML-based objective function selection.

This work can be extended in several ways. First, by using multi-class rather than binary classification, NECTAR-ML can be trained to select from a broader set of objective functions, possibly based on additional structural graph properties. As proposed in [7] for the NECTAR algorithm, search heuristics that target some weighted average of several objective functions, rather than selecting just one of them, might further improve performance.

Also, despite the fact that most community detection algorithms require at least one user-provided parameter, decreasing their number makes it easier to use the algorithm. In future work, we will attempt to do so by learning them from the input graph's structural features as well. For instance, predicting the optimal value for the β parameter could be addressed as a regression problem.

References

1. Ahn, Y.Y., Bagrow, J.P., Lehmann, S.: Link communities reveal multiscale complexity in networks. Nature **466**(7307), 761–764 (2010). https://doi.org/10.1038/nature09182
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008). https://doi.org/10.1088/1742-5468/2008/10/p10008
3. Bornstein, A., Rubin, A., Hendler, D.: Machine-learning based objective function selection for community detection (2022). https://doi.org/10.48550/ARXIV.2203.13495. https://arxiv.org/abs/2203.13495
4. Brandes, U., et al.: On finding graph clusterings with maximum modularity. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 121–132. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74839-7_12
5. Chen, M., Kuzmin, K., Szymanski, B.K.: Extension of modularity density for overlapping community structure. In: 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014), pp. 856–863 (2014). https://doi.org/10.1109/ASONAM.2014.6921686
6. Clauset, A., Moore, C., Newman, M.E.J.: Hierarchical structure and the prediction of missing links in networks. Nature **453**(7191), 98–101 (2008). https://doi.org/10.1038/nature06830
7. Cohen, Y., Hendler, D., Rubin, A.: Node-centric detection of overlapping communities in social networks. In: Shmueli, E., Barzel, B., Puzis, R. (eds.) NetSci-X 2017. SPC, pp. 1–10. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55471-6_1
8. Collins, L.M., Dent, C.W.: Omega: a general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. Multivar. Behav. Res. **23**(2), 231–242 (1988). https://doi.org/10.1207/s15327906mbr2302_6. pMID: 26764947
9. Flake, G., Lawrence, S., Giles, C., Coetzee, F.: Self-organization and identification of web communities. Computer **35**(3), 66–70 (2002). https://doi.org/10.1109/2.989932
10. Gao, Y., Zhang, H., Zhang, Y.: Overlapping community detection based on conductance optimization in large-scale networks. Phys. A Stat. Mech. Appl. **522**, 69–79 (2019). https://doi.org/10.1016/j.physa.2019.01.142. https://www.sciencedirect.com/science/article/pii/S0378437119301487

11. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002). https://doi.org/10.1073/pnas.122653799. https://www.pnas.org/ content/99/12/7821
12. Gong, M., Cai, Q., Chen, X., Ma, L.: Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition. IEEE Trans. Evol. Comput. **18**(1), 82–97 (2014). https://doi.org/10.1109/TEVC.2013.2260862
13. Gong, M., Fu, B., Jiao, L., Du, H.: Memetic algorithm for community detection in networks. Phys. Rev. E **84**, 056101 (2011). https://doi.org/10.1103/PhysRevE.84.056101. https://link.aps.org/doi/ 10.1103/PhysRevE.84.056101
14. Gong, M., Ma, L., Zhang, Q., Jiao, L.: Community detection in networks by using multiobjective evolutionary algorithm with decomposition. Phys. A Stat. Mech. Appl. **391**(15), 4050–4060 (2012). https://doi.org/10.1016/j.physa.2012.03.021. https://www.sciencedirect.com/science/article/pii/S0378437112002579
15. Gregory, S.: Finding overlapping communities in networks by label propagation. New J. Phys. **12**(10), 103018 (2010). https://doi.org/10.1088/1367-2630/12/10/103018
16. Gregory, S.: Fuzzy overlapping communities in networks. J. Stat. Mech. Theory Exp **2011**(02), P02017 (2011). https://doi.org/10.1088/1742-5468/2011/02/p02017
17. King, A.D., Pržulj, N., Jurisica, I.: Protein complex prediction via cost-based clustering. Bioinformatics **20**(17), 3013–3020 (2004). https://doi.org/10.1093/bioinformatics/bth351
18. Kondor, R.I., Lafferty, J.: Diffusion kernels on graphs and other discrete structures. In: Proceedings of the ICML, pp. 315–322 (2002)
19. Krogan, N.J., et al.: Global landscape of protein complexes in the yeast saccharomyces cerevisiae. Nature **440**(7084), 637–643 (2006)
20. Lancichinetti, A., Fortunato, S.: Community detection algorithms: a comparative analysis. Phys. Rev. E **80**, 056117 (2009). https://doi.org/10.1103/PhysRevE.80.056117. https://link.aps.org/doi/10.1103/PhysRevE.80.056117
21. Lancichinetti, A., Fortunato, S., Kertész, J.: Detecting the overlapping and hierarchical community structure in complex networks. New J. Phys. **11**(3), 033015 (2009). https://doi.org/10.1088/1367-2630/11/3/033015
22. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**, 046110 (2008). https://doi.org/10.1103/PhysRevE.78.046110. https://link.aps.org/doi/10.1103/PhysRevE.78.046110
23. Lee, C., Reid, F., McDaid, A., Hurley, N.: Detecting highly overlapping community structure by greedy clique expansion (2010)
24. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection, June 2014. http://snap.stanford.edu/data
25. Liu, C., Liu, J., Jiang, Z.: A multiobjective evolutionary algorithm based on similarity for community detection from signed social networks. IEEE Trans. Cybernet. **44**(12), 2274–2287 (2014). https://doi.org/10.1109/TCYB.2014.2305974
26. Pizzuti, C.: GA-Net: a genetic algorithm for community detection in social networks. In: Rudolph, G., Jansen, T., Beume, N., Lucas, S., Poloni, C. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 1081–1090. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87700-4_107
27. Pizzuti, C.: A multiobjective genetic algorithm to find communities in complex networks. IEEE Trans. Evol. Comput. **16**(3), 418–430 (2012). https://doi.org/10.1109/TEVC.2011.2161090

28. Pizzuti, C., Rombo, S.E.: Algorithms and tools for protein-protein interaction networks clustering, with a special focus on population-based stochastic methods. Bioinformatics **30**(10), 1343–1352 (2014). https://doi.org/10.1093/bioinformatics/btu034
29. Prat-Pérez, A., Dominguez-Sal, D., Brunat, J.M., Larriba-Pey, J.L.: Shaping communities out of triangles. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM 2012, pp. 1677–1681. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2396761.2398496
30. Shi, C., Yan, Z., Cai, Y., Wu, B.: Multi-objective community detection in complex networks. Appl. Soft Comput. **12**(2), 850–859 (2012). https://doi.org/10.1016/j.asoc.2011.10.005. https://www.sciencedirect.com/science/article/pii/S1568494611003991
31. Šíma, J., Schaeffer, S.E.: On the NP-completeness of some graph cluster measures. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 530–537. Springer, Heidelberg (2006). https://doi.org/10.1007/11611257_51
32. Tasgin, M., Herdagdelen, A., Bingol, H.: Community detection in complex networks using genetic algorithms (2007)
33. Tian, Y., Yang, S., Zhang, X.: An evolutionary multiobjective optimization based fuzzy method for overlapping community detection. IEEE Trans. Fuzzy Syst. **28**(11), 2841–2855 (2020). https://doi.org/10.1109/TFUZZ.2019.2945241
34. Viamontes Esquivel, A., Rosvall, M.: Compression of flow can reveal overlapping-module organization in networks. Phys. Rev. X **1**, 021025 (2011). https://doi.org/10.1103/PhysRevX.1.021025. https://link.aps.org/doi/10.1103/PhysRevX.1.021025
35. Wen, X., et al.: A maximal clique based multiobjective evolutionary algorithm for overlapping community detection. IEEE Trans. Evol. Comput. **21**(3), 363–377 (2017). https://doi.org/10.1109/TEVC.2016.2605501
36. Xie, J., Kelley, S., Szymanski, B.K.: Overlapping community detection in networks: the state-of-the-art and comparative study. ACM Comput. Surv. **45**(4) (2013). https://doi.org/10.1145/2501654.2501657
37. Xie, J., Szymanski, B.K.: Towards linear time overlapping community detection in social networks. In: Tan, P.-N., Chawla, S., Ho, C.K., Bailey, J. (eds.) PAKDD 2012. LNCS (LNAI), vol. 7302, pp. 25–36. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30220-6_3
38. Yang, J., Leskovec, J.: Community-affiliation graph model for overlapping network community detection. In: 2012 IEEE 12th International Conference on Data Mining, pp. 1170–1175 (2012). https://doi.org/10.1109/ICDM.2012.139
39. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2015). https://doi.org/10.1007/s10115-013-0693-z10.1007/s10115-013-0693-z

Randomness for Randomness Testing

Daniel Berend, Shlomi Dolev, and Manish Kumar(✉)

Ben-Gurion University of the Negev, Be'er Sheva, Israel
berend@bgu.ac.il, dolev@cs.bgu.ac.il, manishk@post.bgu.ac.il

Abstract. Given a binary sequence, one may inquire whether it is produced by a true random source. There are several tests designed to answer this question, such as the statistical test suite of the National Institute of Standard and Technology (NIST) and the Diehard tests.

The problem is that, given deterministic tests of randomization, an adversary may know/learn, the adversary may tailor a non-random (deterministic) sequence, guided by the deterministic tests, that passes the tests.

We suggest to use a true random source for randomness tests and thus make the tests significantly harder to being misled. We design tests that use true random sources and demonstrate their ability to detect non-random sequences that NIST classifies as random.

Keywords: Property testing · Randomness testing · Truly random generator

1 Introduction

The need for true randomness in cryptography and randomized algorithms is inherent to the validity of the results. One may ask whether the used randomness is based on a truly random source or (a malicious) non-random source. Verifying that a given sequence is produced by a truly random source is impossible for any bounded length sequence, as there is an automaton (with the number of states proportional to the sequence length) that produces the sequence. Still, one can examine the sequence according to statistical tests.

What is a truly (uniformly) random sequence? Out of the three sequences 0000000000, 0101010101, and 0111001001, of length 10, most people would probably classify the first two as non-random, but agree to pass the third as random. Note that a truly random generator (TRG), asked to provide a random sequence of length 10, has the same probability of $1/2^{10}$ of producing each of the sequences above. So why is the third sequence "more random" than the first two?

D. Berend—Research supported in part by the Milken Families Foundation Chair in Mathematics and the Cyber Security Research Center at Ben-Gurion University.

S. Dolev—This research was (partially) funded by a grant from the Ministry of Science and Technology, Israel & the Japan Science and Technology Agency (JST), the German Research Funding (DFG, Grant #8767581199), Genesis Consortium, the Rita Altura trust chair in computer science, and by the Lynne and William Frankel Center for Computer Science.

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 153–161, 2022.
https://doi.org/10.1007/978-3-031-07689-3_11

Similar questions are relevant in the context of gambling. A casino will have a hard time convincing a judge that a roulette, that has shown red for an entire day, is honest, even though this sequence of results is just as likely as any other sequence. One expects that, if a (long) sequence of length n is produced by a TRG, each sequence of some fixed length $k << n$ will show up with a frequency of about $1/2^k$ among the sub-blocks of sequence.

A judge may decide to base a test of the distribution of the sub-blocks of some length k within the considered sequence. The problem is that, if the test is known in advance (as is the case with NIST's tests), an adversary may use this knowledge to design structured sequences that will pass it. To overcome this problem, we suggest using a non-deterministic judge, who will use many random tests. If the sequence fails in at least one of these tests, we decide that it was not generated by a TRG.

The suggestion of testing randomness with randomness seems circular. How do we verify that a supposedly random source we use is indeed random? Fortunately, there are several suppliers of true random devices based on quantum (or other similar physical) phenomena [22]. We may bit-wise XOR several sequences produced by such sources to obtain a more trusted true random sequence. Even if but one of the sources is indeed truly random, the resulting sequence is also such. Once we have (typically expensive) truly random devices, we can test a supplier of a cheaper device/procedure for the (level of) randomness claimed, prior to approving the new device for use.

Throughout the paper, a *truly random source* is a source generating binary sequences of an arbitrary length, such that (i) each bit has the same probabilities 1/2 and 1/2 of being 0 or 1, and (ii) distinct bits are statistically independent. When discussing a random sequence, we refer to a sequence generated by a truly random source. If we want to refer to a sequence generated by some source (random or not), which does not satisfy these requirements, we will emphasize this fact.

Organization of the Paper. In Sect. 2 we present some related work. Section 3 introduces NIST's tests. Section 4 is devoted to the idea of the tests proposed in this paper. In Sect. 5 we detail the sequences on which we have tested both NIST's tests and ours. Section 6 presents and discusses the results of the evaluation.

2 Related Work

Statistical Test Suite. Randomness tests are used to analyze the distribution pattern of a set of data. Researchers proposed a variety of algorithms for randomness testing. In 1938, Kendall and Smith [11] introduced the first randomness test, based on *Pearson's chi-squared test.* In 1995, Marsaglia introduced the *Diehard tests* for checking the quality of a random number generator and applied it to some data. Nowadays, the standard tests in use are published by the National Institute of Standards and Technology (NIST); their set of tests was published in 2001.

The NIST Test Suite [17] is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrarily long) binary sequences, produced by either hardware or software-based cryptographic random or Pseudo Random Number Generators (PRNG). (In fact, there were initially 16 tests, but one of them – Lempel-Ziv's test – was later removed due to implementation problems). These tests are widely used in many applications. For example, in the evaluation of the Advanced Encryption Standard (AES) candidate algorithms, one of the criteria was their performance as PRNGs. The NIST test suite was used in order to determine the number of rounds in which the algorithms behave like a PRNG, to understand the security level of the algorithms [20].

All of the NIST's tests are parameterized by a parameter n, denoting the length (in bits) of the processed bitstream. Some of the tests use an additional parameter M. In these tests, the given bitstream is divided into n/M chunks of equal length M each, and the distribution of some specific feature in these parts is examined.

Property Testing. Property testing, as introduced in the seminal works [8,16], deals with algorithms for deciding whether a given object has a pre-determined property or is *far* from every object having this property. Property testing has been used for many specific properties, such as testing linearity, testing monotonicity, testing graph properties, testing properties of distributions, etc.

In the *distribution property testing* [1,2,7], given sample access to one or more unknown distributions, we have to determine whether they satisfy some global property or are far from satisfying it. The roots of distribution testing lie in statistical hypothesis testing [14,15]. One of the most fundamental tasks in this field is deciding whether or not an unknown discrete distribution is approximately uniform on its domain. The problem is known as the problem of *uniformity testing.*

Uniformity testing was discussed by Goldreich and Ron in [9], motivated by the question of testing the expansion of graphs. They proposed a simple and natural uniformity tester that relies on the *collision probability* of unknown distribution. The collision probability of a discrete distribution $\mathcal{P}$ is the probability that two samples, drawn independently from $\mathcal{P}$, are the same. The main idea here is that the uniform distribution has the minimum collision probability among all distributions on the same domain, and that any distribution that is ε-far from uniform has a noticeably larger collision probability. Diakonikolas et al. [4] showed that a collision-based uniformity tester succeeds after drawing $O(N^{1/2}/\varepsilon^2)$ samples from the unknown distribution. Chan et al. [3] introduced a *chi-squared type* tester that uses $O(N^{1/2}/\varepsilon^2)$ samples. Here and below, ε denotes the "allowed" deviation from uniformity.

A related uniformity testing problem is the following: Given two distributions over an N element set, we wish to check whether these distributions are statistically close by only sampling. Batu et al. [1] showed that $\Omega(N^{2/3})$ samples are necessary but $\tilde{O}(N^{2/3}/\varepsilon^4)$ samples are sufficient in order to distinguish a pair of identical distributions from a pair of distinct distributions, where N is an upper bound on the support of the distribution.

3 NIST Randomness Tests Suite

The NIST test suite implements 15 empirical tests developed to test the randomness of binary sequences. Some of the NIST tests have a preprocessing phase.

We go here over some of the tests. The first NIST randomness test is the *frequency test*. This test checks whether the frequencies of 0-s and 1-s across the sequence are approximately equal. Thus, for example, the periodic sequence 010101... passes the test. The second NIST randomness test is the *frequency test within a block*. It examines the proportion of 1-s within blocks of some arbitrary length. The third NIST randomness test is the *run (oscillation) test*. It focuses on the total number of *runs* in the sequence, where a run is an uninterrupted sequence of identical bits. It checks whether the numbers of runs of 0-s and of 1-s of various lengths are, as may be expected for a random sequence. In particular, it verifies that the number of oscillations from 0-s to 1-s and vice versa is neither too small nor too large.

In general, NIST's tests may be divided into three classes [21], as follows:

- The fastest tests, that process each bit of the bitstream but once – *Frequency, Block frequency, Runs, Longest run, Cumulative sums, Random excursion, and Random excursion variant.*
- Fast tests that process M-bit blocks – *Non-overlapping template matching, Overlapping template matching, Universal, Serial, and Approximate entropy.*
- Slow and complicated tests – *Linear complexity, Discrete Fourier transform, Binary matrix rank.*

4 Using (True) Randomness for Randomness Testing

In this section, we present our approach for randomness testing. Unlike the best deterministic algorithms used in practice, we propose probabilistic algorithms. Thus, an attacker, who wants to produce sequences passing the randomness test, has a much harder task.

Random Sampling of Blocks. Our approach consists of sampling random blocks. Here, a "block" may be a string made of several consecutive entries in the given sequence, but it may also consist of any random entries. The basic observation is that, once the entries are distinct, the probability of such a block of a given length M being any specific binary block of length M is $1/2^M$. We would like to choose the blocks in such a way that, if the sequence is not a TRG sequence, the statistics of the blocks we sample will indicate this fact.

Suppose the given sequence is a TRG sequence. We select randomly K blocks $a_1, \ldots, a_K$ of length M each. There may be some overlapping between the blocks, but the number K of blocks is usually sufficiently small relative to n to make the overlapping negligible. We count the pairs $(a_i, a_j), 1 \leq i < j \leq K$, of blocks for which $a_i = a_j$. We use the proportion of these pairs out of all $\binom{K}{2}$ pairs to decide whether or not our sequence is random. It is easy to verify that, if the sequence is random, then the proportion is on average $1/2^M$. The key observation is that, if the sequence is constructed by a random generator, but not a TRG, regardless of the precise randomness mechanism, the proportion in question is on average strictly larger than $1/2^M$. Given a parameter ε, denoting the "allowed error", we take $K = \sqrt{2^M}/\varepsilon^4$. The sequence is accepted as a TRG sequence if the proportion of equal pairs is less than $(1+2\varepsilon^2)/2^M$, and rejected otherwise [7, Algorithm 11.3].

In our experiment below, we have used the following four types of blocks. All blocks are of length $M = 5$.

- **Test 1:** Consecutive blocks with a uniformly random initial bit.
- **Test 2:** Blocks of length M, consisting of M uniformly random bits.
- **Test 3:** Blocks of the form $[i, i+\Delta, i+2\Delta, \ldots, i+(M-1)\Delta]$. Here, the locations are to be understood modulo n, the parameter Δ is a pre-determined integer between 1 and $n-1$, and i is chosen uniformly randomly between 1 and n at each iteration. In our experiment, we used $\Delta = 5$.
- **Test 4:** Blocks of the form $[i, i+\Delta, i+2\Delta, \ldots, i+(M-1)\Delta]$, where Δ is chosen uniformly randomly for the whole test, and i is chosen uniformly randomly at each iteration.

In all tests, we have used an error parameter $\varepsilon = 0.1$.

5 Types and Parameters of Tested Sequences

Some types of non-TRG data may be harder to detect than others. Let us consider a few types of such sequences, used later in our tests for comparing the various algorithms for randomness testing.

- **Type 1:** Perhaps the simplest way to generate a non-random sequence is to take each entry of the sequence as 0 or as 1 with distinct probabilities, keeping the assumption that bits are independent. The non-randomness of such a sequence should be easily detected unless the two probabilities are very close to 1/2. In our tests below, we have taken $P(0) = 0.6, P(1) = 0.4$.
- **Type 2:** Markov chain based sequences. The first bit is chosen uniformly randomly, and then we continue according to some 2×2 transition matrix $(P(i,j))_{i,j=0}^1$. That is, if some bit in the sequence is $i \in \{0,1\}$, then the next bit is 0 with probability $P(i,0)$ and is 1 with probability $P(i,1) = 1 - P(i,0)$. The interesting case is when the stationary probability vector is still $(1/2, 1/2)$. Thus, in our experiments we have used the matrix
$$\begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}.$$
With this matrix (for $p \neq 1/2$), the overall proportion of 0-s and of 1-s is 1/2 each, but consecutive bits are dependent. Each bit tends to be followed by the same bit if $p > 1/2$ and by the opposite bit if $p < 1/2$. We have tested the resulting sequences for $p = 1/3$ and for $p = 2/3$.

- **Type 3:** Sequences based on higher-order Markov chains. The first several bits are chosen uniformly randomly and independently, and then each bit depends on several preceding bits. Again, we wanted the stationary probability vector to be $(1/2, 1/2)$. Specifically, we have taken the transition matrix

$$\begin{pmatrix} p & 1-p \\ 1/2 & 1/2 \\ 1/2 & 1/2 \\ 1-p & p \end{pmatrix}.$$

  Here, the first row provides the transition probabilities from 00 to 0 and to 1, the second – from 01 to 0 and to 1, and so forth. We have used this matrix with $p = 1/3$ and with $p = 2/3$.
- **Type 4:** Sequences with some number of initial bits chosen uniformly randomly, and continuing periodically from that place on. Here, the sequence looks locally as truly random, and it seems to be trickier to detect the non-randomness. In our experiment, we have used a period of 100.
- **Type 5:** A *de Bruijn sequence of order N* is a binary sequence of length 2^N, such that every binary string of length N appears exactly once as a substring. Here, the sequence is considered as cyclic, so that the number of substrings of length N is 2^N. We mention in passing that one may extend this notion to sequences over any finite alphabet. (For more information on de Bruijn sequences, we refer to [6,18,19].)

  While de Bruijn sequences are constructed by various algorithms (see, for example [6,18,19]), and thus do not qualify as TRG sequences, they display "better behavior" on various statistics than do TRG sequences. For example, the proportion of 0-s in a de Bruijn sequence is exactly 1/2, whereas in a TRG sequence, it is expected only to be approximately 1/2. The same holds for blocks of any length $\leq N$. These facts allow de Bruijn sequences to pass various randomness tests with flying colors.
  In our experiments, we have taken random de Bruijn sequences, using the python code at [10].

6 Results

In our experiment, we have generated TRG sequences, as well as non-TRG sequences of all the types described in the preceding section. All sequences were of length $n = 10^6$, except for the de Bruijn sequences, which were of order 20, and so of length 2^{20}, which is the power of 2 closest to 10^6. We have taken 100 sequences of each type.

All TRG sequences passed all randomness tests, both NIST's and ours. Thus in Table 1 we present the results for the non-TRG sequences only. If a certain test has succeeded in identifying all 100 sequences of a certain type as non-TRG, we marked a "+" in the corresponding entry of the table, while if it failed for all sequences we marked a "−". When it succeeded for some sequences and failed for others, we marked in the table the percentage of successes.

Table 1. Rate of success for various non-TRG sequences

	Random P(0) = 0.6 P(1) = 0.4	Markov		Markov order 2		Random periodic	Random de Bruijn
		p = 1/3	p = 2/3	p = 1/3	p = 2/3		
Frequency	+	5%	–	–	–	92%	–
Block frequency	+	+	–	–	–	41%	–
Runs	+	+	+	+	+	+	–
Longest run	+	+	+	+	3%	+	–
Binary matrix rank	2%	1%	1%	–	–	+	–
Discrete Fourier transform	+	+	+	+	+	+	9%
Non-overlapping template matching	+	+	+	46%	+	+	–
Overlapping template matching	+	+	+	7%	2%	+	–
Universal	+	+	+	+	+	+	+
Linear complexity	–	1%	–	2%	–	+	3%
Serial	+	+	+	+	+	+	–
Approximate entropy	+	+	+	+	+	+	+
Cumulative sums	+	8%	–	–	–	94%	–
Random excursions	47%	69%	31%	–	2%	33%	–
Random excursions variants	2%	+	4%	2%	1%	11%	–
Our test 1	+	+	+	+	+	+	–
Our test 2	+	–	–	–	–	44%	–
Our test 3	+	–	–	–	–	+	+
Our test 4	+	–	–	–	–	61%	–

The results are mixed. Each test succeeds more in detecting that some types of non-TRG sequences are such, but succeeds less on others. Altogether, the tests we propose seem to perform well on some types, and may add credibility to the claim that certain sequences are TRG.

Throughout our experiments, we have used pseudo-random sequences instead of TRG sequences. The reader should bear with us, as we were able to present to the judge a pattern in the examined sequence that reveals a deterministic pattern. This is just like a competition between two sequences, with a judgeable evidence as an output. Thus, a better (pseudo or true) random sequence may be able to "fight" a weaker pseudo-random sequence. Specifically, the pseudo-random sequences we have used instead of TRG sequences are based on *Math.random()*. This function returns a *double* value in the interval $[0, 1)$. The returned value is chosen pseudo-randomly with (approximately) uniform distribution in that range. When the function is invoked to generate a random sequence, the seed is initialized to a value based on the current time in milliseconds. Two random sequences, generated using this same seed, will be identical. According to the documentation of the *Util* Java package, *Math.random()* uses a linear congruential pseudo-random number generator, as defined by Lehmer and described by Knuth [12, Sect. 3.2.1].

7 Concluding Remarks

We have initiated a study on using randomness to test randomness. The motivation is to cope with an algorithm designed to produce deterministic sequences that pass the (NIST) randomness tests. In other words, the (accumulating [5]) automata that represent the (NIST) tests define a language of sequences (just as automaton defines languages) that are considered as sequences produced by non-true random source, and at the same time defined sequences not in the language, including, non-random sequences that can be constructed to pass the tests. Introducing randomness to the testing can imply the need to extend the input by augmenting possible random sequences (with exponential possibilities for such sequences) in order to (surely) identify a non-random sequence that passes the randomness testing.

A randomized test allows repeated examination of a sequence claimed to be random. Even if the conclusion is due to be that the sequence is a TRG-sequence, but one of the random choices made during the examination reveals a pattern in the examined sequence, this random choice may serve as an evidence in court. This, in turn, may be an incentive for the sequence creator to use a truly random device.

The implementation of our randomness tests is available on Github [13]. For instance, we have a sequence [13, HarvestedSequence.txt], in which we have planted in equally spaced locations (821 bits away) 1-s and 0-s, in an alternating order, within a random sequence. Thus, the sequence 010101... can be found in the sequence if one reads the bits at the indices $821 \cdot j$ for $j = 1, 2 \ldots$. The resulting sequence passes all of NIST's tests. A version of Test 3 in which a very "large block" (or sub-sequence) is constructed and then examined by the existing tests, for example finding the sequence 010101... by randomly choosing $i = 1$ and $\Delta = 821$ or $i = 822$ and $\Delta = 821$. A sequence 101010... together with $i = 822$ and $\Delta = 821$ can obviously serve as an evidence in court.

Acknowledgment. We thank Dean Doron for many interesting discussion related to this research.

References

1. Batu, T., Fortnow, L., Rubinfeld, R., Smith, W.D., White, P.: Testing that distributions are close. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12–14 November 2000, Redondo Beach, California, USA, pp. 259–269 (2000)
2. Batu, T., Fortnow, L., Rubinfeld, R., Smith, W.D., White, P.: Testing closeness of discrete distributions. J. ACM **60**(1), 4:1–4:25 (2013)
3. Chan, S., Diakonikolas, I., Valiant, P., Valiant, G.: Optimal algorithms for testing closeness of discrete distributions. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, 5–7 January 2014, pp. 1193–1203 (2014)

4. Diakonikolas, I., Gouleakis, T., Peebles, J., Price, E.: Collision-based testers are optimal for uniformity and closeness. Chic. J. Theor. Comput. Sci. **2019**, 1–21 (2019). http://cjtcs.cs.uchicago.edu/articles/2019/1/contents.html
5. Dolev, S., Gilboa, N., Li, X.: Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation. Theor. Comput. Sci. **795**, 81–99 (2019)
6. Fredricksen, H., Maiorana, J.: Necklaces of beads in k colors and k-ary de Bruijn sequences. Discret. Math. **23**(3), 207–210 (1978)
7. Goldreich, O.: Introduction to Property Testing. Cambridge University Press, Cambridge (2017)
8. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. J. ACM **45**(4), 653–750 (1998)
9. Goldreich, O., Ron, D.: On testing expansion in bounded-degree graphs. In: Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman, pp. 68–75 (2011)
10. jmviz: random-debruijn (2020). https://github.com/jmviz/random-debruijn/blob/master/debruijn.py
11. Kendall, M.G., Smith, B.B.: Randomness and random sampling numbers. J. R. Stat. Soc. **101**(1), 147–166 (1938). http://www.jstor.org/stable/2980655
12. Knuth, D.E.: The Art of Computer Programming, Seminumerical Algorithms, vol. 2, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1997)
13. Kumar, M.: Randomness test of sequences (2022). https://github.com/manishkk/Randomness-Test-of-Sequences
14. Lehmann, E.L., Romano, J.P.: Testing Statistical Hypotheses. STS, Springer, New York (2005). https://doi.org/10.1007/0-387-27605-X
15. Neyman, J., Pearson, E.S.: On the problem of the most efficient tests of statistical hypotheses. Philos. Trans. R. Soc. London Ser. A **231**, 289–337 (1933). http://www.jstor.org/stable/91247
16. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)
17. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-Allen and Hamilton Inc. McLean VA (2001)
18. Sawada, J., Williams, A., Wong, D.: A surprisingly simple de Bruijn sequence construction. Discret. Math. **339**(1), 127–131 (2016)
19. Siu, M., Tong, P.: Generation of some de Bruijn sequences. Discret. Math. **31**(1), 97–100 (1980)
20. Soto, J., Bassham, L.: Randomness testing of the advanced encryption standard finalist candidates, 1 April 2000
21. Sýs, M., Říha, Z.: Faster randomness testing with the NIST statistical test suite. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) SPACE 2014. LNCS, vol. 8804, pp. 272–284. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12060-7_18
22. Turan, M.S., Barker, E., Kelsey, J., McKay, K.A., Baish, M.L., Boyle, M., et al.: Recommendation for the entropy sources used for random bit generation. NIST Special Publication **800**(90B), 102 (2018)

Botnet Attack Identification Based on SDN

Avresky Dimiter[1] and Dobrin Dobrev[2(✉)]

[1] IRIANC, Munich, Germany
autonomic@irianc.com
[2] TU – Sofia, Sofia, Bulgaria
d_dobrev@tu-sofia.bg

Abstract. The framework that we are proposing is based on Virtual Security Functions (VSF), Openflow, Wasuh (Open-Source Security Platform), Software Define Network, Mininet, Pox Controller, Virtual Switches. By using Openflow protocol through virtualized environment of SDN we are capable to analyse entire data stream in network environment. By creating botnet identification virtual security functions, we are capable to increase network security by blocking the attack at the time of the initiation. We are constantly monitoring the network connections and in case of malicious activities Pox Controller is blocking it. VSF will allow to use the capability of framework, in order to protect against different botnet attacks. Each security functions can be activated concurrently for anomaly detection. All functions can be run in parallel and based on stream analyses of Openflow table can identify anomaly.

Keywords: Security function · Botnet · Distributed denial-of-service attack · Network function virtualization · Virtualization · Openflow · Flowtable · Controller · Wasuh

1 Motivation

Nowadays DDoS attacks initiated by Botnet has become one of the most reproduced methods for stopping businesses operations. Potential lost for lack of operations has been evaluated for millions of dollars a year [1]. Ransom Distributed Denial of Service (DDoS) has become very popular due to the direct monetarization from attackers. During AWS DDoS attack more than 2.3 Tbit attack has been observed. DDoS attacks are the primary driver of larger network volumetric events. The most commonly observed vector of this attack were DNS reflection, NTP reflection [2] and SYN flood [3]. Based on this motivation, the need of an efficient solution to block and significantly reduce DDoS attacks has emerged.

2 Selected Features of SDN for Solving Security Problems

Software-defined networking (SDN) consist of multiple kinds of network technologies designed to make the network more flexible and agile [4]. It is a new approach of using

https://doi.org/10.1007/978-3-031-07689-3_12

open protocols, such as OpenFlow, to apply globally aware software control [5]. The decision how packet is to be transferred from one hop to another is taken by SDN Controller or Control plan [6]. The delivery is possible with the network virtualization and the separation of Data plan and Control plan [7]. Several research ideas based on SDN have been proposed in [8], since the publication of DDoS protection for SDN [9], which is a key component in realizing the SDN concept for decentralized protection. We will focus on security protection realized by VSF. Botnet attacks initiating DDoS attacks are targeting directly availability of the systems. This area needs to be developed to assure a proper level of protection. Current possible solutions on the market consist of big and expensive scrubbing centers where the entire traffic is inspected and black hole [10]. Most of the cases attacks are generated from botnets. A Botnet is a group of computers which have been infected by malware and have come under the control of a malicious actor [11]. Our approach is to identify those compromised host and to stop their activities by cutting the connection with the Command-and-Control server or by quarantine the entire host. In order to have full visibility of the internal segment we are including Wasuh [12] agent as open-source security platform. The main goal of DDoS identification as security function is to identify that some of the hosts are part of Botnet network and is sending malicious packets or spoofed request. As soon as this malicious device is identified Botnet Mitigation [13] security function is triggered by the Pox Controller. This VSF is capable of send blocked request over the virtualized environment and to put directly dynamic access list to the firewall. To prove our theory, we are building our environment totally virtualized. On the hypervisor we have installed one Linux with Ubuntu 14.04.4 and customized version on Mininet version 2.2.2 [14]. We use also install Pox Controller [15], Mininet we use to emulate SDN environment. In Mininet environment, we have one SDN switch and emulated hosts [14]. Botnet machine will send spoofed request to the victim over the Sflow Switch [16] and over the edge router. Incoming traffic is mirrored in order to be analyzed by the Pox Controller [17]. Wasuh agent is installed on host and all logs are centralized to the Wasuh Management. Suspicious IPs are identified by the amount of generated traffic. If the generated traffic is bigger than the acceptable functional level, this traffic should be blocked in order to preserve the system. Based on the proposed security function we had prepared a model covering actions in case of suspicious activities are identify (Fig. 1).

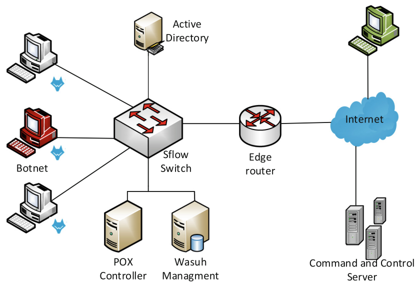


Fig. 1. Theoretical model based on Mininet environment

The novelty of our work is by programming the SDN controller to identify compromised host part of Botnet used for DDoS attacks. The entire dataflow is presented on Fig. 2. On this flow we are inspecting the outcoming traffic. The first request is for examining the if the communication is known for us. We are looking in our internal database to identify if the packet is from known source. After that we are verifying if this IP is from blocked country. We have a list of countries that are blocked for communication due to excessive number of attacks generated by them. We are we are verifying the public IP addresses by sending queries in the public databases. Next verification that we are applying is for abnormal amount of request generated from this IP. By this verification we can establish if this IP address is used as command-and-control server. As next step we are pushing this IP to the Botnet mitigation security function for restricting it. Our goal is to block the possibility of the compromised host to communicate with the command-and-control server. If we block this illegal communication channel, host is not part of Botnet network. By minimizing the number of hosts in the Botnet we are minimizing the amount of packet sent for DDoS attack. DNS requests are one of the most common ways for generating amplification attack. And in case of excessive amount of traffic IP is sent to Botnet Mitigation function for blocking. Similar approach we are applying for NTP amplification attack. An NTP amplification attack is a volumetric DDoS attack in which an attacker exploits a Network Time Protocol (NTP) server functionality in order to overwhelm a targeted network. For monitoring purposes, we are sending alerts to system for further analyses. All those details may be seen in Fig. 2 as functional blocks. In order to protect our segment, we had used model presented in [15] as a foundation in order to discover anomalies in the dataflow. As soon as one address is classified as malicious it is pushed to Botnet Mitigation function. Details scheme of this Security function can be seen in Fig. 2.

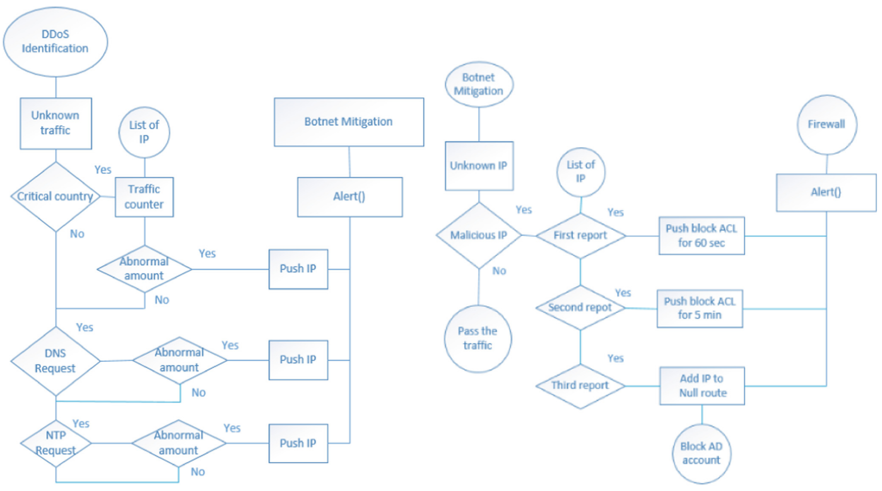


Fig. 2. Block diagram DDoS Identification function

In our work we are analyzing if the provided IP is malicious and in it is classified as suspicious, we are applying different blocks on it. 3 types of alerts are classified based on the frequency of the register events. In case, the IP address is identified for the first time, we are blocking only the communication between the source and the destination for 60 s. In case there are second attempts for communication we are blocking the port for 5 min. If for a third time is registered such activity, we are putting the hardest sanitation measures that we can apply. Source host is assigning with null route, in this situation it is not able to send any packet even in the internal network. This kind restriction is needed in order to minimize the risk for compromising another host in the local network. And as second measure respective user account is blocked in the active directory. This measure is needed for eliminating the possibility for compromising the entire infrastructure and to eliminate the risk for starting other type of DDoS attacks. In this function we are operating on two device edge firewalls by applying dynamic access lists and the Active Directory to enforce the active response for the malicious host. Information is collected from two nodes Wasuh Management Console and flow traffic from the Sflow Switch. With all this collecting information we have full visibility of the entire segment. This approach give is the possibility to increase the protection of our network. We are easily identifying if there is a compromised host. We can enforce direct remediation to limit the spread of attack.

3 Experimental Results

By using Mininet different type of topologies have been created. We can use those type of graphs in order to describe our framework's environment and to test our concepts and models. One example for a simple topology is shown in Fig. 3.

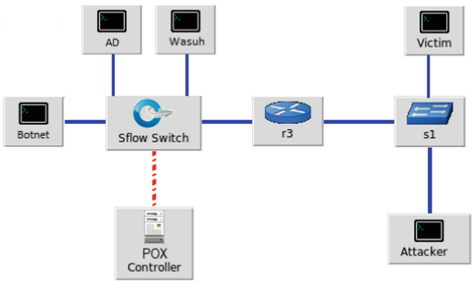


Fig. 3. Emulated environment for testing DDoS environment.

By using Mininet functionalities, we are emulating the real environment. Each host is virtualized. For verification of the connectivity between hosts, we are establishing connection over IP. We would like to clarify that in Fig. 3, we are flooding the remote host called Botnet. All send packet are with changed source IP in order all replies to be sent to the Victim. We will use Network Time Protocol (NTP) request in order to prove our concepts. It is important to point out that this type of protocol is most used for such type of attack. The main reason is that the packet reply is much bigger set of data comparing to the request. This type of attack is used for reflective attacks, where the initiator spoofs the destination and send fake request. On this topology, we also use Sflow Switch. This device is used for visualization of the result in the emulated environment. For validation of users and rights it is necessary to represent Windows Active Directory, it is presented as "AD". The monitoring system is also represented on the diagram. For this service, we are using dedicated host and we are connecting it to the same virtual environment. Wasuh console will be used for a visualization of the result and a real time analysis of the generated events from all hosts. As soon as, Pox Controller identify compromised host a Push request are sent to the AD. This is used for minimizing the exposure of compromised host. To obtain more specific logs and to identify real threat scenarios we had mad special configuration in the Windows Event Viewer [18]. Also, we may identify if information gathering tools and malicious applications are running on this host. Some examples are Netcat, Metasploit or PsExec presented in reference [19]. For avoiding such cases, reports are generated in the Wasuh management interfaces. Also, the push request is sent to the Pox Controller. The main idea is to recognize when the host is compromised. On the remote host which is named Attacker presented in Fig. 3 we are using open-source tool for Denial-of-Service attacks. This tool is called Low Orbit Ion Cannon (LOIC) [20]. It is used for network stress testing or for generation of targeted attack. For the testing case we are generating 10000 packets. Used protocol is NTP working on port 123 over UDP. UDP requests have been sent without waiting for reply for more effective attacks. The configuration of the tool is presented in Fig. 4. The IP address of the targeted host is the following: 10.104.254.119.

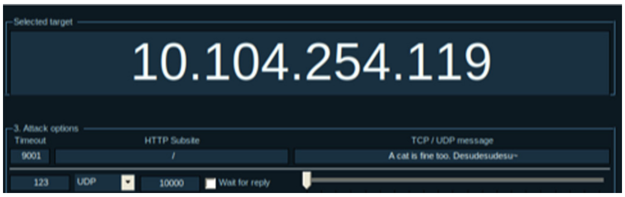

Selected target
10.104.254.119
3. Attack options
Timeout: 9001
HTTP Subsite: /
TCP / UDP message: A cat is fine too. Desudesudesu~
123
UDP
10000
Wait for reply

Fig. 4. Low Orbit Ion Cannon

Looking to the bandwidth generated from the Attacker, we may see that the entire interface is flooded. Only for 2 min, we are able to generate traffic up to 118 Mbit/s of outgoing NTP requests as shown on Fig. 5.

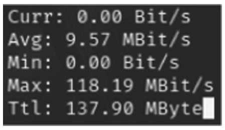

```
Curr: 0.00 Bit/s
Avg: 9.57 MBit/s
Min: 0.00 Bit/s
Max: 118.19 MBit/s
Ttl: 137.90 MByte
```

Fig. 5. Attackers interface statistics.

For validation of our result, we start flooding the single host for 20 min. During the first 2 min the system receives big number of packets. After those 2 min the protection starts blocking the packets. As a result, for 1 min the interface bandwidth goes down. Then the amount of traffic goes to the normal state. After that, the protection was stopped based on the algorithm presented in Fig. 2. As a result, the interface become overloaded again since the flooding was not stopped. After identifying the same attempt for attacks the protection was started again. In this case, the traffic was normalized for 5 min. After the expiration of this time interval, the traffic increases again and we see the anomaly of the traffic in the statistics. At this point, the permanent access list was applied on the firewall. These results are presented in detailed graph in Fig. 6. Based on the proposed algorithm for mitigation of DDoS attacks, we are able to eliminate one of the host from the Botnet network. In the proposed scenario, we are identifying if malicious information is generated.

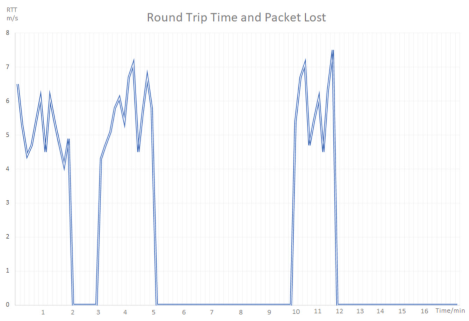
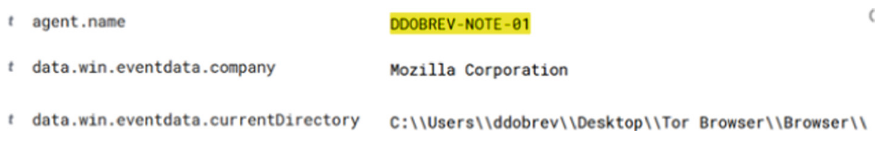


Fig. 6. Round Trip Time and packet lost distribution

For validation of the second method for protection, we started Tor browser on one of the host. The idea is to test the possibility for detecting malicious applications. Sysmon identify it as malicious application and has been sent this request to Wasuh Management interface. View from the interface is presented in Fig. 7.

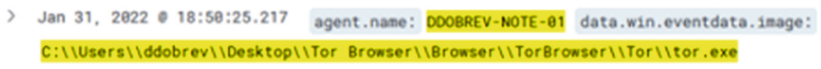


t agent.name	DDOBREV-NOTE-01
t data.win.eventdata.company	Mozilla Corporation
t data.win.eventdata.currentDirectory	C:\\Users\\ddobrev\\Desktop\\Tor Browser\\Browser\\

Fig. 7. Wasuh alarm triggered by Tor browser

Statistics with all triggered events can be find under Wasuh management console presented in Fig. 8. It is possible to identify the destination IP of the malicious host, the host that is used as Botnet Command and Control server and filtering. Sequence number and agent ID can be parsed for monitoring. Under the same console we can monitor different domain. In each packet we see the source domain and the main services that are triggering this event.

Based on the configuration of Sysmon, we can inspect different services and commands that are logged and analyzed. The novelty of our work is focused on the combination of different tools and technics for protection. By using the Software Defined



```
> Jan 31, 2022 @ 18:50:25.217  agent.name: DDOBREV-NOTE-01  data.win.eventdata.image:
  C:\\Users\\ddobrev\\Desktop\\Tor Browser\\Browser\\TorBrowser\\Tor\\tor.exe
```

Fig. 8. Tor Browser activity statistics

Network, we are having better visibility of the protected segment. Based on this, we have the possibility to interact directly with the data flow by the Pox Controller. It is capable of filtering the traffic in both directions. In this case, we can protect externally for malicious Slowloris attack, or we can protect the internal segment for compromised host trying to establish malicious communication. The interaction with the host is visible based on the integration of active directory and Wasuh Management Server. By using enhanced feature of the configured Sysmon and Wasuh agent we see all activities from the host. Based on the analyses of the created alarm, we may perform remediation action either on the firewall or on the active directory. This kind of approach give us better protection for all our services.

4 Future Work

As a future work, we are planning to adopt threat intelligence report as additional attack knowledge base. This will automatically update our functions for the new upcoming threats. It is possible to create security functions and to identify threats as a part of the behavior of specific attack. Different sub protocols can be used in the current DDoS identification mechanism. When a specific threat activity is identified, we will update our knowledge base. It will allow us to trigger security alerts. As soon as Pox Controller has managed all network traffic, then the specific threat can be blocked. Email alerts can be configured for specific threats for further analyses. Our goal is to implement a prediction model for Botnet detections and detections of compromised host in a real time. This prediction model will increase the automation of the service and can interact in a faster way for mitigating attacks. Current model is focusing of the approach of identifying the compromised host and quarantine him. Main idea for this detection is to identify the Command-and-Control servers that are interacting with the compromised host and to notify other users for such malicious actor. By this approach we will be able to increase global security.

5 Conclusion

In the paper, it was identified that one of the current problems in SDN is the security. Based on the proposed framework, using Pox Controller and Wasuh monitoring service we are capable to control dataflow by creating different security functions. This framework can work in a real-time. As well, it will be capable to detect different type attacks and block specific type of network traffic. We can analyse the Botnet traffic and by this to reduce DDoS attacks. This is achievable based on Pox Controller possibility to control the entire protected segment. As a result, the legitimate traffic cannot be lost. We can see that after filtering the traffic with our proposed Security Functions, it is possible to control the malicious request and to block them at the beginning of the generation of the attacks. In this way, we can protect the web server (Victim) from overloading. Monitoring service detects malicious activities in the level of initiation. Alerts are tiggered and full visibility is achieved on the protected segment. As a result, we are capable to increase network's security.

References

1. Pinho, M.: AWS Shield Threat Landscape Report - Q1 2020, 29 May 2020
2. Herzberg, B., Bekerman, D., Zeifman, I.: Breaking Down Mirai: An IoT DDoS Botnet Analysis – Imperva reading, 26 October 18
3. Antonakakis, M., April, T.: Understanding the Mirai Botnet. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, Canada (2017)
4. Braga, R.S., Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN) (2010)
5. Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., McKeown, N.: Elastictree: saving energy in data centernet works. In: 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010 (2010)
6. Popa, L., Yu, M., Ko, S.Y., Stoica, I., Ratnasamy, S.: Cloud Police: taking access control out of the network. In: Proceedings of the 9th ACM Hot Topics in Networks, Hot Nets (2010)
7. Bholebawa, I.Z., Dalal, U.D.: Performance analysis of SDN/OpenFlow controllers: POX versus floodlight. Wirel. Pers. Commun. **98**(2), 1679–1699 (2017). https://doi.org/10.1007/s11277-017-4939-z
8. Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M.: FRESCO: modular composable security services for software defined networks. In: Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS 2013 (2013)
9. Yoon, C., Park, T., Lee, S., Kang, H., Shin, S., Zhang, Z.: Enabling security functions with SDN: a feasibility study. Comput. Netw. **85**, 19–35 (2015)
10. McKeown, N., et al.: Open Flow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**, 69–74 (2008)
11. Cloudflare knowledge base - What is blackhole routing
12. Vukalović, J., Delija, D.: Advanced persistent threats - detection and defense. In: 2015 IEEE 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (2015)
13. Feily, M., Shahrestani, A., Ramadass, S.: A survey of botnet and botnet detection. In: 2009 3rd International Conference on Emerging Security Information, Systems and Technologies. IEEE (2009)
14. Bawany, N.Z., Shamsi, J.A., Salah, K.: DDoS Attack Detection and Mitigation Using SDN: Methods, Practices, and Solutions. Computer Engineering and Computer Science
15. Kaur, S., Singh, J., Ghumman, N.S.: Network programmability using POX controller. In: International Conference on Communication, Computing & Systems (2014)
16. Sflow-RT Telemetry, analytics, and control with sFlow standard
17. Medeiros, I., Neves, N., Correia, M.: Statically Detecting Vulnerabilities by Processing Programming Languages as Natural Languages. 1, 1, Article 1 (2016)
18. Manso, P., Moura, J., Serrão, C.: SDN-based intrusion detection system for early detection and mitigation of DDoS attacks. Information **10**(3), 106 (2019). https://doi.org/10.3390/info10030106
19. Sazak, S., Rebane, J.: Sysmon - Wazuh Sigma Rules – GitHub, 29 June 2007. GNU Public License
20. Praetox, A.B.: Low Orbit Ion Cannon (LOIC) 2018 network stress testing tool

Setting Up an Anonymous Gesture Database as Well as Enhancing It with a Verbal Script Simulator for Rehabilitation Applications

Yoram Segal(✉) and Ofer Hadar(✉)

Communication Systems Engineering Department, Ben Gurion University of the Negev (BGU), 84105 Beer-Sheva, Israel
yoramse@post.bgu.ac.il, hadar@bgu.ac.il

Abstract. Physical therapy patients are rehabilitated by performing exercises at home that do not consider proper movement and can be detrimental to the healing process. Maintaining patient anonymity is an important aspect of collecting patient data. Using our method, we are able to collect information about limb movements in a completely anonymous manner by taking a picture of the patient in the clinic and immediately converting the picture into an anatomical skeleton. A human gesture database accompanied by a verbal script simulator and anonymous tagging was created with the intention of tagging, measuring, and inferring human gestures using neural networks. We have developed a system that utilizes neural network autoencoder architecture to classify the quality and accuracy of patients' movements in videos. Since there is a lack of videos of tagged physiotherapy exercises, we simulate patients' movements to enhance the database. The purpose of this paper is to describe a simulator that mimics the output of OpenPose software so that synthetic human skeletal movements can be computed without utilizing OpenPose. As inputs, these vectors are fed to the autoencoder which, after compressing them into low dimension vectors, classifies them according to their movement using the Dynamic Time Warping (DTW) distance algorithm. Validation of the research was performed on a dataset of 7 different physiotherapy exercises, and 91.8% accuracy was achieved.

Keywords: OpenPose · Anonymous Gestures · Simulation · Siamese network · Physiotherapy exercises · Metaverse

1 Introduction

The purpose of this paper is to identify and describe body gestures by using machine learning technologies in order to create a "dictionary of gesture" - the Vocabulary Dictionary (VOCD). The VOCD will enable users to design a vowel-based textual language. Thus, it will be able to transform gestures to text and vice versa. Similarly, sign language

This work was supported a grant from the Ministry of Science & Technology, Israel & the Ministry of Education, Youth and Sports of the Czech Republic.

© Springer Nature Switzerland AG 2022
S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 170–179, 2022.
https://doi.org/10.1007/978-3-031-07689-3_13

for the deaf/mute can be transformed to text and vice versa; an avatar can then read the text strings and deliver the sign language. Lip-reading can also be converted to text and vice versa, resulting in an avatar whose mouth moves when the words are spoken. Aside from that, the tags will be used to compress and convert digital video to a written summary. In this paper we will show how to extract human body movements from a video, including gesture name, gesture time, repetition count, gesture speed, movement acceleration (e.g., calories), and so on. Through the use of algorithms such as OpenPose [1] and Mediapipe [2], we will characterize human motions by feeding them into neural network architectures such as Autoencoder [3], Siam's Twines [4], and DTWNet [5]. In this paper, we will address common denominator questions with respect to human gesture, such as synchronization and normalization. We will describe a neural network as a tool for analyzing, measuring, and labeling skeleton graphs in time and space. We will present practical results of our research study through collaboration with the University of Prague and within the framework of the Israeli Ministry of Science and Technology, by exposing software that enables learning and assessment of physiotherapy activity in a home environment during rehabilitation. This innovative solution indicates that our study's results will help to lower the danger of physical contact and the associated costs of rehabilitation. We demonstrated how to combine data from multiple sources and extract metrics, to provide a quantitative description and labeling of motions using only one or a few cameras. Deep-Learning-extracted patient metrics was utilized to calculate the deviation from the doctor's initial exercises. The underlying concept of the algorithm is to demonstrate how the study findings could be used, for example, for remote patient rehabilitation. Our aim is to explain how to provide patients with quantifiable data, and also to inform them of the discrepancy between their necessary and actual gestures. The VOCD will also be utilized to identify transient patterns of motion in the patient's motions while performing the exercise, as well as to generate a therapy report for the therapist. Especially in cases involving a large number of patients recuperating from hip, knee, elbow, or shoulder surgeries, this capability may be employed for remote therapy. Participants might be able to create a family of neural network architectures with the tutorial outcomes, by creating a wide range of contactless medical solutions.

2 Related Work

2.1 OpenPose (OP) as a Tool to Extract Human Body Skeleton

A real-time multi-person system named OpenPose is the first to jointly detect human body, hand, facial, and foot key points (in total, 135 points) on single images (or frames in videos) [6, 7]. For this paper, we only use the joints of the body which contain 25 vertices per frame. An individual vertex is composed of three components: the x, y location, and the c component which represents the level of confidence, where zero means there is no confidence and one mean that there is 100% confidence. As a result, it produces a skeleton vector on the basis of 75 components (25 vertexes and each has three components - x, y, and c). In this case, the data is presented in the form of a.json file. OpenPose requires a powerful computer in order to run, so we decided to develop a method to accelerate OpenPose's performance by using the H264 video encoder motion vector as a vertex tracking method [8, 9]. Another way to eliminate the need to use OpenPose is by developing a simulator. Body tracking is the first element that forms the basis for the presented algorithm. As there are precise systems with perfect tracking possibilities, the price is relatively high. Usually, assistance is required before or during the operation, so these systems are not suitable for use in the home environment. As part of our research, we are focusing on a system with affordable prices that is as easy to use as possible. There is something like a gold standard when it comes to body tracking. A system based on infrared markers is an expensive and complicated system. In the early 1980s, one of the most well-known systems was Vicon, which was the first commercial motion capture system. Some of the similar systems with similar features are OptiTrack [10], Qualysis [11], or BTS [12]. Many systems are based on multiple synchronous infrared cameras, several infrared reactive markers, and individual body models. It has several advantages, including Very precise, Long-term research, and many existing models. Its disadvantages are: costly, it takes a long time to prepare for measuring, it requires qualified operators, and it cannot be used in a home environment. For this paper, we will use algorithms such as OpenPose, which provide good skeleton extraction quality. Google Mediapipe [2], PoseNe [13], or WrnchAI [14] can be considered as an alternative. The research of OpenPose is presented in detail by the authors of the system and publications [1, 15], and [16]. The pose estimation is based on pre-trained models of body parts. A deep neural network is used to construct the models. As the name suggests, the system is frame-based. A separate estimate of position is made for every single image. One of the most well-known 3D scanners currently available [17] is the Microsoft Kinect. Xsens [18], Rokoko [19], and even the intelligent captain suit developed by [20] can be used as an example of a full-body commercial capturing suit. The systems can be used for both full-body or specific applications. A framework presented by researchers at Dublin University was used to capture the 3D trajectory of a golf swing or other sports equipment [21]. The technique is based on inertial sensors, such as accelerometers, gyroscopes, and magnetic sensors. Using mathematical transformations, Kalman filters from acceleration, angular velocity, and gravitational force to obtain a 3D position in space.

2.2 Human Body Simulation

Simulation of human characters is often critical in the development of video games, virtual reality, and fiction films, as well as in NN synthetic datasets. Deep neural networks have driven recent developments in deep learning and deep reinforcement learning in order to achieve this. The authors in the article [22] present a thorough assessment of state-of-the-art techniques for the animation of skeleton-based human characters that are based either on deep learning or deep reinforcement learning. To begin, they explained motion data formats, the most commonly used human motion datasets, and how fundamental deep models can be upgraded to allow the discovery of spatial and temporal patterns within motion data. They proceed to discuss state-of-the-art methodologies in three broad categories of human animation pipeline applications: motion synthesis, character control, and motion editing. Last but not least, they discuss the constraints of existing state-of-the-art approaches for the animation of skeletal human characters based on deep learning and/or deep reinforcement learning, as well as potential avenues for future research to relieve current restrictions and meet expectations. In this article, it is demonstrated that there is an extensive amount of activity taking place in the area of simulations of human movements. It is important to mention that the simulator we have developed is innovative in two ways: the first is that it calculates human skeletal structures instead of the human's body, and the second is that it does not rely on OpenPose software, which is a heavy consumer of computing resources and requires computers with GPU-based parallel processing capabilities.

3 Proposed Method

The main objective of the study is to investigate human body movements as compared to a ground truth movement (i.e. an exercise performed by a physiotherapist). In order to accomplish this objective, we must extract the body movements from a video stream and represent them as a collection of vectors that depict a graph of the human skeleton. The OpenPose algorithm is used to extract the human posture from a video frame and convert it into a vector with 75 components (75 dimensions). One of the problems is that each exercise has its own dominant vertices. During clapping hands for example, the feet vertexes are irrelevant, meaning that they do not move. In order to determine which vertices appear to be dominant, an algorithm must be used. Essentially, a dominant vertex is a vertex in which the variance is high and exactly this is what the PCA algorithm measures. The PCA algorithm is implemented by using an autoencoder network to implement the general PCA algorithm. Using the pose vector, an autoencoder is then used to reduce the 75-dimensional vector to 15 main dimensions (similar to performing PCA by using NN, but in a more general manner that includes support for nonlinear behavior as well). This restricted vector is then fed into the DTW algorithm, which calculates the distance between the therapist and the patient vectors. This distance is normalized, and it produces an indication of the extent to which the exercise is applicable (exercise scoring). It is the Autoencoder network which is used to provide the dominant vertices to be selected, which has been designed and trained in the manner explained above. One of the problems is that neural networks require a large database to work with. The creation of a large database in the world of medicine is a very challenging task. To address this problem, we have developed an algorithm that generates human movement from a textual description that is then converted into a vector representation similar to that produced by OpenPose. To be more specific, we generate for each frame a vector that describes a graph that is a representation of a human skeleton. The result of this process is a 75-dimensional matrix. The number of rows is the number of frames that are needed in an exercise cycle, and the number of columns is 75 (25 vertices and for each vertex, three components, x, y, c, totaling 75 columns). After the synthesized data set has been processed through NN, the vector dimension of the vector is reduced.

4 Patients Database

4.1 General Description

The proposed training set is based on physiotherapy exercises developed by Ben-Gurion University and the University of Prague with the support of the Chief Scientists of Israel and the Czech Republic. There are six basic physiotherapy exercises in the database, which have been carefully selected to be suitable for analyzing and processing with a single camera (two-dimensional processing).

4.2 Database Exercises Content

There are 100 participants in the database, who each perform six exercises.

(1) AFR - Exercise arm full range (side view)
(2) ARO - Exercise arm rotation (front view)
(3) LBE - Exercise leg backward extension (exercise with a chair, side view)
(4) LFC - Exercise lifting from the chair (side view)
(5) SLL - Exercise side leg lift (lower limb abduction, front view)
(6) TRO - Exercise trunk rotation (front view)

Ten cycles comprise each exercise (e.g. rotating the right arm). Exercises are performed once with a right tilt and once with a left tilt (for example, once with a right foot rotation and once with a left foot rotation). A total of about 7500 motion cycle videos have been tagged and timed in the database.

4.3 Database as Human Skeletons

The entire database has been encoded as skeletons - a skeleton in every frame. A NUMPY array file, a MATLAB file, and a JSON file are used to maintain the database. Performing exercises creates skeletal structures. The human body is represented by 25 vertices in each skeleton. The vertex has three components: Coordinate X, Coordinate Y, and Coordinate C, which indicates the level of certainty about each point in the skeleton on a scale from 0 to 1 (1-absolute certainty, 0 absolute uncertainty). Therefore, a single skeleton extracted from a frame is represented by a single vector with 75 components (3×25). The JSON files are structured so that one file exists for each frame, and inside each file is a list of several skeletons, one for each person in the image. Both the MATLAB files and the NUMPY files are organized in the same manner. Every file contains a matrix. The matrix contains 75 columns and the number of rows is equal to the number of frames in the video. In other words, each row describes an individual skeleton. If more than one skeleton was sampled in a single frame, then only the first skeleton, marked with ID 0, will be extracted.

5 Gesture Generator via Text Script

AI (artificial intelligence) is a revolutionary technology that is finding applications in a wide range of products and services that we use on a daily basis. An amazing application of AI technology is Deep Learning, which is based on neural networks. Deep learning-based application development requires a large dataset with a sufficient number of training examples. In the absence of such databases, researchers and data scientists use synthetic data to circumvent the problem. The work target is performing body movement classification from synthetic human skeletal vectors. The proposed solution for the classification is designing and training a NN. The process began with gathering and filming videos of predetermined movements and then converting these videos to OP format and manually tagging them. Our next step was to create a “Pose Dictionary”, which corresponds to each pose that is defined according to OP and compliments the vertices of a skeletal human body. Each movement is built from a sequence of poses in the pose dictionary; each movement is derived from a sequence of poses. The second step was to create a movement vector database. An artificial skeletal vector was created in order to train the NN with a large number of artificial skeletons. A Python program that uses a CSV file as input and creates a set of vectors that represent human skeletal movement from a set of input parameters including the name, the speed, the number of repetitions, and noise from the movement. The database has been divided into training and testing sets. We then created a NN autoencoder model and trained and optimized it on the training set, and evaluated it on the test set.

6 Synthetic Motion Simulator

The simulator receives a CSV file as an input that includes movement names, speed, repetitions, and noise. The output consists of a.npy matrix of the synthetic movements and a video file of the skeletal movement in a.mp4 format.

6.1 The Simulator Algorithm

The process of creating one movement can be seen in Fig. 1. Using this process, several movements can be repeated. As a consequence, we can create a choreography of one long movement as a result of the input. The movement can be combined into one motion by performing multiple movements in parallel or sequentially. Once the motion is complete, the.npy matrix of the new synthetic motion is saved, and from that, we make an animation of the.npy matrix that is saved in a.mp4 video file.

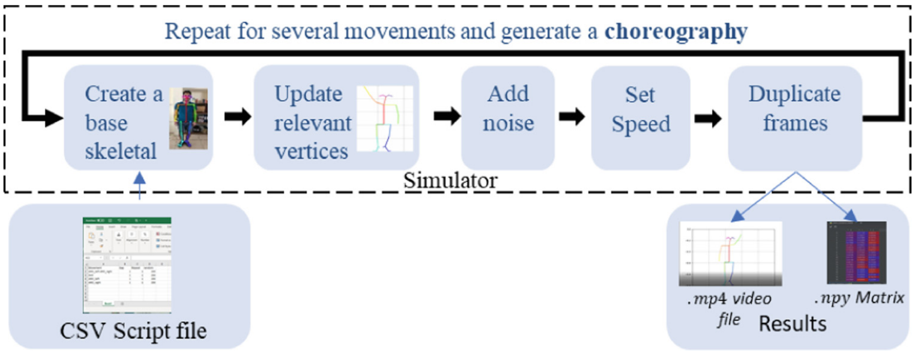


Fig. 1. Simulator flow

In order to build the simulator, we performed the work in several stages. In the beginning, we only updated the vertices that were relevant to the movement while the other parts of the body remained still. Because of this, it does not reflect real human movement because when an organ moves, the rest of the body moves slightly along with

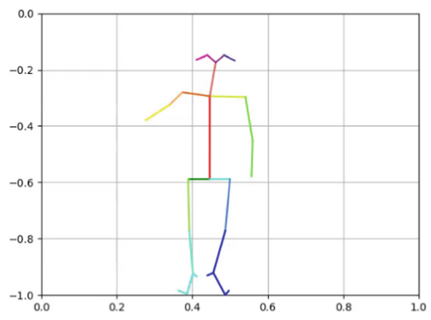


Fig. 2. ARO right movement.

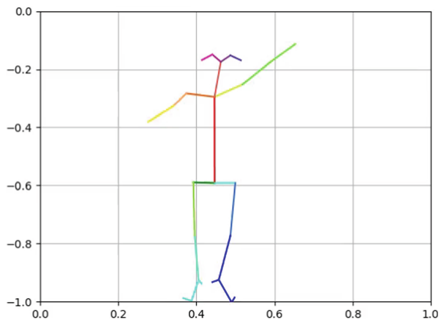


Fig. 3. ARO right and ARO left combined

it as well. We can see in Fig. 2 that the relevant vertices have changed, however, the body as a whole remains the same.

It is possible to see from Fig. 3 that we have added noise to the movement of the objects. It now looks like the objects are moving more naturally. Additionally, the number of repetitions and the speed of movement were also added to the parameters. To conclude, we combined several movements into one motion. By combining several movements, a new human movement was created. This can be accomplished in a parallel manner or by combining them in a sequence.

7 Experimental Results

The neural network that we built for evaluated the quality of the simulator was used. We test 2800 ($400_{\text{per motion}} \times 7_{\text{predefined motions}}$) motions, the data differs in type, speed, number of repetitions, the noise of the movement. The confusion matrix visualizes the performance of the system. Each row of the matrix represents the actual class of the tested motion while each column represents the predicted class. The accuracy is 91.82%:

8 Discussion

In this study, a new framework for the evaluation of rehabilitation exercises is presented using an enlarged database provided by a simulator. Measurements of performance, scores and neural network models are all part of the system. The effectiveness of various measures for assessing the consistency of rehabilitative motions is compared. In the study, Euclidean distances and DTW distances are taken into account. The dimensionality reduction of human motions has been achieved using the autoencoder NNs instead of PCAs and maximum variances. The goal of our deep learning architecture is to model rehabilitation activities through hierarchical Spatio-temporal modeling at several levels of abstraction. NNs are taught to perform all of these activities by using an autoencoder, which identifies quality ratings from inputs consisting of exercise repetitions. Despite the fact that the human body is not completely static, the simulator can accommodate this. Changing the position of the hand will cause the entire body to move. We were able to achieve a more natural movement by including noise in the movement. Simulators now have the option to choose from a variety of noise levels. The OpenPose software format does not offer a suitable tool for displaying animation. Consequently, we developed a Python-based program that receives a multidimensional matrix corresponding to a human skeleton as an input and converts it into the MP4 file format for the animation of the skeletal movements. In order to avoid overfitting of the neural network, we perform movement augmentation and inject this augmented skeletal information into a neural network to get a more reliable network that answers the expectations. Because the human body produces complex movements in which more than one limb moves at the same time, we chose to create a script that combines several movements executed one after the other, creating a motion that looks natural. Using the Euclid distance as a tagging and classification method, we were able to identify the movements. Therefore, we need to synchronize the movements of the compared images (the same movement in the same frame). It is a significant problem because the speed at which the motions are performed and the number of repetitions vary from person to person. A parameter that affects how many frames are generated. We were able to solve this problem using the DTW algorithm which allows signal comparison between signals that are not synchronized. An OpenPose matrix is created for each patient session (each row contains a vector that represents a human pose). Using the NN autoencoder, we were able to reduce the matrix dimension from 75 to 15 principal components. In order to be able to assign a score to the quality of the patient's movement relative to that of the therapist, we determined the DTW distances between each pair of matching columns - a column from the therapist's matrix (reference matrix) and a column from the patient's matrix. In order to determine the accuracy of the exercise, we accumulated all the results and averaged them to obtain the score for the degree of accuracy of the exercise in relation to the required.

9 Conclusions

We created a simulator of synthetic movements and generated over 4000 movements for the Neural Network database and testing. The simulator can create "choreography" by combining seven predefined movements. Adding more movements in the future is a simple operation. We have added parameters for each movement as speed, number of repetitions, and the volume of noise. These parameters differentiate the movements from each other generate a diverse database and avoid overfitting. Moreover, we have created an innovative suitable tool for displaying animation in the format of Open Pose software. Our system can classify synthetic movements out of 7 predefined movements. There is no limit of length, speed, noise, and number of repetitions. The same as the performance specification, the number of layers in the NN that give the best result is 5 hidden layers. Although we expected accuracy of 86% according to OpenPose accuracy, the overall system manages to produce and classify synthetic data with high accuracy of 91.82%. We met the project goals. Created a "Movement Dictionary" and built a synthetic motion simulator, generated a large motion database, and built and trained a model of an autoencoder Neural Network that classified the motions.

10 Further Work

The validation of such a solution is typically performed with expensive optical motion capture systems on healthy volunteers undergoing rehabilitation activities. Further, the majority of the validation relies on data without any ground truth evaluation by a physician regarding the quality of the simulated movements. As a result of the shortcomings in the current work, we intend to provide solutions by ensuring that the framework is fully validated by focusing on rehabilitation exercises done by patients and labeled by a panel of doctors who will award a quality rating in future work. Our main focus is on improving the simulator and the system in general by proposing a series of actions as follows: (1) Adding more motions to the simulator could result in a wider dataset and a broader variety of choreography by adding more poses to it. (2) Develop a deep neural network based on the DTW classification and employ it in the model. (3) To improve the performance of the NN, we can introduce a capability to determine the accuracy of the motions in addition to the ability to classify more complex motions.

Acknowledgment. This work was supported a grant from the Ministry of Science & Technology, Israel & The Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., Sheikh, Y.: OpenPose: realtime multi-person 2D pose. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7291–7299 (2017)
2. Lugaresi, C., et al.: MediaPipe: a framework for perceiving and processing reality. In: Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) (2019)
3. Liao, Y., Vakanski, A., Xian, M.: A deep learning framework for assessing physical rehabilitation exercises. IEEE Trans. Neural Syst. Rehabil. Eng. **28**, 468–477 (2020)
4. Koch, G.: Siamese Neural Networks for One-Shot Image Recognition. Graduate Department of Computer Science University of Toronto, Toronto (2015)
5. Cai, X., Xu, T., Yi, J., Huang, J., Rajasekaran, S.: DTWNet: a dynamic TimeWarping network. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
6. Tanaka, R., Oshima, C., Nakayama, K.: Intention inference from 2D poses of preliminary action. In: GECCO'19: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1697–1700, July 2019
7. Cao, Z., Hidalgo, G., Simon, T., Wei, S., Sheikh, Y.: OpenPose: realtime multi-person 2D pose estimation using Part Affnity Fields. arXiv preprint arXiv:1812.08008 (2018)
8. Segal, Y., et al.: Camera setup and OpenPose software without GPU for calibration and recording in telerehabilitation. In: IEEE E-Health and Bioengineering, Lasi, Romania (2021)
9. Segal, Y., Hadar, O.: Interpolation of missing frames of human body movements via video motion vectors - OpenPose accelerator. In: IEEE Conference on IoT for Rural Health Care, IEEE, CIRH-2021, Guntur, India (2021)
10. Cheung, K.H.: Optitrack - Estimation of Opti-track motion capture system data. Department of Electrical Engineering (2020)
11. Qualisys | motion capture systems, March 2020. https://bit.ly/YS-Qualisys

12. Motion capture system | BTS bioengineering, SMART-DX, March 2020. https://www.btsbioengineering.com/
13. Chen, Y., Shen, C., Wei, X., Liu, L., Yang, J.: Adversarial PoseNet: a structureaware convolutional network for human pose estimation. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 1212–1221, October 2017
14. Wrnch - teaching cameras to read human body language, March 2020. https://wrnch.ai/
15. Simon, T., Joo, H., Matthews, I., Sheikh, Y.: Hand keypoint detection in single images using multiview bootstrapping. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 1145–1153 (2017)
16. Wei, S., Ramakrishna, V., Kanade, T., Sheikh, Y.: Convolutional pose machines. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4724–4732 (2016)
17. Zhang, Z.: Microsoft Kinect sensor and its effect. IEEE Multimedia **19**(2), 4–10 (2012)
18. Home - Xsens 3D motion tracking, March 2020. www.xsens.com
19. Rokoko - motion capture system - Smartsuit Pro, March 2020. https://www.rokoko.com/en/
20. Shadow motion capture system, March 2020. https://www.motionshadow.com/
21. Ahmadi, A., Destelle, F., Monaghan, D., O'Connor, N.: A framework for comprehensive analysis of a swing in sports using low-cost inertial sensors. In: SENSORS, 2014 IEEE, pp. 2211–2214, November 2014
22. Lucas, M., Hoyet, L., Le Clerc, F., Schnitzler, F., Hellier, P.: A survey on deep learning for skeleton-based human animation. In: COMPUTER GRAPHICS Forum (CGF) (2021)

Fake News Detection in Social Networks Using Machine Learning and Trust

Nadav Voloch[1] (✉), Ehud Gudes[1], Nurit Gal-Oz[2], Rotem Mitrany[1], Ofri Shani[1], and Maayan Shoel[1]

[1] Ben-Gurion University of the Negev, P.O.B. 653, 8410501 Beer-Sheva, Israel
voloch@post.bgu.ac.il
[2] Sapir Academic College, 79165 Sderot, Israel

Abstract. Fake news propagation is a major challenge for Online Social Networks (OSN) security, which is not yet resolved. Fake news propagates because of several reasons, one of which is non-trustworthy users. Non-trustworthy users are those who spread misleading information either for malicious intentions or innocently as they lack social media awareness. As a result, they expose their sub networks to false or inaccurate information.

In our previous research we have devised a comprehensive Trust-based model that can handle this problem from the user Trust aspect. The model involves Access Control for the direct circle of friends and Flow Control for the friends' networks. In this paper we use this model and extend it for the purpose of preventing Fake News. We analyze user's activity in the network (posts, shares, etc.) to learn their contexts. Using Machine Learning methods on data items that are fake or misleading, we detect suspicious users. This addition facilitates a much more accurate mapping of OSN users and their data which enables the identification of the Fake News propagation source. The extended model can be used to create a strong and reliable data infrastructure for OSN.

Keywords: Online social networks security · Fake News detection · Trust-based security models

1 Background and Related Work

One of the major social media problems today is the spreading of fake news. Users spread information even if not necessarily procured from a reliable source or went through proper fact checking. The term *Fake News* may refer to all sorts of disinformation and misinformation. While disinformation implies false information that is deliberately created, misinformation refers to false or inaccurate information that are created by mistake. At extreme cases, fake news can get a lot of exposure and cause harm.

For example, text based on fabricated data about vaccines can cause a significant percentage of a population to not get vaccinated, and vice versa, which may consequently affect people's health.

In our previous work, we have created an OSN security model for the Ego network, that is composed of three main phases addressing three of its major aspects: Trust,

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 180–188, 2022.
https://doi.org/10.1007/978-3-031-07689-3_14

Role-based Access Control [1, 2] and information flow, by creating an Information Flow-Control model for adversary detection [3], or a trustworthy network [4], as we did in [5]. In this paper we use this model as a basis to address the problem of Fake News in OSN. Social network users are daily exposed to a lot of information from their network: posts by people with whom they are connected, or posts shared by their connections.

Fake News involve two major concerns which have become subjects for research:

a. Detecting and identifying fake news. This is a complicated task and has only partial success. The detection on social media is defined and presented in [6], and in [7] an important typology of Fake News is done in categorizing six different categories of Fake News: Native advertising, News satire, Propaganda, Manipulation, News parody, and Fabrication. The most severe form of Fake News is Fabrication since it has a very high Author's immediate intention to deceive, and a very low Level of facticity.

   [8] presents an approach that uses psychological estimation of OSN users to detect misinformation spreading in the network, and [9] uses semantic context to detect and analyze different categories of Fake News.
b. Preventing Fake News propagation. [10] deals with the propagation of Fake News and shows that the spreading of Fake News is done in a fast and thorough manner, since its nature is one of an extensive content, that has the potential of extremity.

In [11] different types of data spreading scenarios are described. Most of these vulnerabilities occur from discretionary privacy policies of OSN users.

These privacy policies create a misleading knowledge concerning the number and type of users exposed to this shared data. Most of the solutions suggested demand changes in these specific policies.

Figure 1 describes an Ego user's information spreading to friends of friends (Users A1, A2 and A3), triggered by an action (a comment in this example) taken by the ego node's direct friend (User A) on the Ego node's data.

The use of NLP in information security is currently researched and used (e.g. [12, 13]), and the implementation of such techniques is an important feature that is used to evaluate the users' content in different types of action on the OSN.

For identifying Fake News with our context model, we need to compute the different Trust values per context for the user's network - different users are expected to have different Trust values in different context categories. For that, we need to identify the context of a data instance. Such identifications are presented in [14] and [15].

Detecting Fake News by different types of learning is the topic of many very recent research papers such as [16], that proposes a linguistic model to find out the properties of content that will generate language-driven features, and [17] that uses geometric deep learning to detect Fake News in OSN.

[18] uses supervised learning for this detection, and the work was done specifically on Twitter datasets, that are naturally very accessible due to twitter's publicly open infrastructure. [19] surveys Fake News in a comprehensive manner, in terms of detection, characterization and mitigation of false news that propagate on social media, using a data-driven approach.

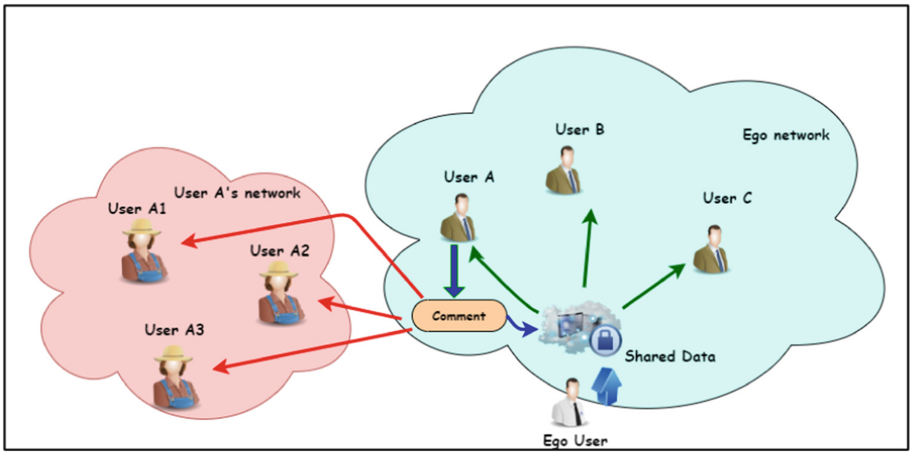


Fig. 1. Data spread, not necessarily intended, in OSN

The research presented here is based on the papers mentioned above, and its novel contribution is the unique method for Fake News detection by means of calculating Trust based on OSN features and context. A data instance can be characterized by its context (e.g., politics, sports, etc.), and the trust measure must be refined by this context. For context evaluation, we categorize different users in the Ego network by their Trust per context.

We calculate this Trust for the friends in the Ego network: different trust values for every category, meaning that they have a User Trust Value per category denoted here as UTV_κ for each category κ. The calculation is presented in [20].

2 Prevention of Fake News Propagation

An important use of the context extension to the model is the detection of Fake News and the prevention of their propagation. The model analyzes the network in a deep and comprehensive manner trust wise. The users and their social content are monitored and users that are identified by the model as not trustworthy, are suspected as potential spreaders of false data, and even as the possible source of this data. In this part we use the Sentiment Analysis Factor for an Action α in a κ category denoted $SAF_\alpha A_\kappa$ (taken from [20]) as an important indicator of Fake News. This is because Fake News usually contain polarized emotions (very positive or negative), as described thoroughly in [21]. The second indicator we use is the user's Trust value UTV^j_κ, that is described in [22]. The actions that were used to compute the $SAF_i A_\kappa$ for the UTV^j_κ are different from $SAF_\alpha A_\kappa$- which is a new action that we now examine whether it is considered as Fake News or not. To calculate the possible prediction that we here denote FNP_i (Fake News Propagation) whether a certain user or a page indexed i, can be Fake news propagators we combine three different factors: UTV, SAF and the Machine Learning Factor (MLF), on which we will elaborate on the next section. We give as default equal weights to these factors as described here:

$$FNP_i = \frac{w_{UTV} UTV + w_{SAF} SAF + w_{MLF} MLF}{\langle w \rangle} \tag{1}$$

As a basis for our sentiment analyzer, we used our predecessors' analyzer as well, to which we added some changes.

Instead of using a threshold to determine whether the post is very positive or very negative and then return a binary result, we took into consideration the negative and positive results and calculated a grade between zero and one. Zero indicates a neutral post, and one indicates an extreme post: either very positive, very negative, or both. Our basic premise in this part of the model is that actions that have very high, or very low (polar) values of $SAF_\alpha A_\kappa$, done by users that have low UTV^j_κ values, have the potential of being Fake News. For this purpose, we set at the initial state of the system, threshold values for these parameters, that can dynamically change in the process of learning. Although $SAF_i A_\kappa$, is used as a part of the calculation of UTV^j_κ, it serves a different purpose here- not as Trust estimator, but as a detector of polarized sentiment of data, thus it must be considered separately.

At the end of the learning process, we aim to detect users that have the most prominent potential of being Fake News propagators. These values can be adapted to another important parameter mentioned above, the Total number of Friends (*TF*). The higher this number is, the higher is the potential harm of this user. Thus, we can apply stricter thresholds for users that have a large network.

2.1 Our System - The Fake News Trust Based Analyzer

The purpose of our ML analyzer it to scrape Facebook post of different Facebook users that can be people, pages or groups. By analyzing their posts that have the most traffic, thus, most of the public influence, we can detect their potential of being fake news propagators. We denote these posts in the parameter *numOfPosts*. Our Scraper is based on Selenium WebDriver. The Scraper relies heavily on the structure of Facebook web pages, which differ greatly from one another. We support various page structures, but if an inspected web page is of a structure we don't support, some or all the data is not retrieved. Initially the Scraper checks the URL type. It then redirects to the appropriate function; there is a designated function for each type of URL:

Post: Retrieve the text in the post and account information of the writer. If the post was posted by a page, Page information is extracted (instead of account information). Return a Post object.

Group: Retrieve group information, such as group age and how many members it has. Retrieve the topmost *numOfPosts* posts. Return a Group object.

Account: If the user is logged in, retrieve account information such as account age, friendship duration, etc. Retrieve the topmost *numOfPosts* posts. Return an Account object.

Page: Retrieve page information, such as age and number of followers. The topmost *numOfPosts* posts are retrieved as well. Return a Page object.

If something other than these four URL types was entered, the function produces an error. When the scraping is finished, the returned object is sent to the Analyzer (via the manager). The architecture of the system is presented in Fig. 2.

In general, three different analyses can be calculated by the Analyzer: User Trust Value (*UTV*), Sentiment Analysis, and Machine Learning. These are calculated independently, and each returns a value between zero to one which is then converted to percentages. Finally, the Analyzer calculates a (non-weighted) average of the three, and produces a single grade, which is a percentage between 0–100, indicating reliability: 0 indicates low reliability (highest potential of spreading Fake News), and 100 indicates high reliability (lowest potential to spread Fake News). Four parameters are defined as prerequisites for all three analyses to be made; if they are not satisfied, then one or more will return N/A value, indicating it has not been calculated. In these cases, only the calculated grades will enter the final average calculation. *UTV* is calculated for an account, page, or group. We used our predecessors' code [23] as a basis and made some changes.

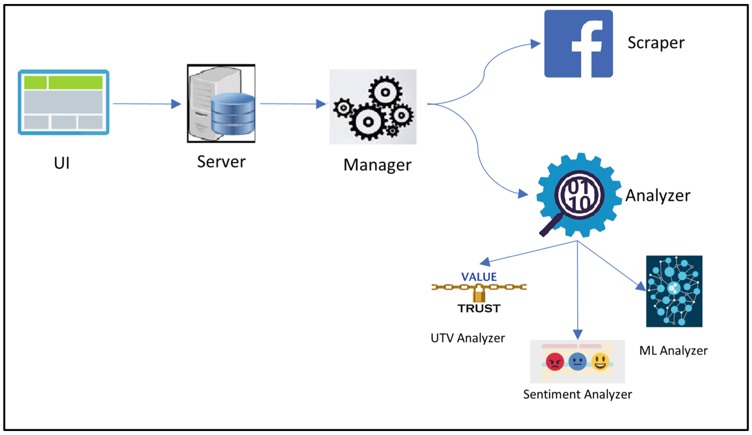


Fig. 2. System architecture for the implementation of the ML Fake News analyzer

We used similar parameters for Page and Group, with slight changes in thresholds. For an account, the parameters are age of user account (*AUA*), total friends (*TF*), mutual friends (*MF*), and friendship duration (*FD*). The *FD* parameter can be calculated only when the user is connected, and they are Facebook friends with this account. If these requirements are not met, N/A is returned. For a group or page, we calculate age, total friends or likes/followers, and mutual friends. The difference is in the threshold for *TF*: in an account it's 23.82. For a page it's 25000, and for a group it's 50000. A trust value, a number between 0 and 1 is returned, and converted to percentages.

3 Experimental Evaluation

For the experimental part of this research, we used a dataset of a real Facebook network. Like in the Sentiment Analysis part, the test using our ML model can also be calculated for all URL types and is performed on the text for text/single post check, and on the topmost *numOfPosts* posts for the other cases, where we return the average of the scores. In this case, since our model is fined-tuned for checking fake news regarding Covid-19, we require that the text is Covid-19 related. Furthermore, we require at least one Covid-related post, in order to perform calculation. The analysis is thus done in two steps. First, we perform a simple check to confirm that the post is covid-related. This check is done in the Analyzer level. We have a list of words related to covid, such as COVID, vaccinations, masks, etc. We assume that if a post contains at least one word from this list, then the post is related to covid. In the second step, the post is evaluated using our neural network. Training was done in advance. We saved the trained model in a designated.pkl file, and when the program runs, this model is loaded and used to evaluate a given post. ML return a value between 0 and 1, zero stands for fake and one stands for real. This value is also converted to percentages.

For training we used a pre-trained BERT-based model – AlephBERT [24]. We fine-tuned AlephBERT by adding layers to the neural network. AlephBERT is a pre-trained Language Model (PLM), trained on unlabeled modern Hebrew data, which provides contextualized word representations.

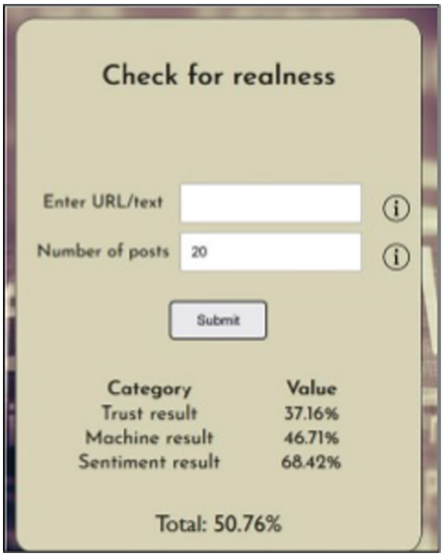
The purpose of the model is to provide a basis for language-specific tasks, that has a deeper understanding of the semantics of the language (Hebrew in this case). We added further training to this fine-tuned model, on 85% of a dataset that we created specifically for this task. We conducted basic cleaning of the data; removed punctuation, removed links, and transformed recurring words written in English such as 'FDA' to Hebrew. Finally, we evaluated the model on the remaining 15% of the data, achieving 76% accuracy. We fine-tuned the model to our specific task of Covid-19 related Fake News Classification. We added two layers to the model: the first is a *Dropout layer*, a technique that temporarily removes neurons (along with their connections) from the neural network and has been proven effective to avoid overfitting.

The output from this layer is fed into the next layer we added: a *Linear layer*, a fully connected layer which applies linear transformation to the incoming data. Finally, the output from Linear is further transformed by the sigmoid function, to return a single floating-point number between 0 and 1.

We trained this fine-tuned model on a labeled dataset that we created specifically for this task, consisting of Facebook posts and comments regarding Covid. We tried training with different values for the parameters epoch number, batch size and learning rate. The best results were achieved with 6 epochs, 16 batch size, and learning rate $3 \cdot 10^{-6}$. We evaluated the model's performance on the test portion of the data, which consisted of 51 posts. We used Binary Cross Entropy as the loss function. We achieved 76% accuracy, which is the highest accuracy score we have achieved in training, compared to training on this model with different parameter values, and compared to previous ML models we built for this task (SVM, CLF, LSTM).

We expect that an experiment on a larger dataset, will produce a higher accuracy score. Data was the biggest challenge we dealt with in the ML part of the project. There is no open Hebrew dataset regarding fake news. Initially we tried using datasets in English and deep translation in order to overcome this issue. However, we encountered problems while using this method, and obtained very poor results. There were significant differences between the posts in the English datasets and the Hebrew posts we examined and used to test the model. For example, in English the virus is usually referred to as 'Covid', 'Covid-19', 'Coronavirus', whereas in Hebrew it's referred to as 'Corona'.

Another big difference is in the content - many posts in English refer to things happening in countries that posts in Hebrew don't reference, and a lot of posts in Hebrew are about vaccine-related issues, a topic not widely covered in the English datasets. We tried to overcome some of the differences, for example by replacing all 'Covid-19' instances with 'Corona' instances. After still achieving poor results, we decided to change direction and work with a Hebrew dataset directly.


Check for realness

Enter URL/text

Number of posts 20

Submit

Category	Value
Trust result	37.16%
Machine result	46.71%
Sentiment result	68.42%

Total: 50.76%

Fig. 3. Results of an Anti-Vaccination page of the ML Fake News analyzer

Since there is no available Hebrew dataset for this (or related) purposes. We created our own small dataset of posts, comments, and statements in Hebrew. It is available on the GitHub repository of our project [25]. We used two common data augmentation techniques: random swap and random deletion. Our model randomly chose 7% of the sentences from the training set, and in each sentence randomly chose two words and swapped their positions.

This produced new sentences that were inserted to the training data. Then, 7% sentences were randomly chosen from the new training data, and from each sentence some words were deleted randomly (with a probability of 17% for each word to be deleted).

We can see an example of the analyzer results in Fig. 3 that gives the results of one of the Anti-vaccination pages in Hebrew after analyzing 40 of the pages' posts.

4 Conclusion and Future Work

In this research we presented a Trust-based model that uses context and user evaluation for preventing the propagation of Fake News. The model is based on attributes are hard to fake since they are built on real OSN user presence and real numerical assets. Other experiments can be done in different OSN-setting such as different categories, other sizes of networks, etc. Methodically the categorizing of Fake News instances, as explained in the beginning of this paper (Fabrication, Satire, propaganda, etc.), could lead to interesting research directions for example:

Is there a correlation between the type of Fake news and the parameters or methods that should be applied to identify it and/or a correlation between the trust attributes of fake news propagators and the type of Fake news they spread.

In future work we also intend to improve the ML result and the Sentiment Analysis result. Furthermore, the thresholds for trust value calculation for non-accounts will be explored. For ML, we will create a larger and more accurate dataset, by collecting more data and reviewing the labels with experts from the field.

Regarding sentiment analysis, it is agreed that our method of using translation and analyzing in English has limited results. Currently sentiment analyzers in Hebrew are not abundant, and they are either difficult to use or produce poor results. Work on such analyzers continues to develop and better these tools, so we expect that in the future it will be possible to perform a more accurate sentiment analysis.

References

1. Voloch, N., Nissim, P., Elmakies, M., Gudes, E.: A Role and Trust Access Control model for preserving privacy and image anonymization in Social Networks. In: Meng, W., Cofta, P., Jensen, C.D., Grandison, T. (eds.) IFIPTM 2019. IAICT, vol. 563, pp. 19–27. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33716-2_2
2. Voloch, N., Levy, P., Elmakies, M., Gudes, E.: An Access Control model for Data Security in Online Social Networks based on role and user credibility. In: Dolev, S., Hendler, D., Lodha, S., Yung, M. (eds.) CSCML 2019. LNCS, vol. 11527, pp. 156–168. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20951-3_14
3. Gudes, E., Voloch, N.: An Information-Flow Control model for Online Social Networks based on user-attribute credibility and connection-strength factors. In: Dinur, I., Dolev, S., Lodha, S. (eds.) CSCML 2018. LNCS, vol. 10879, pp. 55–67. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94147-9_5
4. Voloch, N., Gudes, E.: An MST-based information flow model for security in online social networks. In: The 11th IEEE International Conference on Ubiquitous and Future Networks (2019)
5. Voloch, N., Gudes, E., Gal-Oz, N.: Preventing fake news propagation in social networks using a context trust-based security model. In: Yang, M., Chen, C., Liu, Y. (eds.) NSS 2021. LNCS, vol. 13041, pp. 100–115. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92708-0_6
6. Shu, K., Sliva, A., Wang, S., Tang, J., Liu, H.: Fake news detection on social media: a data mining perspective. ACM SIGKDD Explor. Newsl. **19**(1), 22–36 (2017)
7. Tandoc, E.C., Jr., Lim, Z.W., Ling, R.: Defining ‘fake news’ a typology of scholarly definitions. Digit. Journal. **6**(2), 137–153 (2018)

8. Kumar, K.P.K., Geethakumari, G.: Detecting misinformation in online social networks using cognitive psychology. HCIS **4**(1), 1–22 (2014). https://doi.org/10.1186/s13673-014-0014-x
9. Levi, O., Hosseini, P., Diab, M., Broniatowski, D.A.: Identifying nuances in fake news vs. satire: using semantic and linguistic cues. arXiv preprint arXiv:1910.01160. (2019)
10. Vosoughi, S., Roy, D., Aral, S.: The spread of true and false news online. Science **359**(6380), 1146–1151 (2018)
11. Li, Y., Li, Y., Yan, Q., Deng, R.H.: Privacy leakage analysis in online social networks. Comput. Secur. **49**, 239–254 (2015)
12. Atallah, M.J., McDonough, C.J., Raskin, V., Nirenburg, S.: Natural language processing for information assurance and security: an overview and implementations. In: NSPW, pp. 51–65, September 2000
13. Tsoumas, B., Gritzalis, D.: Towards an ontology-based security management. In: 20th International Conference on Advanced Information Networking and Applications (AINA 2006), vol. 1, pp. 985–992. IEEE, April 2006
14. Wang, X., Tokarchuk, L., Poslad, S.: Identifying relevant event content for real-time event detection. In: 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014), Beijing, pp. 395–398 (2014)
15. May, C., Ferraro, F., McCree, A., Wintrode, J., Garcia-Romero, D., Van Durme, B.: Topic identification and discovery on text and speech. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 2377–2387, September 2015
16. Choudhary, A., Arora, A.: Linguistic feature based learning model for fake news detection and classification. Expert Syst. Appl. **169**, 114171 (2021)
17. Monti, F., Frasca, F., Eynard, D., Mannion, D., Bronstein, M.M.: Fake news detection on social media using geometric deep learning. arXiv preprint arXiv:1902.06673. (2019)
18. Helmstetter, S., Paulheim, H.: Weakly supervised learning for fake news detection on Twitter. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 274–277. IEEE, August 2018
19. Pierri, F., Ceri, S.: False news on social media: a data-driven survey. ACM SIGMOD Rec. **48**(2), 18–27 (2019)
20. Voloch, N.: Using Sentiment Analysis and context evaluation for preserving Trust-based privacy in Social Networks (2020). https://www.cs.bgu.ac.il/~voloch/FinalProjectReport.pdf
21. Cui, L., Wang, S., Lee, D.: SAME: sentiment-aware multi-modal embedding for detecting fake news. In: Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 41–48, August 2019
22. Voloch, N., Gal-Oz, N., Gudes, E.: A trust based privacy providing model for online social networks. Online Soc. Netw. Media **24**, 100138 (2021)
23. https://www.youtube.com/watch?v=uSPknKXaWJ4&ab_channel=voloch
24. Chen, Y.P., Chen, Y.Y., Lin, J.J., Huang, C.H., Lai, F.: Modified bidirectional encoder representations from transformers extractive summarization model for hospital information systems based on character-level tokens (AlphaBERT): development and performance evaluation. JMIR Med. Inform. **8**(4), e17787 (2020)
25. https://github.com/rotemit/FakeNewsDetector

Reinforcement Based User Scheduling for Cellular Communications

Nimrod Gradus[1,2](✉), Asaf Cohen[1](✉), Erez Biton[2](✉), and Omer Gurwitz[1](✉)

[1] Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Beersheba, Israel
gradusn@post.bgu.ac.il, {coasaf,gurewitz}@bgu.ac.il

[2] Parallel Wireless, Kefar Sava, Israel
ebiton@parallelwirelss.com

Abstract. Scheduling in cellular networks is one of the most influential factors in performance in wireless deployments such as 4G and 5G and is one of the most challenging and influential resource allocation tasks performed by the base station. It requires the handling of two important performance metrics, throughput and fairness. Fundamentally, these two metrics challenge one another, and maximization of one might come at the expense of the other. On the one hand maximizing the throughput, which is the goal of many communication networks, requires allocating the resources to users with better channel conditions. On the other hand, fairness requires allocating some resources to users with poor channel conditions. One of the prevalent scheduling schemes relies on maximization of the proportional fairness criterion that balances between the two aforementioned metrics with minimal compromise. Proportional fairness based schedulers commonly rely on a greedy approach in which each resource block is allocated to the user that maximizes the proportional fairness criterion. However, typically users can tolerate some delay especially if it boosts their performance.

Motivated by this assertion, we suggest a reinforcement-based proportional-fair scheduler for cellular networks. The suggested scheduler incorporates users' channel estimates together with predicted future channel estimates in the process of resource allocation, in order to maximize the proportional fairness criterion in predefined periodic time epochs. We developed a reinforcement learning tool that learns the users' channel fluctuations and decides upon the best user selection at each time slot in order to achieve the best fairness in throughput trade-off over multiple time slots. We demonstrate through simulations how such a scheduler outperforms the standardized proportional fairness. We further implemented the suggested scheme on a real live 4G base station, also known as an EnodeB, and showed similar gains.

1 Introduction

The desire to cope with the fast proliferation of services and applications such as Internet-of-Things (IoT) and machine-to-machine communications (M2M), coupled with the never-ending growth and complexity in global mobile data traffic, has driven the broadband cellular network industry (5G and the upcoming 6G) to rely on novel disciplines such as Machine Learning (ML) and Artificial Intelligence (AI). Resource allocation and in particular scheduling remains one of the biggest challenges of the cellular industry.

The scheduler in a cellular-based network, which determines which user(s) shall be given resources (e.g., channels, power) at each downlink and uplink data transmission opportunity, plays a fundamental role in every cellular generation base station [3]. Such scheduling decisions should be determined according to predefined performance criteria and goals, such as maximizing the overall rate, minimizing the average latency, supporting due date constraints, etc. (e.g., [1]). Many scheduling algorithms have been adapted to wireless systems which, for the sake of simplicity, are oblivious to the users' channel, e.g., Round Robin (RR) and Weighted Fair Queuing (WFQ) [17]. Even though such schedulers are simple and save the burden (and overhead) of acquiring such information, they compromise performance. Accordingly, modern cellular networks (3G and beyond have given rise to *Opportunistic schedulers* that take into account physical layer information, such as user channel states [1]. Since in a wireless network, users are not homogeneous and different users not only have different abilities, but they experience distinct channel conditions due to fading, multipath and interference, an approach that favors users based on their wireless channel quality and abilities can result in severe unfairness and even starvation of some users. Many schedulers adopt some fairness metric to avoid such significant differences between the throughput attained by various users, ensuring that users receive a "fair" share of the system resources. However, as maximizing the overall throughput can be at the expense of fairness, also fairness might come at the expense of the overall throughput. For example, balancing the throughput attained by all users can result in a severe sum throughput degradation, as a user with poor channel conditions can consume most of the resources. Furthermore, in contrast to maximizing the overall throughput metric, the fairness metric is ambiguous and can assume different definitions. Indeed, several fairness metrics have been devised over the years in general and have been adapted to communication networks in particular, e.g., max-min fairness criterion, Jain's fairness index, Weighted Fair Queuing (WFQ) [9]. One of the most acceptable fairness metrics, according to which many communication network schedulers operate, is the proportional fairness (PF) criterion that balances between the two aforementioned metrics, maximizing the aggregate throughput while distributing the resources "fairly" between the users.

While the problem of proportionally fair scheduling has been studied thoroughly, with many different variations and implementations, common schedulers maximize the proportional fairness metrics at each scheduling instance, based on current users' channel states and on the aggregate throughput each user attained so far. Specifically, the scheduler (at the cellular base station) utilizes the users' current channel condition feedback and a moving average of the throughput attained by each user to allocate resources to the user that maximizes the proportional fairness metric at each scheduling instance. However, typical QoS constraints do not require the optimization of any performance metric on each and every instance and typically measure such key performance indicators (KPI) only periodically. Accordingly, taking into account estimations of users' future channel condition, delaying allocation for a user with poor channel condition that maximizes the current proportional fairness metric, waiting for the user to get better channel condition within the KPI periodicity can improve the proportional fairness performance at the evaluation instances.

This study explores a PF downlink scheduler that is examined at the end of a number of scheduling instances, rather than maximizing performance after every instance. Accordingly, it not only examines the past received rates and current channel conditions, it also tries to predict future channel conditions. We utilize Reinforcement Learning (RL) for this purpose, specifically, Q-learning, which does not require knowing the transition probability distribution and hence it is termed a model-free algorithm. Q-learning's goal is to estimate the optimal action-state value, and iteratively update the values estimated, by storing them in a table. We evaluated the suggested RL-PF algorithm via simulation, showing its gains over the greedy PF scheduler and a round-robin-based scheduler (giving equal timeshare to each user) under various network scenarios. Then, we implemented the RL-PF algorithm in an operational LTE base station, evaluating the scheduling of users under different path losses, giving a proof of concept to utilize such a scheduler in a cellular base station.

Specifically, the main contributions of this study are: (i) We formalize the PF scheduling, which measures performance only at periodic scheduling instances. (ii) We devise an RL-based downlink scheduler that predicts and incorporates near future users' channel conditions to allocate each resource block to the user that is expected to maximize the PF metric at the end of the periodic scheduling instances. The suggested scheduler utilizes Q-learning to learn each user's channel fluctuations and select the user at each time slot accordingly. (iii) Based on the scheduling simulator we implement, which supports different downlink channel models to the connected users, we demonstrate that our algorithm outperforms the standard PF both in fairness and throughput. (iv) We implement the RL-PF algorithm in an operational LTE base station, with three connected users, showing that such an algorithm is practical and can work on real-life deployment.

2 Proportional Fairness (PF) Background

In this section, we briefly explain the proportional fairness (PF) metric and describe the conventional PF algorithm, which is commonly used, and to which we compare our results.

The proportional fairness criterion, which was suggested in the seminal paper by Kelly [10], introduced an alternative to the max-min fair metric that can sacrifice a considerable amount of system resources in order to improve the performance of the most inferior user even if just by a little. In the context of throughput distribution in wireless networks and in particular in this study, the proportional fairness criterion can be defined as follows: consider a network where K users share a single link. Each user, $k \in K$, is associated with rate $r \in R$, denoted as r_k, where R is the set of all possible rates a user can support. Proportional fair distribution is the one that maximizes:

$$\sum_{k=1}^{K} \log r_k \tag{1}$$

In contrast to Kelly's model, which considers that the service can be partitioned such that all users can receive allocations simultaneously, usually in communication networks there are atomic resource units that cannot be partitioned (see appendix, Sect. 8.2 for more details). Accordingly, the proportional fairness metric was modified to consider the average throughput a user has received at the end of a time interval. Denoting by T_k the average rate user k has received, the proportional fair distribution is the one that maximizes:

$$\sum_{k=1}^{K} \log T_k \tag{2}$$

In order to attain the proportional fairness criteria in 2, [18] suggested a user selection algorithm. The proposed algorithm relies on the users' channel quality in time slot t, which is translated to an instantaneous data rate $R_k(t)$ and a moving average of each user attained throughput $T_k(t)$. The scheduling algorithm chooses the user k^* which,

$$\arg\max_{k \in K} \frac{R_k(t)}{T_k(t)}$$

The average throughputs $T_k(t)$ are updated using an exponentially weighted low-pass filter

$$T_k(t+1) = \begin{cases} (1 - \frac{1}{t_c})T_k(t) + \frac{1}{t_c}R_k(t), & k = k* \\ (1 - \frac{1}{t_c})T_k(t), & k \neq k^* \end{cases} \tag{3}$$

[18] proved that the suggested proportional fair algorithm with time window $t_c = \infty$, maximizes 2. Note that real-life networks such as LTE and 5G measure performance (e.g., KPI) on a finite time scale, hence taking t_c to ∞ or even keeping it very high is usually not practical.

Since the adaptation of Kelly's proportional fairness metric to communication networks, several other studies have suggested different proportional fairness-based algorithms. For example In [2], an analysis of the gains in utilizing channel predictions for the proportional fair scheduler was made, assuming that future channel states are known (the estimates of the users' supported data rates for the current $L-1$ future time slots are available to the scheduler upon each time slot), and computed the optimal allocations for the finite L time slots, comparing it to the PFS greedy allocation.

3 Model Description

Consider a single cellular base station, supporting bidirectional data traffic to K users, however in this work, we focus on the downlink traffic. We assume that time is divided into periods of L time slots and we shall refer to such periods as cycles. We further assume that users' downlink transmission rates which are based on the user's channel quality and link adaptation module (see appendix, Sect. 8.2 for details), are known to the scheduler prior to each time slot. We assume that the wireless channel remains static within the time slot, as in [12],[15],[13]. Prior to each time slot, the scheduler selects a user and transmits to it at a certain rate which with some probability might fail. The rates values received by the users are reset at the beginning of each cycle and are evaluated at the end of the cycle, Fig. 1. We generally followed the conventional frequency division duplex (FDD) cellular resource structure, in which the time is slotted and divided into constant 1 ms intervals, where each slot is divided into atomic parts of resource units (Sect. 8.1 for more information).

The performance is measured at the end of each cycle based on the proportional fairness metric. Specifically, denoting the accumulated throughput of user k at the end of cycle i by T_k^i, i.e., $T_k^i = \sum_{j=1}^{L} T_k^i(j)$. The system's proportional fairness utility metric at the end of the i-th cycle is:

$$\mathbb{U}^i = \sum_{k=1}^{K} \log T_k^i \tag{4}$$

Since we mainly deal with the scheduling of a single cycle throughout this study, to ease notations, we will omit the cycle index when discussing a general cycle, i.e., $T_k(j)$ denotes the throughput attained by user k in the j-th time slot and T_k denotes user k's accumulated throughput at the end of the cycle.

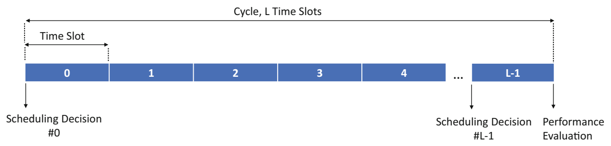


Fig. 1. Our model, where a cycle is divided into L time slots, and a scheduling decision is made at the beginning of each time slot, while we evaluate the performance at the end of the cycle.

4 Reinforcement Learning Proportional Fairness Based Scheduler

Since the performance is measured only at the end of each cycle, sometimes it is beneficial to delay an allocation for a user waiting to receive better channel conditions within the same cycle. To get some intuition, we exemplify this via a simple toy example. Assume two users are receiving data from the BS ($K = 2$). And assume each cycle has ten time slots ($L = 10$).

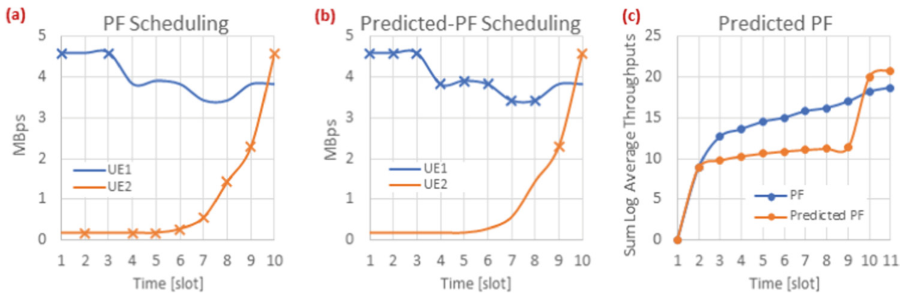


Fig. 2. The supported rates for two users with $L = 10$. A superimposed cross corresponds to an allocated slot. (a) Schedule according to the PF algorithm. (b) Schedule according to the Predicted PF algorithm. (c) Sum log throughputs of PF and predicted PF at the end of each time slot.

The Figures in 2 depict the supported rates for two users (UE 1 and UE 2, depicted by the blue and orange curves, respectively) over a single cycle. The asterisks in figure (a) depict the user selected at each time slot according to the PF algorithm described in Sect. 2, e.g., the first slot is allocated to UE-1, and the second slot is allocated to UE-2. Figure (b) time-slot allocation is based on a scheduler that postpones the allocation for UE-2, waiting for it to attain better channel conditions. Figure (c) is the sum log throughput of the two users (Eq. 2) after every time slot. As can be seen in the example, allocating slots 2 and 4–8 to UE-2 by the PF scheduler provides a better instantaneous gain in Eq. 2, however, allocating it slots 9–10 when it has better channel conditions, attains higher PF utilization at the end of the cycle. Additional motivation can be found in [2] that showed that the availability of future channel estimations could help improve the proportional fairness criterion.

Maximization of $\mathbb{U}$, as defined in Eq. 4 encounters two main challenges: i) it requires the scheduler to know the channel condition of the users throughout the entire cycle, *a priori*. ii) it requires an exhaustive search over all users' scheduling possibilities. To overcome both challenges in this study we utilize a Reinforcement Learning scheme.

Reinforcement learning associates actions to states to maximize a numerical reward in an uncertain environment. Accordingly, reinforcement learning is characterized by the tuple: agent, environment (states), actions and rewards. The agent is the one carrying out the actions, in our case the scheduler, which is deployed at the base station. The environment is characterized by the state space, which consists of all possible system setups. The action space includes the possible actions the agent can take and the policy that describes the agent's strategy to associate actions to states. In the sequel, we formalize the main components in the PF scheduling setup.

- The state space is defined by a K element vector, an entry for each user, in which each entry comprises of the 2-tuple $(T_i(n), CQI_i(n))$, wherein $T_i(n)$ represents the accumulated throughput user i has attained so far in the cycle (at the beginning of time slot n), and $CQI_i(n)$ represents user i channel condition at the beginning of time slot n. In this respect, there is a set of states associated with each time slot in the cycle. Specifically,

$$\mathbb{S}_n = \{(T_1(n), CQI_1(n))(T_2(n), CQI_2(n)), \ldots (T_K(n), CQI_K(n))\}$$

  $\forall 1 \leq n \leq L$, denotes the set of states that are associated with the beginning of time slot n.
- The action space, denoted by $\mathbb{A}_n$ is defined by the designated user equipment (UE), which will be scheduled at the beginning of time slot n, and will be provided with the whole bandwidth. In the problem we have herein, the challenge is that the action (which UE to schedule next) taken at each time slot should be evaluated based on the reward attained in the future (delayed rewards).
- The reward, denoted by $\mathbb{R}$, is calculated based upon the system's proportional fairness criterion, also formulated in Eq. 4 ($\mathbb{R} = \sum_{k=1}^{K} \log T_k$), at the end of each scheduling cycle. It is worth noting that instantaneous rewards are not given after each intermediate action (before the cycle has ended).

The reinforcement learning algorithm we utilized in this paper is Q-learning, a model-free learning algorithm that learns the value of each action for each state. In order to explore different scheduling decisions, the scheduler either selects a user based on the already learned policy for the current state or, with probability ϵ tries to schedule a random user. After every scheduling decision, a Q-value, corresponding with the pair (state, action), at time slot n, denoted by, $\mathbb{Q}_n(\mathbb{S}_n, a_n)$, is calculated and stored in the Q-table. The calculation of the aforementioned value is based on the previous value and the maximal value of the next state multiplied by the discount rate, γ, formalized as, $\mathbb{Q}_n(\mathbb{S}_n, a_n) = \mathbb{Q}_n(\mathbb{S}_n, a_n) + \alpha[\mathbb{R}_n(\mathbb{S}_n, a_n) + \gamma \max \mathbb{Q}_{n+1}(\mathbb{S}_{n+1}) - \mathbb{Q}_n(\mathbb{S}_n, a_n)]$. The exploration/exploitation parameter, ϵ, in the algorithm is based on the ϵ*−Greedy* policy, which is a well-known policy in reinforcement learning, [16]

Algorithm 1 depict a pseudo code of the suggested Q-learning algorithm.

Algorithm 1. Q-Learning Scheduler

```
1: Initialize Q-table
2: while Training do
3:     Update $\mathbb{S}_n$ with UEs' CQI reports
4:     if $\mathbb{S}_n$ not in Q-table then
5:         Add $\mathbb{S}_n$ to Q-table
6:     end if
7:     With probability $\epsilon$ select a random UE to schedule
8:     Otherwise select the UE that corresponds to $a_k = \arg\max_a \mathbb{Q}(\mathbb{S}_n, \mathbb{A})$
9:     Allocate DL transmissions to UE k
10:    Update UE k throughput with the corresponding rate, $T_k = T_k + r_k$
11:    if $n < L$ then
12:        proceed to next time slot $n = n + 1$
13:        Receive CQI reports
14:        $R = 0$
15:        Transit to next state $\hat{\mathbb{S}}_n$
16:    else if $n = L$ then
17:        Calculate reward $R = \sum_{k=1}^{K} \log T_k$
18:        Initialize $T_k,\ \forall k \in K$
19:    end if
20:    Update $\mathbb{Q}(\mathbb{S}_n, a_k) = \mathbb{Q}(\mathbb{S}_n, a_k) + \alpha[R + \gamma \max \mathbb{Q}(\hat{\mathbb{S}_n}, \hat{a}) - \mathbb{Q}(\mathbb{S}_n, a_k)]$
21: end while
```

The basic blocks in the Q-learning scheduler that we implemented are as follows; the algorithm works by first deciding which UE to schedule at each time slot through the exploration/exploitation policy (lines 7–9), either choosing a random UE or choosing the UE that maximizes the reward as was learned. Upon downlink transmission to the chosen UE, the next state is updated with the throughput and new CQI reports observed (lines 10–15). The reward calculation is made at the end of the scheduling cycle, i.e., there is no reward calculation in the mid of a cycle and therefore the reward is equal to zero. (lines 16–18) depicts the state initialization for the next cycle. The Q-value calculation is found at line 20 where, the current Q-value for time slot n is updated with the current reward together with the difference between the maximum possible discounted future Q-value and the current Q-value, adjusting the Q-value steps in the direction of maximum reward at the end of the cycle. Worth noting that the reward calculated at the end backward-propagates to all states that were involved in the cycle.

5 Simulations

In this section, we evaluate via simulations the performance of the suggested RL-based scheduler under different parameters and configurations. We start by presenting the simulation setup.

5.1 Simulation Setup

We devised a python-based simulation platform that was utilized to implement the suggested reinforcement-learning algorithm (Q-learning) for the downlink scheduling. We considered a single base station and assumed K UEs were connected; all were fully downlink backlogged (i.e., the base station always had backlogged data to each UE). We also assumed that a single UE was scheduled per time slot. We assumed a time-varying block-fading channel model, in which the channel of each user was modeled as a two-state Markov chain as introduced by Gilbert [8] and Eliot [7], and that the users supported two rates, $r_1, r_2 \in R$, where R is the set of rates applicable for data transmission, which were the same for each user. We followed the usual notation of a good (G) and bad (B) state that corresponded with a user's supported rate, i.e., without loss of generality, we assumed that $r_1 > r_2$; hence, r_1 was denoted as good and r_2 as bad channel states. The channel transition probabilities between the channel states $Prob(G \rightarrow B)$ and $Prob(B \rightarrow G)$ were denoted by $1-p_k$ and $1-q_k$, respectively. The channel state was assumed to remain fixed during the course of a time slot (transmission). We explored various p_k and q_k, chosen independently between the users. We assumed that the scheduler knew the channel state of each user before each schedule decision (see Sects. 3 and 10 for details) and transmitted at a rate appropriate for this channel state. In the first part of the simulations, we assumed that each transmission rate was appropriate to the channel state; hence all transmissions were successfully received. In the second part, we extended the simulations to more realistic scenarios where downlink transmissions weren't always received successfully, and there was some probability for an erroneous transmission. We assumed cycles of L time slots each and ran each setup for at least 1000 cycles. The results presented the average PF attained throughout the simulations.

We compared our algorithm to the prevalent opportunistic PF scheme presented in Sect. 2. As baselines, we compared our approach to the traditional round-robin scheme, in which the users were allocated slots sequentially in a predetermined order, regardless of channel condition or any fairness criteria, and to the optimal scheduling, which searches a-posteriori at the end of each cycle for the scheduling that would have maximized the PF criterion (4) based on the attained users' channel states. Note that the optimal scheduling is not feasible as it needs to determine which UEs should have been scheduled in the already finished cycle (in retrospect) when all users' channel state instantiations are already known to the BS. Furthermore, note that the optimal scheduling is also not scalable.

Besides the average PF metric $(Avg_i(\sum_{k=1}^{K} \log T_k^i))$ we also present the average accumulated user's throughput per cycle $(Avg_i(\sum_{k=1}^{K} T_k^i)$. Even though the objective is to maximize the PF criterion, we also present the Jain's fairness index, due to its popularity as a common fairness criterion:

$$Jain's\ index = \frac{(\sum_{k=1}^{K} T_k)^2}{K \times \sum_{k=1}^{K} {T_k}^2}.$$

Due to the enormous state space involved, we shall focus on a small number of users starting from only two users, which is extended later to three and four users.

Two UEs Scenario. In order to gain some insight, we start with a simple two UE setup in which the channel of one user (UE-1) is constantly in a good channel state, i.e., $p_1 = 1$, and $q_1 = 0$, while the second user (UE-2) alternates between good/bad channel state. Specifically, $p_2 = 0.3$, and $q_2 = 0.7$. The downstream traffic for both UEs is fully backlogged. Figure 3 compare the average sum log throughput, average Jain's fairness index and average cell throughputs, respectively, for a scheduling cycle of $L = 5$. The average of all three metrics was calculated over all scheduling cycles.

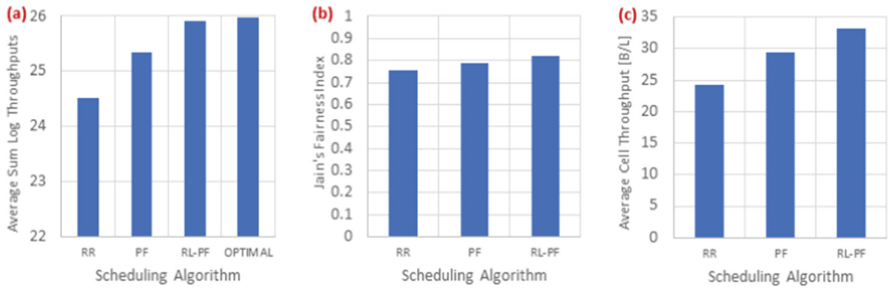


Fig. 3. 2 UE setup with $L = 5$. (a) Sum log throughput. (b) Jain's fairness index. (c) Average cell throughput.

Figure 3(a), shows a 3% increase in the PF metric of the RL algorithm compared with the standard PF, indicating that even in a small-scale environment, RL scheduling brings some gains over PF. A comparison with the optimal scheduler shows that the RL scheduler attains nearly the performance of the optimal one. Examining the detailed results shows that the RL scheduler attained the optimal schedule in over 95% of the scheduling cycles. Figure 3b shows that also with respect to the Jains fairness index, the RL scheduler outperforms the PF scheduler. Figure 3c compares the overall cell throughput which shows that using RL scheduling results in 12% aggregated cell throughput increase compared with the standard PF scheduler, and about 36% increase over the RR scheduler, which doesn't take into consideration the UEs' channel conditions or the attainable rates.

We leverage this simple example to better understand the gains from utilizing RL, in this setup, in which one of the UEs always experiences good channel conditions, while the other mostly experiences bad channel conditions and only occasionally experiences good channel conditions. Accordingly, the scheduler needs to try and schedule the second user whenever it experiences good channel conditions, yet no more than two times per cycle. Accordingly, the scheduler can postpone allocations to UE-2 waiting for it to have good channel conditions. If UE-2 has received no allocations in the first three slots it will get the last two slots regardless of its channel condition. In contrast, the PF algorithm will always allocate at least one of the three first slots to UE-2, even if in all of them UE-2 experienced bad channel conditions. Note that in the case that after three bad channel conditions the last two slots UE-2's channel was good, the scenario is similar to the one presented in Fig. 2. Similarly, if UE2's channel was in good condition in the first two slots, the RL scheduler will assign both slots to it while the PF scheduler will assign only one of the to it and the other to UE 1.

Four UEs Scenario. Next, we examine a four UE comparison for three setups, where $L = 5$. (i) UE-1 and UE-2 are kept at constant good state, i.e., $p_1 = p_2 = 1$, UE-3 is kept at a constant bad state, $q_3 = 1$, and the last UE alternates between good and bad with probabilities, $p_4 = 0.3$, and $q_4 = 0.7$. (ii) UE-1 was kept at a constant good state, $p_1 = 1, q_1 = 0$, while UE-2 and UE-3 alternate between good and bad, i.e., $p_2, p_3 = 0.3$ and $q_2, q_3 = 0.7$, while UE-4 is kept at a constant bad state, i.e., $q_4 = 1$. (iii) UE-1 is kept at a constant good state, i.e., $p_1 = 1$ while UE-2, UE-3 and UE-4 alternate between good and bad with $p_2 = p_3 = p_4 = 0.3$ and $q_2 = q_3 = q_4 = 0.7$.

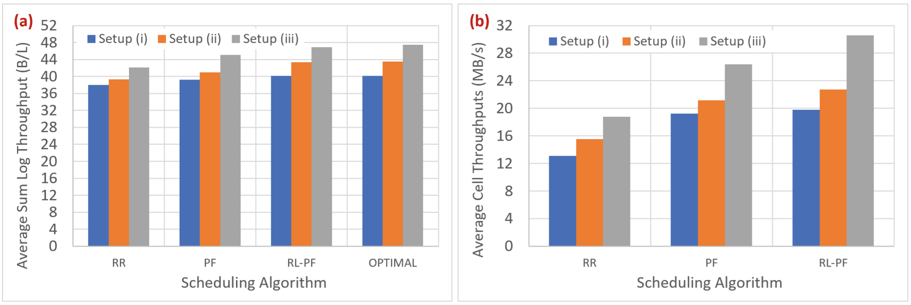


Fig. 4. 4 UE setup with $L = 5$ for three different setups. (a) Average sum log throughput. (b) Average cell throughput.

Figures 4(a) and 4(b) depict the average sum log throughputs and the average cell throughput, respectively, for all three setups. Figure 4(a), which depicts the average sum log throughput, clearly shows the gain of using RL scheduling when a 3%, 5%, and 6% increase was found for each setup, respectively. Figure 4(b), which shows the results of the average cell throughput, indicates a 3%, 8% and 16% gain over the standard PF for each setup, respectively. As in the two UEs scenario, the RL-PF scheduler attains almost the same PF results as the optimal one.

5.2 Unsuccessful Transmissions

In the first part of the results, we assumed that all downlink transmissions were received successfully. In this part of the results, we considered a more realistic scenario in which each downlink transmission had a certain probability of being unsuccessful. We denoted the probability for successful transmission by $P_k^{succ} < 1$, for UE k. As in operational cellular networks, we assumed that the feedback (ACK/NACK) was delayed and was received only after the cycle had ended. Results presented in this section took into consideration the throughputs attained by the UEs at the end of each cycle, i.e., considered only the successful transmissions.

In this part, we examined the simple two UE scheduling in which UE-1 always experienced a good channel condition and In contrast, UE-2 channel state alternated between good and bad states with probabilities, $p_2 = 0.3$, $q_2 = 0.7$. We assumed that the transmissions of UE-1 were always successful, i.e., $P_1^{succ} = 1$ and examined two different success probabilities for UE 2, (i) $P_2^{succ} = 0.3$ (most of the downlink allocations for UE-2 were not received successfully). (ii) $P_2^{succ} = 0.9$ (most UE-2's transmissions were received successfully). Note that for the optimal scheduler to be able to compute the attainable rates for all possible schedules, at each time slot, we simulated a transmission by each UE and noted, based on the success probability of each UE, whether it was successfully received.

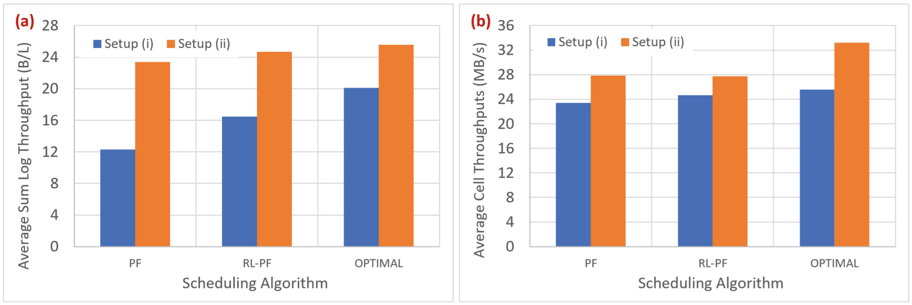


Fig. 5. 2 UE setup with $L = 5$ and probability for unsuccessful transmissions for two different setups. (a) Average sum log throughput. (b) Average cell throughput.

Figure 5 depict the average sum log throughputs and the average cell throughput for both setups. Figure 5(a) depicts, that with respect to the PF metric (average sum log throughput), the RL-based algorithm outperforms the PF algorithm (25% gain in setup (i) and 6% gain in setup (ii). However, as can be seen in Fig. 5(b), the PF metric gain came at the expense of the average cell throughput, i.e., the PF scheduling resulted in a higher cell throughput than the RL based scheduler for both setups. The reason for this degradation relies on the fact that the RL-based scheduler learns that UE 2's transmissions are not always successful hence allocates more resources (time slots) to it; some of these extra transmissions are at a low rate, and some will also fail (especially in setup (i)). On the other hand, the PF-based scheduler ignores the failed transmissions, hence favors UE 1, whose transmissions are always successful and at a high rate. Accordingly, even though the proportional fairness criterion tries to balance fairness and channel utilization, a significant number of resources are still spent on the less advantaged user(s). As can be seen, both schedulers are far from attaining the optimal scheduler results, which can examine all schedule possibilities (2^5 different schedules) and the outcome for each possible schedule, concerning the number of successful transmissions for each UE.

6 Implementation

6.1 Testbed

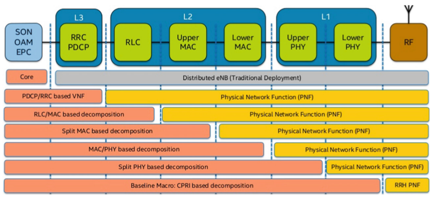

L3
L2
L1
SON
OAM
EPC
RRC
PDCP
RLC
Upper
MAC
Lower
MAC
Upper
PHY
Lower
PHY
RF
Core
Distributed eNB (Traditional Deployment)
PDCP/RRC based VNF
Physical Network Function (PNF)
RLC/MAC based decomposition
Physical Network Function (PNF)
Split MAC based decomposition
Physical Network Function (PNF)
MAC/PHY based decomposition
Physical Network Function (PNF)
Split PHY based decomposition
Physical Network Function (PNF)
Baseline Macro: CPRI based decomposition
RRH PNF

Fig. 6. The different splits in RAN architecture, split 8 is between the RF and upper layers

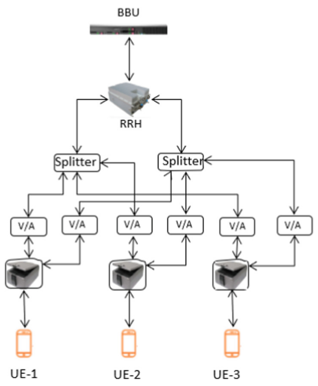


Fig. 7. Testbed illustration

We have implemented the Q-learning RL-PF algorithm inside an operational LTE base station with three connected users. Details of the testbed and results are discussed in brief next.

Implementation testbed comprises a 10 MHz bandwidth single cell running 4G network, specifically, option 8, where there is a split between the RF, also known as the remote radio head (RRH), and the upper layers, e.g., PHY, MAC, which are implemented in a COTS server, also known as baseband unit (BBU), Fig. 6. The scheduling decisions are made in the scheduler entity that resides in the MAC layer and is configured to schedule one user in each time slot. The RRH is connected to two RF splitters, which allows us to connect the 2×2 RRH to three RF cages where the users are placed. Variable attenuators (V/A) are placed in each RF connection in order to control the users' path loss to the LTE cell. The testbed architecture is illustrated in 7. In our system, we examined the described setup, where UE-1 is in almost perfect channel condition, path loss of

60 [*db*], UE-2 is experiencing 117 [*db*] path loss, and UE-3 is experiencing a path loss of 123[*db*]. Apart from our RL-PF scheduler, we have implemented the PF scheduler, as formulated in Sect. 2 as well. UDP downlink data is generated for each UE such that all UEs are fully backlogged.

6.2 Adaptation of RL-PF Model for Implementation

As described in Sect. 4, a state in our RL-PF scheduler consists of the UEs' CQI reports, which indicate the channel quality of each UE. The assumption was that such reports were available to the scheduler prior to every time slot. Such an assumption is invalid when dealing with an actual base station, where CQI reports for each UE are received every 80 ms. However, common operational base stations utilize implicit channel feedback in the forms of ACKs/NACKs, which are received for each transmitted packet and give an indication to the rate adaptation module with respect to the current supported transmission rate. Accordingly, we cope with the aforementioned CQI reporting granularity by re-defining the RL-PF state presentation to include the transport block index (ITBS) instead of CQI, where ITBS is the output of the link adaptation mechanism corresponding to the CQI reports and ACK/NACKs feedback. We further elaborate on this in Appendix 8.2.

6.3 Implemented RL-PF Learning Methodology

The learning process is performed offline after a measurement session where the UEs' ITBS and throughputs at each time slot were gathered. ACK/NACK feedback is considered and is received at the scheduler 4 ms after the downlink transmission, as defined by the 4G standard. Therefore, after gathering all feedback from each UE, actual throughputs at the end of a scheduling cycle are collected, and calculation of PF criteria is made and used for the learning process.

6.4 Implementation Results

We implemented both schedulers, RL-PF, and PF schemes. We kept the cycle length of 5 slots ($L = 5$), and compared the results attained by the RL-based scheduler with the PF and optimal scheduling scheme for the setup described in 6.1.

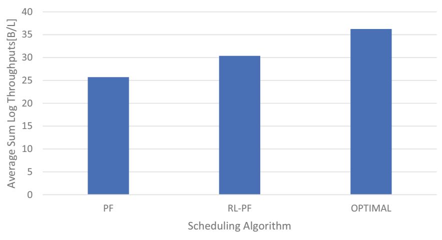


Fig. 8. RL-PF vs PF vs Optimal , average sum log throughputs, where $L = 5$.

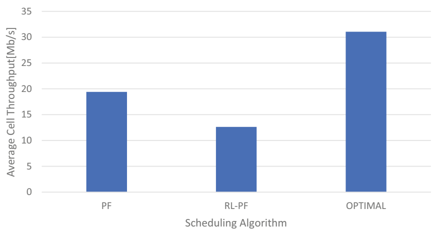


Fig. 9. RL-PF vs PF vs Optimal, average cell throughput, where $L = 5$.

Figures 8 and 9 depict the average sum log throughputs and the average cell throughput, respectively. The basic trend for both metrics agrees with the results presented in the simulations for two users with a probability for unsuccessful transmission. Specifically, Fig. 8 which presents the average sum log throughputs, depicts an 18% gain over PF in the measured setup. On the other hand, the average cell throughput presented in Fig. 9 shows that the PF scheme has a higher cell throughput, as also observed in the simulations 5.2. This is explained by the fact that the RL-PF allocates extra resources to the UEs, which tend to lose packets. These UEs have inferior channel conditions, hence transmit at lower rates, at the expense of allocation to the superior UEs, contributing much more to the overall cell throughput. Explicitly, the RL-PF scheduling strategy allocates more slots to the UEs that are more likely to return a NACK, hence, in our setup UE-1 will get only one slot (almost perfect channel conditions, i.e., very low block error rate), while the other two UEs will share the other four slots, as those UEs have a higher probability of failure due to their path loss. In contrast, the PF scheduler will allocate at least two slots to UE-1 and share the other three slots between the other two UEs. Accordingly, the effect is twofold (i) PF allocates more slots to UEs with higher data rates. (ii) PF allocates more slots to UEs with higher success probabilities hence a higher number of utilized slots. Moreover, the downlink link adaptation (DLLA) mechanism, explained in Appendix 8.2, is responsible for converging the users to a target block error rate (which is, as per design, the same for all users) and therefore, UE-2 and UE-3 will have the same probability of successful transmission, however, with different transmission rates. With that in mind, RL-PF will learn to allocate two slots to UE-2 and UE-3. Note that since the RL-based scheduler strategy is based on the anticipated loss probabilities in some cycles it can lose with respect to both metrics. For example, when all five slots were successfully transmitted, the PF scheduling policy outperforms the RL based on both the sum log throughputs and the aggregate cell throughput metrics. On the other hand, based on the PF scheduler, the BS allocates only a single transmission (time-slot) to one of the UEs with inferior channel conditions, which occasionally will fail. Accordingly, this UE attains zero throughput on many cycles, which is unacceptable in terms of the proportional fairness criterion. We note that in order for the sum log throughput results to be in scale, for these unfair PF slots, we assumed that the $logT$ of the UE that received $T = 0$ contributes 0 to the sum rather than minus infinity.

7 Conclusions

This work presents a novel PF scheduling algorithm that estimates and predicts users' future channel rates, specifically, utilizes a reinforcement learning tool, Q-learning, for optimal user selection allocation at each time slot. We presented a simulation framework that consists of different scenarios, specifically, different number of users to be scheduled. We have also implemented a POC, showing the achievable gains with a real base-station and users, which both show results that outperform other scheduling algorithms, such as PF, in the latter, sometimes at the expense of maximum throughput. We believe that our work opens a gateway for better practical schedulers that may utilize ML in their scheduling decisions. Future research is recommended for scaling our algorithm to a higher number of users and time slots, utilizing more complex ML concepts such as deep learning

8 Appendix

8.1 LTE Basic Terms

In this study we generally follow the conventional frequency division duplex (FDD) cellular resource units, in which the time is slotted into frames, and each frame is divided into constant 1 ms intervals, denoted as sub-frames. Each subframe is divided into parts termed physical resource resource blocks, which we shall refer to as simply resource blocks. Each such resource block comprises a bandwidth and time duration, e.g., in LTE each resource block comprises 12 sub-carriers in the frequency domain and 14 OFDM symbols in the time domain.

8.2 Downlink Link Adaptation (DLLA)

As mentioned earlier, opportunistic scheduling, e.g., proportional fairness, takes into consideration the users channel quality reports for better scheduling decisions. In particular, note that in the algorithm presented above, in order for the scheduler to select the user according to Eq. $\arg\max_k \frac{R_k(t)}{T_k(t)}$ it needs to know the instantaneous rates of all users. In wireless networks, these channel states of users are attained via reports indicating the users' supported rates for transmission. Furthermore, each practical system supports only a finite set of rates. Link Adaptation is the mechanism where the users' transmission code rates and modulation schemes are selected based on the channel conditions.

In this section, we briefly explain the concepts and processes of DLLA that is utilized in simulations and experimental results for scheduling using RL. since in the evaluation part both in the simulations and experimental results we follow a typical LTE DLLA, in the following subsection we will provide a technical description of the DLLA we utilized. Our description follows the common terminology and the accepted acronyms hence it is somewhat cumbersome.

The DLLA process is a crucial part of current wireless communication systems. Such technique increases the data rate that can be reliably transmitted [4] and has been adopted as a core feature in cellular standards such as LTE. The LA role in the MAC layer of the base station (BS) is to suggest the scheduler an appropriate modulation and coding scheme (MCS) to be used in the next transmissions to a certain user equipment (UE) in order to keep the block error rate (BLER) below a target. The proposed MCS is signaled from the UE by means of channel quality indicator (CQI) in the form of reports it sends to the BS, [14]. Afterwards, the BS uses a pre-calculated table for the mapping of CQI to a transport block size index (ITBS), an integer ranging from 1–26, which is used in the decision of the transport block (TB) size to be transmitted to the UE. The TB size is also determined by the number of physical resource blocks (PRBs) which can be allocated to the UE. In LTE the radio resources are allocated in the time/frequency domain. In particular, the time is slotted into intervals of 1 ms corresponding to 14 OFDM symbols. and in the frequency domain, the total bandwidth is divided into sub-channels of 180 kHz, each one with twelve consecutive and equally spaced OFDM sub-carriers. A time/frequency radio resource spanning over 1 ms time slot/14 OFDM symbols and twelve consecutive sub-carriers is called a physical resource block(PRB), or just RB, and corresponds to the smallest radio resource unit that can be assigned to a user for transmission. As the sub-channel size is fixed, the number of RBs varies according to the system bandwidth configuration, and it is the scheduler's decision to divide the total number of RBs to each scheduled UE in the time slot. The ITBS, together with the number of RBs that are allocated to the UE are mapped to the size of the TB.

The CQI reported by the UE on a per transmission time interval (TTI) basis, delivers information on how good/bad the downlink communication channel is. The UE's measurement of CQI depends solely on the chipset vendors and is derived from UE's measurement of the reference signals transmitted by the BS. The reference signals received power (RSRP) that is measured by the UE is than used to calculate the link quality metric (LQM) which quantifies the quality of the downlink and is used to determine the CQI. The LQM that is mostly used in LTE is the exponential effective SNR mapping (EESM) [5]. The process of selecting the most suitable MCS based on the link quality measurements is called inner loop link adaptation (ILLA) [6].

Due to various errors in the CQI measurements of the UE, the delay in the reporting process and deviations from the assumed channel conditions, e.g., multi-path environment, UE speed [11], a compensation process is needed and called outer loop link adaptation (OLLA). The correction of OLLA is based on the hybrid automatic repeat request (HARQ) feedback and is depicted as follows, the mapped ITBS from the UE's CQI report, defined as, $ITBS(CQI)$, is updated by a margin, $ITBS_{margin}$, for each received positive/negative acknowledgment (ACK/NACK) from the UE. When an ACK is received, $ITBS_{margin}$ is decreased by Δ_{down}, and when a NACK is received, the margin is increased by, Δ_{up}. The ratio $\frac{\Delta_{down}}{\Delta_{up}}$ is controlled by the target BLER that OLLA is designed to converge to, given by

$$\frac{\Delta down}{\Delta_{up}} = \frac{BLER_T}{100 - BLER_T}$$

Intuitively, if $BLER_T$ is set to 10%, this means that the user should receive at least 90% successful downlink transmissions. As explained the OLLA process is formulated as such,

$$ITBS = ITBS(CQI) - ITBS_{margin}$$

$$ITBS_{margin} = \begin{cases} ITBS_{margin} - \Delta_{down} & \text{if ACK} \\ ITBS_{margin} + \Delta_{up} & \text{if NACK} \end{cases} \quad (5)$$

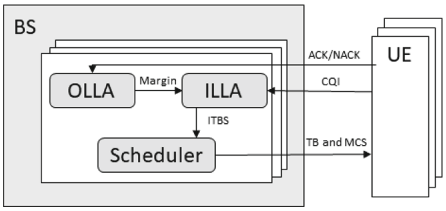


Fig. 10. DLLA block diagram

References

1. Asadi, A., Mancuso, V.: A survey on opportunistic scheduling in wireless communications. IEEE Commun. Surv. Tutor. **15**(4), 1671–1688 (2013)
2. Bang, H.J., Ekman, T., Gesbert, D.: Channel predictive proportional fair scheduling. IEEE Trans. Wirel. Commun. **7**(2), 482–487 (2008)
3. Capozzi, F., Piro, G., Grieco, L.A., Boggia, G., Camarda, P.: Downlink packet scheduling in ITE cellular networks: key design issues and a survey. IEEE Commun. Surv. Tutor. **15**(2), 678–700 (2012)
4. Chung, S.T., Goldsmith, A.J.: Degrees of freedom in adaptive modulation: a unified view. IEEE Trans. Commun. **49**(9), 1561–1571 (2001)
5. Donthi, S.N., Mehta, N.B.: An accurate model for EESM and its application to analysis of CQI feedback schemes and scheduling in ITE. IEEE Trans. Wirel. Commun. **10**(10), 3436–3448 (2011)
6. Duran, A., Toril, M., Ruiz, F., Mendo, A.: Self-optimization algorithm for outer loop link adaptation in ITE. IEEE Commun. Lett. **19**(11), 2005–2008 (2015)
7. Elliott, E.O.: Estimates of error rates for codes on burst-noise channels. Bell Syst. Tech. J. **42**(5), 1977–1997 (1963)
8. Gilbert, E.N.: Capacity of a burst-noise channel. Bell Syst. Tech. J. **39**(5), 1253–1265 (1960)
9. Huaizhou, S.H.I., Venkatesha Prasad, R., Onur, E., Niemegeers, I.G.M.M.: Fairness in wireless networks: issues, measures and challenges. IEEE Commun. Surv. Tutor. **16**(1), 5–24 (2013)
10. Kelly, F.: Charging and rate control for elastic traffic. Eur. Trans. Telecommun. **8**(1), 33–37 (1997)
11. Morales-Jimnez, D., Scnchez, J.J., Gmez, G., Aguayo-Torres, M.C., Entrambasaguas, J.T.: Imperfect adaptation in next generation OFDMA cellular systems (2009)
12. Ouyang, W., Eryilmaz, A., Shroff, N.B.: Downlink scheduling over Markovian fading channels. IEEE/ACM Trans. Netw. **24**(3), 1801–1812 (2015)
13. Piazza, D., Milstein, L.B.: Multiuser diversity-mobility tradeoff: modeling and performance analysis of a proportional fair scheduling. In: Global Telecommunications Conference, 2002 (GLOBECOM'02), vol. 1, pp. 906–910. IEEE (2002)

14. Sesia, S., Toufik, I., Baker, M.: LTE-the UMTS Long Term Evolution: From Theory to Practice. Wiley (2011)
15. Shmuel, O., Cohen, A., Gurewitz, O.: Performance analysis of opportunistic distributed scheduling in multi-user systems. IEEE Trans. Commun. **66**(10), 4637–4652 (2018)
16. Tokic, M., Palm, G.: Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In: Bach, J., Edelkamp, S. (eds.) KI 2011. LNCS (LNAI), vol. 7006, pp. 335–346. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24455-1_33
17. Tsai, T.-Y., , Chung, Y.-L., Tsai, Z.: Introduction to packet scheduling algorithms for communication networks. Sciyo (2010)
18. Viswanath, P., Tse, D.N.C., Laroia, R.: Opportunistic beamforming using dumb antennas. In: Proceedings IEEE International Symposium on Information Theory, p. 449. IEEE (2002)

A Heuristic Framework to Search for Approximate Mutually Unbiased Bases

Sreejit Chaudhury[1], Ajeet Kumar[2], Subhamoy Maitra[2](✉), Somjit Roy[3], and Sourav Sen Gupta[4]

[1] Jadavpur University, Kolkata, India
[2] Applied Statistics Unit, Indian Statistical Institute, Kolkata, India
subho@isical.ac.in
[3] Department of Statistics, University of Calcutta, Kolkata, India
[4] Nanyang Technological University, Singapore, Singapore

Abstract. Mutually Unbiased Bases (MUBs) have varied applications in quantum information. However, obtaining the optimal number of MUBs is a challenging problem for different dimensions. The problem has received serious attention for several decades and still number of questions are unsolved in this domain. As optimal number of MUBs may not always be available for different composite dimensions, Approximate MUBs (AMUBs) received serious attention in recent time. In this paper, we present a heuristic to obtain AMUBs with significantly good parameters. Given a non-prime dimension d, we note the closest prime $d' > d$ and form $d'+1$ MUBs through the existing methods. Then our proposed idea is (i) to apply basis reduction techniques (that are well studied in Machine Learning literature) in obtaining the initial solutions, and finally (ii) to exploit the steepest ascent kind of search to achieve further improved results. The efficacy of our technique is shown through construction of AMUBs in dimensions $d = 6, 10, 46$ from $d' = 7, 11$ and 47 respectively. Our technique provides a novel framework in construction of AMUBs that can be refined in a case-specific manner. From a more generic view, this approach considers approximately solving a challenging (where efficient deterministic algorithms are not known) mathematical problem in discrete domain through state-of-the-art heuristic ideas.

Keywords: Mutually Unbiased Bases (MUBs) · Approximate Mutually Unbiased Bases (AMUBs) · Quantum Key Distribution · Dimension reduction · Singular Value Decomposition (SVD) · Gradient ascent

1 Introduction

MUBs form a very interesting and well studied mathematical structure that have applications in Quantum Key Distribution (QKD), dense coding, teleportation, entanglement swapping, covariant cloning and state tomography (see [6] and the references therein).

Definition 1. *Two orthonormal bases* $\mathbf{A} = \{|a_1\rangle, |a_2\rangle, \ldots, |a_d\rangle\}$ *and* $\mathbf{B} = \{|b_1\rangle, |b_2\rangle, \ldots, |b_d\rangle\}$ *of a d-dimensional Hilbert space* $\mathbb{C}^d$ *are Mutually Unbiased Bases (MUBs) if*

$$|\langle a_i|b_j\rangle| = \frac{1}{\sqrt{d}}, \qquad \forall\ i, j = 1, 2, \ldots, d. \tag{1}$$

MUBs over high-dimensional Hilbert spaces are of practical significance in quantum cryptology (see [1] and the references therein, and in particular to Quantum Key Distribution [3]), and construction of MUBs pose interesting research questions in combinatorics (see [6,9] and references therein). If $d = p_1^{n_1} p_2^{n_2} \cdots p_s^{n_s}$, it is known that the number of MUBs of the Hilbert space $\mathbb{C}^d$ is bounded by

$$\min\{p_1^{n_1}, p_2^{n_2}, \ldots, p_s^{n_s}\} + 1 \ \le\ \mathfrak{M}(d)\ \le\ d + 1. \tag{2}$$

Thus, if $d = p^k$ is a prime power (or a prime), there exist $\mathfrak{M}(d) = d + 1$ MUBs. The initial methods to construct such MUBs, based on Galois fields, are described in [18]. However, for the other dimensions, it is an extremely challenging question when the lower bound of $\mathfrak{M}(d)$ can be defeated. Even for d as small as $2 \times 3 = 6$, the smallest non-prime-power composite dimension, construction exists for only $2 + 1 = 3$ MUBs, the lower bound. No larger set of MUBs could be discovered to date for $d = 6$, let alone the upper bound $6 + 1 = 7$. This resulted in recent studies on an approximate version of the problem (see [14] and the references therein).

Definition 2. *Two orthonormal bases* $\mathbf{A} = \{|a_1\rangle, |a_2\rangle, \ldots, |a_d\rangle\}$, $\mathbf{B} = \{|b_1\rangle, |b_2\rangle, \ldots, |b_d\rangle\}$ *over a Hilbert space* $\mathbb{C}^d$ *are Approximate Mutually Unbiased Bases (AMUBs) if*

$$|\langle a_i|b_j\rangle| \le \epsilon, \qquad \forall\ i, j = 1, 2, \ldots, d, \tag{3}$$

for a certain upper bound ϵ *(a small constant less than* 1*) on the inner product of the basis vectors from the two orthonormal bases.*

For example, one may consider β-AMUBs [12], which is formally defined as follows.

Definition 3. *Two orthonormal bases* $\mathbf{A} = \{|a_1\rangle, |a_2\rangle, \ldots, |a_d\rangle\}$, $\mathbf{B} = \{|b_1\rangle, |b_2\rangle, \ldots, |b_d\rangle\}$ *over a Hilbert space* $\mathbb{C}^d$ *are* β*-Approximate Mutually Unbiased Bases (*β*-AMUBs) if*

$$|\langle a_i|b_j\rangle| \le \beta/\sqrt{d} \tag{4}$$

with β *upper-bounded by a certain constant.*

Exact theoretical bounds based on our techniques could be a very interesting work, but for the time being the performance of our heuristics are measured numerically through experiments. Thus formalizing and proving theoretical bounds on the approximation are not in the scope of this paper. However, the measures of closeness of AMUBs to MUBs are quantified by two well known measures as explained in Sect. 1.2.

Construction of AMUBs have been attempted using techniques in combinatorial designs and algebraic techniques (see [11,12,16] and the references therein), where the approximation to MUBs is measured by the dispersion of the values $|\langle a_i|b_j\rangle|$ from $\frac{1}{\sqrt{d}}$, the ideal case. For example, algebraic techniques are presented in [16] for construction of Approximate MUBs. Recently, combinatorial designs are used in [12] to construct β-AMUBs for $d = p^k(p^k+1)$, where β remains bounded by a certain constant. While several interesting properties for β-AMUBs have been explored in [12], such combinatorial constructions suffer from limitations when d is of certain forms. Particularly if $d = 2q$, where q is some power of odd prime, such combinatorial designs although gives a large number of AMUBs but fails to construct good β-AMUBs. That is why we consider $6, 10$ and 46 as our examples. The most encouraging results for dimension $d = 6$ are presented in [14]. In this case we have the lower bound of $2+1=3$ MUBs. This is extended through the construction of 4 AMUBs in [14] using a parametric structure. The structure is quite interesting as altering the parameters result in different sets of AMUBs and among those the best result is considered. However, there has been no theoretical advancement in this direction for over a decade, and the construction of such a parametric structure for AMUBs for any arbitrary dimension seems elusive. We also note that, there has been recent advancements in search of Mutually Unbiased Bases and the approximate ones through computational approaches [4], where the three broadly used numerical methods obtain relatable and coinciding results to that of [14]. However, no discussions have been made so far about the approximate version of the higher dimensional Mutually Unbiased Bases. That is where we explore the heuristic ideas that can work efficiently for higher dimensions.

1.1 Motivation, Contribution and Organization

Combinatorial constructions (see [11,12] and references therein) and algebraic as well as search based techniques (see [14] and the references therein) are trending techniques in AMUB literature. However, the strategies have their respective limitations too. The combinatorial constructions work on dimensions of certain kinds, and the computations for the rich but restricted class of unitary transformations explored in [14] are not applicable for higher dimensions. Thus, we explore alternative heuristics to construct AMUBs in any dimension with good computational efficiency.

We propose a generic approach in constructing AMUBs through dimension reduction, using Singular Value Decomposition (SVD) as the primary and initial component of our algorithm. Given an arbitrary dimension d, we start from d', the closest prime power larger than d. As the construction for $(d'+1)$ MUBs exist for prime power d', we apply dimension reduction to those MUBs to obtain a potential collection of AMUBs in dimension d. Finally, a heuristic search on this collection produces $(d+1)$ AMUBs in the d-dimensional Hilbert space $\mathbb{C}^d$. This approach works for any dimension d, and is quite efficient for higher dimensional spaces. Although SVD is quite prominent for its dimensionality-related applications in mathematics and machine learning [15], to the best of our knowledge, such an SVD-based dimension reduction technique has never been studied for constructions of AMUB. For a more detailed background, let us refer to Sect. 1.2, where we present the two measures used in this paper to determine the *approximation closeness* of AMUBs to MUBs. We also briefly explain the basics of dimension reduction using Singular Value Decomposition (SVD) here.

Let us now formally propose the framework. We devise two strategies for dimension reduction using the general idea of SVD – *merged* and *non-merged*. We treat the reduced bases obtained from the SVD routine as our initial solution set for prospective AMUBs, which will further be subjected to a steepest ascent kind of heuristic search, as proposed in [14]. It is thus natural that for $d = 6$, we do not obtain results as good as in [14], as the algebraic structure is not used at all. However, our results for $d = 6$ is encouraging and only slightly weaker than that of [14], though we relax the restrictions related to the parametrization and proceed with a general class of bases. More important is that, our proposed technique can be adapted easily in case of higher dimensions to construct more number of AMUBs exceeding the lower bound on MUBs. These we present in details in Sect. 2. The algorithm is presented in two parts – 'Creation of Bases' and 'Choice of Bases'. We also highlight the heuristic search technique in [14], as we subject the reduced bases obtained from our dimension reduction step to this search routine.

It is generally accepted that to construct large number of MUBs in $\mathbb{C}^d$, where $d = 2 \mod 4$ is quite difficult and in such dimensions using known construction methods for prime powers, we get only 3 MUBs. Hence we implement our techniques for dimensions of the form $d = 2p$, where p is a Sophie Germain prime [5] (that is, both p and $2p + 1$ are primes), as large number of MUBs are available in dimension $d' = d + 1 = 2p + 1$. In particular we consider the dimensions $d = 6, 10, 46$, where our proposed strategy yields AMUBs with good parameters. Section 3 describes the experimental results. Section 4 concludes the paper by summarizing our contribution in the domain of AMUB constructions, and by indicating the scope for further generalization of our framework to higher-dimensional Hilbert spaces.

1.2 Closeness Measures for AMUBs

We broadly use two measures to validate our results in terms of the *closeness* of AMUBs to MUBs: (i) Average Squared Distance (ASD) among the bases and (ii) maximum distance of $\langle a_i | b_j \rangle$ from $\frac{1}{\sqrt{d}}$, termed as Drift Measure.

$\overline{D^2}$ – Average Squared Distance (ASD) Between Bases. To measure the quality of AMUBs generated by our algorithm, we use the concept of distance between two bases, following Bengtsson *et al.* [2], who used the distance between two bases as a yardstick to measure the unbiasedness, and Raynal *et al.* [14], who used a similar measure to determine the results of their proposed algorithms.

The squared distance between two orthonormal bases $\mathbf{A} = \{|a_1\rangle, \ldots, |a_d\rangle\}$ and $\mathbf{B} = \{|b_1\rangle, \ldots, |b_d\rangle\}$ of a d-dimensional Hilbert space $\mathbb{C}^d$ is defined as

$$D^2_{\mathbf{AB}} = 1 - \frac{1}{d-1} \sum_{i,j=1}^{d} \left(|\langle a_i | b_j \rangle|^2 - \frac{1}{d} \right)^2, \tag{5}$$

and for a set of k orthonormal bases in $\mathbb{C}^d$, the Average Squared Distance (ASD) between the $k(k-1)/2$ pairs of bases is defined as the average over all pairs:

$$\overline{D^2} = \frac{2}{k(k-1)} \sum_{\mathbf{A}<\mathbf{B}} D^2_{\mathbf{AB}}. \tag{6}$$

The value of ASD is maximum ($\overline{D^2} = 1$) for a perfect set of MUBs, that is, when $|\langle a_i|b_j\rangle| = \frac{1}{\sqrt{d}}$ for all $i, j = 1, \ldots, d$ and for all pairs of bases **A, B** in the set. Thus, deviation of $\overline{D^2}$ from one in Eq. (6) provides a measure of *closeness* for AMUBs.

S – Maximum Distance of $|\langle a_i|b_j\rangle|$ from $\frac{1}{\sqrt{d}}$. Consider a set of m orthonormal bases in $\mathbb{C}^d$, where we choose pairs of bases $\mathbf{A} = \{|a_1\rangle, \ldots, |a_d\rangle\}$ and $\mathbf{B} = \{|b_1\rangle, \ldots, |b_d\rangle\}$ at a time. One may choose $\binom{m}{2}$ pairs of bases $\mathbf{A}, \mathbf{B}$, and for each pair $\mathbf{A}, \mathbf{B}$, one may compute d^2 inner products $|\langle a_i|b_j\rangle|$ for $i, j = 1, \ldots, d$.

We define the Drift Measure S as the maximum of absolute values of the distance $|\langle a_i|b_j\rangle| - 1/\sqrt{d}$ over all such choices of base pairs and inner products:

$$S = \max_{\mathbf{A},\mathbf{B}} \max_{i,j} \left| |\langle a_i|b_j\rangle| - 1/\sqrt{d} \right| \tag{7}$$

The value of Drift is minimum ($S = 0$) for a perfect set of MUBs, that is, when $|\langle a_i|b_j\rangle| = \frac{1}{\sqrt{d}}$ for all $i, j = 1, \ldots, d$ for all pairs of bases **A, B** in MUBs. Thus, deviation of S from 0 in Eq. (7) provides a measure of *closeness* for AMUBs. In order to understand how is the maximum departure in the form of some $\frac{\alpha}{\sqrt{d}}$ we also refer to α such that,

$$\alpha = \sqrt{d} \cdot S, \tag{8}$$

in the results section (Sect. 3).

In case of β-AMUBs (refer to Definition 3), it can be easily shown that (for a detailed derivation, refer to [13]),

$$S = \frac{\beta - 1}{\sqrt{d}} \tag{9}$$

Therefore, we have,

$$\beta = \alpha + 1 \tag{10}$$

1.3 Dimension Reduction Using SVD

The primary idea of generating AMUBs in our proposal revolves around the concept of dimension reduction. Several dimension reduction techniques are available in the literature [8,10], where the main purpose of these algorithms is to reduce certain high-dimensional data to a suitable lower dimension, preserving the characteristics and properties of the data as much as possible. In case of dimension reduction in this paper, we focus on Singular Value Decomposition (SVD).

The techniques related to SVD for real square matrices were proposed by Beltrami and Jordan and for complex matrices by Autonne (see [17] and references therein). The generic algorithm for SVD in case of rectangular matrices was proposed by Eckart and Young in the Autonne-Eckart-Young Theorem [7]. SVD decomposes a matrix of higher dimension into two unitary (orthogonal) matrices and a diagonal matrix containing the singular values. The singular values are ordered by their magnitudes (importance) in the data. The mathematical formulation of SVD is as follows.

Singular Value Decomposition. Over $\mathbb{C}$, a matrix $\mathbf{A}^{p\times q}$ of rank $\rho(\mathbf{A}) = r$ can be decomposed as $\mathbf{A}^{p\times q} = \mathbf{U}^{p\times p}\ \mathbf{\Sigma}^{p\times q}\ \mathbf{V}^{q\times q}$, where $\mathbf{U}^{p\times p}$ and $\mathbf{V}^{q\times q}$ are unitary matrices, and $\mathbf{\Sigma}$ comprises of the r nonnegative real singular values of the matrix $\mathbf{A}$ along its diagonal as $\mathrm{Diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$, and with all other values equal to 0. The columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{A}$ and the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{A}$. The *complete* Singular Value Decomposition (SVD) of the matrix $\mathbf{A}^{p\times q}$ can be expressed using the partitioning of matrices as follows.

$$\begin{array}{ccc} \mathbf{U}\quad (p\times p) & \mathbf{\Sigma}\quad (p\times q) & \mathbf{V}^T\quad (q\times q) \\ \left[\begin{array}{ccc|ccc} & & & & & \\ & & & & & \\ u_1 & \cdots & u_r & u_{r+1} & \cdots & u_p \\ & & & & & \\ & & & & & \end{array}\right] & \left[\begin{array}{ccc|c} \sigma_1 & & & \vdots \\ & \ddots & & \cdots 0 \cdots \\ & & \sigma_r & \vdots \\ \hline & \vdots & & \vdots \\ \cdots & 0 & \cdots & \cdots 0 \cdots \\ & \vdots & & \vdots \end{array}\right] & \left[\begin{array}{c} v_1^T \\ \vdots \\ v_r^T \\ \hline v_{r+1}^T \\ \vdots \\ v_q^T \end{array}\right] \end{array}$$

SVD in Our Proposal. We intend to generate AMUBs in $\mathbb{C}^d$ by dimension reduction from $\mathbb{C}^{d'}$, where d' is the closest prime power higher than d. In this paper, we provide examples where $d' = d + 1$ is a prime, and $\frac{d}{2}$ is a prime too. To accomplish this, we reduce a $z \times d'$ matrix to a $z \times d$ matrix, where $z \in \mathbb{Z}^+$ varies depending on the choice of our algorithms. We perform SVD on the $z \times d'$ matrix to obtain $\mathbf{U}^{z\times z}$, $\mathbf{\Sigma}^{z\times d'}$ and $\mathbf{V}^{d'\times d'}$. Finally, to obtain the reduced matrix of order $z \times d$ for generating AMUBs, we take the product of $\mathbf{U}^{z \times z}$, the first d singular values from $\mathbf{\Sigma}$ and a sub-matrix of order $d \times d$ from $\mathbf{V}$. If consecutive singular values are identical, there would be various choices to construct reduced matrices. In such cases, we try to optimize choices through exhaustive or heuristic searches.

2 Construction of AMUBs

The algorithm proposed in this section for generating d-dimensional Approximate Mutually Unbiased Bases (AMUBs) is a combination of dimension reduction and then search techniques to obtain appropriate bases with maximum possible ASD and a small value of S, as defined in Sect. 1.2. In the first part, we prepare the suitable bases for AMUB selection through dimension reduction. In the second part, we search for the best set of AMUBs from the bases developed in the first part. Finally, we apply a gradient-ascent kind of heuristic search as in [14] on the available AMUBs to obtain further improved results.

2.1 Creation of Bases

We use SVD to reduce suitable bases from a higher prime power (or prime) dimension d' to the target composite dimension $d < d'$. There are two aspects that we investigate here. That is, we propose two techniques to apply SVD – *Merging Technique* and *Non-Merging Technique*. First let us explain the merging technique through Fig. 1 and then Algorithm 1 follows.

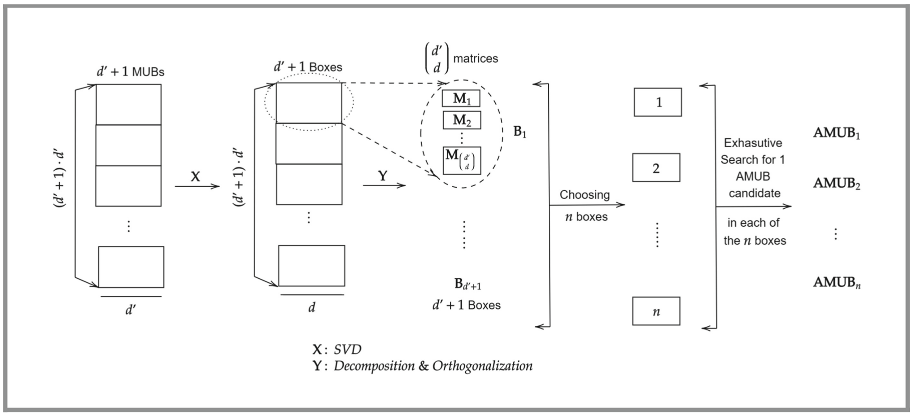


Fig. 1. Merging technique: schematic representation

As one may note, corresponding to $d' + 1$ many MUBs of dimension d', we naturally have each matrix with $d' \times d'$ entries. Thus, one can see that we have total $(d'+1)\cdot d'$ vectors here. Now the dimension will be reduced to d. Now we will consider all the $(d'+1)\cdot d'$ vectors for input to the SVD technique and then put them in different $d'+1$ buckets. Each bucket will contain d' many vectors. We will consider d vectors from those d' and orthogonalize them to have a basis. Thus, there will be several options of a basis from a bucket, and we will choose different buckets to obtain the bases corresponding to an AMUB. This will continue to obtain the best result. Naturally while considering $d' = 7$ and $d = 6$, such buckets can be searched exhaustively. However, the computational requirement becomes much higher and while running on a laptop, such exhaustive searches cannot be completed in reasonable time (say one hour) for $d' = 11, d = 10$. Thus, in these case exhaustive searches are not possible, and we go for only reasonable random samples of the complete possibilities. Nevertheless, one must note that given a powerful computational set-up, such exhaustive search is possible for even higher dimension and the computational effort can be estimated beforehand.

```
Algorithm 1: Merging Technique
  Data: Target dimension $d$
  Step 1 : Generate $d' + 1$ many MUBs of dimension $d'$, where $d'$ is the next higher prime to $d$.
  Step 2 : Merge the bases as generated in Step 1 in a matrix of order $((d' + 1) \cdot d') \times d'$.
  Step 3 : Implement the routine of SVD to reduce the dimension of the matrix formed in Step 2 to $((d' + 1) \cdot d') \times d$.
  Step 4 : The matrix as developed in Step 3 is split into $d' + 1$ many boxes, each containing a matrix of order $d' \times d$.
  Step 5 : In each box, decompose the matrix into $\binom{d'}{d}$ many matrices of order $d \times d$ and then orthogonalize to form the possible candidates for AMUB selection.
```

In the second effort, we consider each $d' \times d'$ matrix separately and apply SVD on each of them to construct each bucket and provide the results.

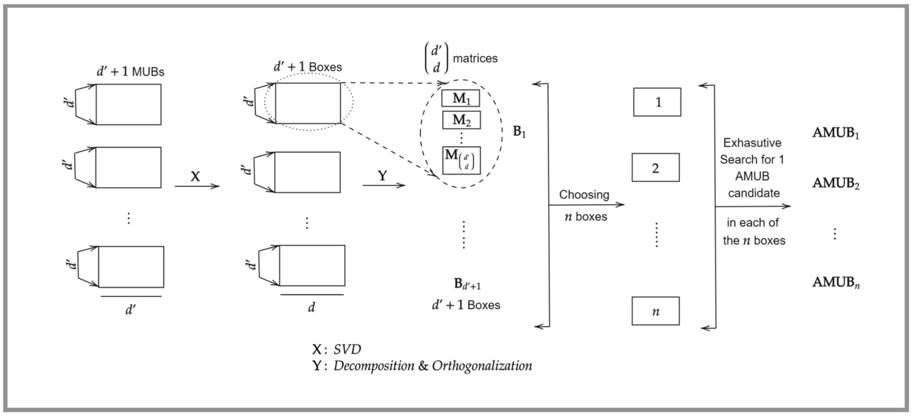


Fig. 2. Non-merging technique: schematic representation

As one can observe in Sect. 3, the results out of merging and non-merging techniques do not differ much. This needs further effort to study these, and several other theoretical issues and we plan to put those in journal version of this paper. In this technique (Non-Merged) of dimension reduction, the MUBs of dimension d' comes out to be unitary matrices. Hence, the SVD decomposition gives all the singular values as identity. Therefore, we always eliminate the last row and last column of the right singular matrix obtained from the decomposition and the corresponding singular value is made 0, post which an exhaustive search is performed as in the Merged Technique and the dimension reduction proceeds as usual. We explain the non-merging technique in Fig. 2 and then in Algorithm 2.

```
Algorithm 2: Non-Merging Technique
    Data: Target dimension $d$
    Step 1 : Generate $d' + 1$ many MUBs of dimension $d'$, where $d'$ is the next higher prime to $d$.
    Step 2 : Instead of merging the bases as generated in Step 1, implement the routine of SVD directly to these $d' + 1$ bases to obtain $d' + 1$ many matrices of order $d' \times d$, and then arrange them into $d' + 1$ many boxes.
    Step 3 : In each box, decompose the the matrix into $\binom{d'}{d}$ many matrices of order $d \times d$ and then orthogonalize to form the possible candidates for AMUB selection.
```

Thus, from both of our techniques as illustrated above in Algorithms 1 and 2, the idea of dimension reduction results in $d' + 1$ many boxes, each containing $\binom{d'}{d}$ bases of dimension d, and we thereby, proceed to our next step of choosing the appropriate bases.

2.2 Choice of Bases

The next component of our AMUB construction technique involves searching through the bases created and stored in $d' + 1$ many boxes (as in Sect. 2.1) to obtain the set of required AMUBs, which will result in best possible measures (as defined in Sect. 1.2) used for the experimentation. In general, the search procedure used to create n many AMUBs of dimension d, as illustrated through the following algorithmic steps relates to an exhaustive combinatorial search procedure.

```
Algorithm 3: The Exhaustive Search Procedure
    Step 1 : Choose $n$ boxes from $d' + 1$ many boxes in $\binom{d'+1}{n}$ ways.

    Step 2 : From each of the chosen $n$ boxes, select a candidate base of dimension $d$ in $\binom{k}{1}$ ways, where $k = \binom{d'}{d}$.

    Step 3 : Then, determine the respective measures for the selected candidate bases as in Step 2.

    Step 4 : Choose the set of bases with the best possible results.
```

This exhaustive search procedure is implemented for dimensions $d = 6, 10$ for four AMUBs and only for $d = 6$ in the case of five MUBs. Mathematically, for n many AMUBs and a dimension d, that we obtain from a higher prime power dimension d', the exhaustive search complexity is $\binom{d'+1}{n} \cdot \left[\binom{\binom{d'}{d}}{1}\right]^n$. However, other than the cases mentioned above, we could not manage the exhaustive search in reasonable time in a simple laptop. However, the formula clearly provides the estimation of computational requirements in practice and can be achieved in a high end computational facility for even larger dimensions.

2.3 Further Heuristics

The set of AMUBs that we obtain from the above dimension reduction technique and search is again subjected to another gradient-ascent kind of heuristic for improved results. The heuristic search algorithm as given in [14] is tweaked and this is applied on the initial solutions (n AMUBs) obtained from Sect. 2.2. This is broadly a steepest-ascent search procedure, where the gradients are computed in each iteration and the set of n bases are altered accordingly keeping the property of orthonormality intact, thereby resulting in the set of AMUBs with optimal approximations. The algorithm is given as follows.

```
Algorithm 4: The Heuristic Search Algorithm
  Step 1 : The gradient $\{G_k : k = 1, 2, \ldots, n\}$ of the $k^{th}$ base is computed
  with respect to the remaining $n-1$ bases, where $G_k$ is given by,
      $G_k = \frac{8}{n(n-1)(d-1)} Im\left[\sum_{l=1}^{n} \sum_{i,j=1}^{d} (|k_i\rangle \langle k_i|l_j\rangle \langle l_j|)^2\right]$
  Step 2 : For the step size ($\epsilon$) of the algorithm, compute $\sigma_k$ corresponding
  to the $k^{th}$ base such that, $\sigma_k = \epsilon G_k$ with a common $\epsilon > 0$.
  Step 3 : Implement a finite unitary change of the basis $k$, i.e.,
  $|k_j\rangle \rightarrow V_k|k_j\rangle$, where $V_k = \mathbf{1} + i\sigma_k$ upto first order of $\sigma_k$.
  Step 4 : Finally, orthogonalize the set of $n$ bases $|k_j\rangle$, $j = 1, 2, \ldots, d$;
  $k = 1, 2, \ldots, n$.
```

We run the algorithm until all the components of the gradient vanishes.

3 Results and Numerical Study

In this section, we present the results obtained from the implementation of the dimension reduction algorithms (refer to Algorithms 1, 2 and 3). Further, we provide the results, when the AMUBs obtained from the dimension reduction algorithms are subjected to the heuristic search technique (refer to Algorithm 4). The values of the dimensions (d and d'), measures, i.e., maximum ASD ($\overline{D^2_{\max}}$) over all the iterations (refer to Eq. 6) and the corresponding Drift Measure S (refer to Eq. 7) and α (refer to Eq. 8). We also highlight the complexity of our computations by reporting the number of iterations (denoted by $\#\mathbf{I}$). Note that, the initial solutions, i.e., the reduced bases obtained from the dimension reduction algorithms (Merging as well as Non-Merging Techniques) are subjected to the heuristic search procedure for a fixed number of iterations ($\#\mathbf{I} = 20000$) for $d = 6, 10$ and ($\#\mathbf{I} = 1000$) for $d = 46$ and step size ($\epsilon = 0.500$). We also present the results corresponding to the heuristic search taking into account i) *maximization of ASD* ($\overline{D^2}$) and ii) *minimization of the Drift Measure* (S). Thereafter, we provide a comparative study of our results along with the results obtained from a combinatorial point of view as in [12, Corollary 1] and [16]. Some relevant observations are also noted in this section corresponding to the results obtained under our proposed frameworks.

First, we perform dimension reduction using the Merged Technique (as in Algorithm 1). The results are presented (Table 1) for $n = 4$ and 5 sets of AMUBs in dimension 6, 10 and 46 respectively. Since the complexity for choosing bases for dimensions $10, 46$ is computationally expensive in certain cases, we randomly choose certain sets of boxes and search for the best results in them. That is the exhaustive search as in Algorithm 3 is not implemented in all the cases.

Table 1. Numerical results for merging technique: Algorithm 1

Number of bases	$d' \to d$	$\overline{D^2_{\max}}$	S	α	$\#\mathbf{I}$
$n = 4$	$7 \to 6$	0.912	0.380	0.931	$\binom{8}{4}7^4$
	$11 \to 10$	0.936	0.405	1.281	$\binom{12}{4}11^4$
	$47 \to 46$	0.979	0.317	2.150	47^4
$n = 5$	$7 \to 6$	0.904	0.389	0.953	$\binom{8}{5}7^5$
	$11 \to 10$	0.928	0.388	1.227	11^5
	$47 \to 46$	0.979	0.350	2.374	47^5

Note 1. The computational results as presented in Table 1 suggests that, our proposed algorithm related to the Merging Technique (Algorithm 1) successfully generates increased sets (*larger than the available bound*) of AMUBs with closer approximations to MUBs (with respect to $\overline{D^2}$).

The resultant reduced sets of bases/AMUBs obtained from the Merging Technique are further subjected to the Heuristic Search Algorithm, the results of which are provided below, both with respect to $\overline{D^2}$ and S in Tables 2 and 3 respectively.

Note 2. Few of the notable observations when the Heuristic Search is aimed at maximizing $\overline{D^2}$ under the Merging Technique, are as follows,

– For a fixed set of bases (n), as the dimension (d) increases, the average distance between the bases ($\overline{D^2}$) tends to increase.
– Now if the dimension (d) is kept fixed, the average distance between the bases seems to vary inversely with n.
– The value of α which normalizes the Drift Measure (S) is higher for larger dimensions (keeping n fixed) as well as for higher values of n (keeping d fixed).

Table 2. Numerical results for heuristic search (Algorithm 4) on the bases obtained from the Merged technique with respect to ASD ($\overline{D^2}$)

Number of bases	$d' \to d$	$\overline{D^2_{\max}}$	S	α	# **I**	Step size (ϵ)
$n = 4$	$7 \to 6$	0.971	0.383	0.938	10000	0.500
	$11 \to 10$	0.973	0.447	1.414	10000	0.500
	$47 \to 46$	0.982	0.366	2.482	1000	0.500
$n = 5$	$7 \to 6$	0.960	0.399	0.977	10000	0.500
	$11 \to 10$	0.930	0.369	1.167	10000	0.500
	$47 \to 46$	0.981	0.314	2.130	1000	0.500

Table 3. Numerical results for heuristic search (Algorithm 4) on the bases obtained from the Merged technique with respect to Drift Measure (S)

Number of bases	$d' \to d$	$\overline{D^2_{\max}}$	S	α	# **I**	Step size (ϵ)
$n = 4$	$7 \to 6$	0.931	0.325	0.797	10000	0.500
	$11 \to 10$	0.964	0.305	0.965	10000	0.500
	$47 \to 46$	0.977	0.320	2.174	1000	0.500
$n = 5$	$7 \to 6$	0.936	0.355	0.871	10000	0.500
	$11 \to 10$	0.917	0.332	1.050	10000	0.500
	$47 \to 46$	0.978	0.305	2.071	1000	0.500

Note 3. In case the Heuristic Search aimed at minimizing the Drift Measure (S) under the Merging Technique, we note the following.

- In view of the values of S (both in the cases when n and d are considered to be fixed) in Table 3, the pattern in which S varies is not definitive unlike $\overline{D^2}$.
- The behavior of α tends to be the same as mentioned in Note 2.

Consequently in Tables 4, 5 and 6, we report the results for the same sets of operations, as in dimension reduction and subjecting the bases to the heuristic search routine, when performed with the Non-Merged sets of bases both with respect to the optimization routine followed in view of maximizing the average distance ($\overline{D^2}$) and the Drift (S). As mentioned earlier, to avoid the high complexity in choosing bases for sets of 4 and 5 AMUBs of dimension 46 and sets of 5 AMUBs of dimension 10, we have randomly chosen certain sets of boxes instead of performing an exhaustive search and reported the best results obtained in them.

Note 4. Observations as reported above in Table 4 shows that the proposed Non-Merging Technique for generating AMUBs results into increased sets of bases (with respect to $\overline{D^2}$) providing close approximations to MUBs.

Note 5. In reference to Tables 5 and 6, we note the following.

Table 4. Numerical results for the non-merging technique: Algorithm 2

Number of bases	$d' \rightarrow d$	$\overline{D^2_{\max}}$	S	α	# **I**
$n = 4$	$7 \rightarrow 6$	0.909	0.423	1.036	$\binom{8}{4}7^4$
	$11 \rightarrow 10$	0.933	0.448	1.417	$\binom{12}{4}11^4$
	$47 \rightarrow 46$	0.974	0.558	3.785	47^4
$n = 5$	$7 \rightarrow 6$	0.901	0.423	1.036	$\binom{8}{5}7^5$
	$11 \rightarrow 10$	0.916	0.520	1.644	11^5
	$47 \rightarrow 46$	0.975	0.447	3.032	47^5

Table 5. Numerical results for Heuristic Search (Algorithm 4) on the bases obtained from non-merged technique with respect to $\overline{D^2}$

Number of bases	$d' \rightarrow d$	$\overline{D^2_{\max}}$	S	α	# **I**	Step size (ϵ)
$n = 4$	$7 \rightarrow 6$	0.972	0.404	0.989	10000	0.500
	$11 \rightarrow 10$	0.973	0.440	1.391	10000	0.500
	$47 \rightarrow 46$	0.982	0.600	4.069	1000	0.500
$n = 5$	$7 \rightarrow 6$	0.959	0.399	0.977	10000	0.500
	$11 \rightarrow 10$	0.967	0.391	1.236	10000	0.500
	$47 \rightarrow 46$	0.983	0.451	3.059	1000	0.500

– Heuristic Search aimed at *maximization of* $\overline{D^2}$ (Table 5) under the Non-Merging Technique leads to an increasing trend in the average distance between the bases both in the cases when i) d increases over a fixed value of n and ii) n increases over a fixed value of the dimension (d).
– When the Heuristic Search *minimizes* S (Table 6) under the Non-Merging Technique does not show any definitive pattern in the values of the Drift Measure, whereas the values α increases with d.

Note 6. Some important observations related to the Merged and Non-Merged Techniques of dimension reduction as well as the measures $\overline{D^2}$ and S are as follows.

– In reference to the results for the Merging and Non-Merging Techniques presented above in Tables 1 and 4, we observe that, the initial solutions in case of the Merging Technique provides more or less better results than the Non-Merging Technique with respect to the value of $\overline{D^2}$ (*more closer to* 1) as well as the value of S (*comparatively lower*).
– It is also to be noted that, we are considering the efficacy of our results, i.e., *closeness* of AMUBs to MUBs in terms of $\overline{D^2}$ as the heuristic search [14] deals with the optimization of $\overline{D^2}$, also we have extended the optimization (*minimization*) with respect to the Drift Measure S. And further we observe that in the Merging and Non-Merging Techniques, achieving *simultaneous control* over both $\overline{D^2}$ and S seems to be quite difficult at this stage, i.e., as

Table 6. Numerical results for Heuristic Search (Algorithm 4) on the bases obtained from non-merged technique with respect to drift measure (S)

Number of bases	$d' \rightarrow d$	$\overline{D^2_{\max}}$	S	α	# **I**	Step size (ϵ)
$n = 4$	$7 \rightarrow 6$	0.963	0.337	0.826	10000	0.500
	$11 \rightarrow 10$	0.926	0.303	0.958	10000	0.500
	$47 \rightarrow 46$	0.974	0.558	3.787	1000	0.500
$n = 5$	$7 \rightarrow 6$	0.908	0.364	0.892	10000	0.500
	$11 \rightarrow 10$	0.917	0.338	1.070	10000	0.500
	$47 \rightarrow 46$	0.983	0.446	3.026	1000	0.500

the value of $\overline{D^2}$ is being optimized (*closer to* 1) by the Heuristic Search, the value of S does not decrease. On the other hand, when the Heuristic Search aims at *minimizing* S, the value of $\overline{D^2}$ moves further away from 1, which is naturally expected from the respective definitions of the measures used.

In the following Table 7, we present the results for dimensions $d = 6, 10$ and 46, where AMUBs have been constructed using a combinatorial technique. In reference to [12, Corollary 1], combinatorial construction for AMUBs is available for every even dimension with $k = 2$. For such construction, it can be shown that,

$$\overline{D^2} = \frac{d \cdot (k^2 - 1)}{k^2 \cdot (d - 1)}, \text{for } k = 2 \tag{11}$$

$$\Rightarrow \overline{D^2} = \frac{3}{4} \cdot \frac{d}{d - 1} \tag{12}$$

It is to be noted that, the value of $\overline{D^2}$ is smaller in case of the combinatorial construction of AMUBs in comparison to the the construction algorithms proposed in this paper for generating AMUBs. Observe that, $\overline{D^2} \rightarrow 1 - \frac{1}{k^2}$ asymptotically for large values of d. And in general for the dimensions under consideration, i.e., $d = 6, 10$ and 46, we have $\overline{D^2} \rightarrow \frac{3}{4} = 0.75 \ll$ all the values of $\overline{D^2}$ achieved using the above techniques. We should also mention that the theoretical result of [16] is not applicable for the small dimensions $d = 6, 10$. The bound for $d = 46$ is also worse than what we obtain by our technique.

Table 7. Numerical results for AMUBs constructed using combinatorial techniques [12, Corollary 1]

Number of bases	d	$\overline{D^2}$	S	α
$n = 4, 5$	6	0.900	0.408	0.999
	10	0.833	0.316	0.999
	46	0.767	0.352	2.387

All the results above are computed in a laptop supported by `Apple M1 chip - 8 core CPU` and `8 GB of RAM`, using the open source `Python` programming language. We have implemented a multiprocessing technique using the `Python` function `multiprocessing.ProcessPool`, with 16 processes to speed up the process of choosing AMUBs in dimension reduction technique. Significant improvement over execution timings as well as number of iterations can be obtained if the program is executed on a GPU integrated environment. The SVD routine was performed using the `numpy.linalg.svd` function from the `Python` library `numpy`. In Algorithms 1, 2 and 4, we utilize the QR Decomposition routine for orthogonalizing the set of bases using the `numpy.linalg.qr` function from the `Python` library `numpy`. The relevant codes of the numerical experimentation and computations are present in the `GitHub Repository` [19].

4 Conclusion

In this paper we have presented a broad framework for the construction of AMUBs. The initial approach is based on the widely used dimension reduction technique, namely the Singular Value Decomposition (SVD). This technique has been successfully exploited in different domains of the Machine Learning literature. The resulting approximate MUBs from this strategy are considered as initial solutions. Consequently, those are further subjected to a steepest ascent kind of heuristic search technique, as in [14], to obtain improved results. The novelty of our approach lies in the fact that the broad framework does not require any prior mathematical formulation and parametrization of the AMUBs to be generated. Observe that, the heuristic search has been implemented aiming to optimize the average distance between the bases ($\overline{D^2}$) [14] and further extended to an optimization of the drift measure (S). Hence, it is to be kept in mind that, this generic approach of computation and experimentation to generate AMUBs presented in this paper can be efficiently used with the two available optimizations as and when required. As we note, these are only initial experimental results that can be improved with different kinds of refinements and even with this initial approach, we could obtain quite encouraging results.

Further, our prospective work will address another general experimentation of generating AMUBs for any dimension d. One may first consider the available MUBs for that dimension d, say k and then produce sets of $k+1$ AMUBs, where the $(k+1)$-th basis may be explored from the dimension reduction technique, i.e., which involves the reduction of dimension d' (next higher prime to d) to d. For example, consider dimension $d = 6 = 2 \cdot 3$, for which $k = 3$ MUBs are known to exist. To generate sets of $k+1 = 4$ AMUBs, we will take the 4-th AMUB from the bases created by the dimension reduction routine (reducing dimension from $d' = 7$ to $d = 6$). We may go for some steepest ascent kind of heuristic further. Similarly, for dimension $d = 12 = 2^2 \cdot 3$, one can obtain sets of $k+1 = 5$ AMUBs from the $k = 4$ MUBs that are known to exist. The 5-th AMUB would be generated through dimension reduction (from $d' = 13$ to $d = 12$) and further search. Likewise, this approach can be extended to higher dimension, providing another generic framework in obtaining Approximate Mutually Unbiased Bases.

References

1. Aguilar, E.A., Borkala, J.J., Mironowicz, P., Pawlowski, M.: Connections between mutually unbiased bases and quantum random access codes. Phys. Rev. Lett. **121**, 050501 (2018)
2. Bengtsson, I., Bruzda, W., Ericsson, A., Larsson, J., Tadej, W., Życzkowski, K.: Mutually unbiased bases and Hadamard matrices of order six. J. Math. Phys. **48**, 052106 (2007). https://arxiv.org/abs/quant-ph/0610161
3. Bennett, C.H., Brassard, G.: Quantum cryptography: public key distribution and coin tossing. In: Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, Bangalore, pp. 175–179 (1984)
4. Colomer, M.P., Mortimer, L., Frérot, I., Farkas, M., Acín, A.: Three numerical approaches to find mutually unbiased bases using bell inequalities. arXiv preprint arXiv:2203.09429 (2022)
5. Dubner, H.: Large Sophie Germain primes. Math Comput. **65**(213), 393–397. https://doi.org/10.1090/S0025-5718-96-00670-9
6. Durt, T., Englert, B., Bengtsson, I., Życzkowski, K.: On mutually unbiased bases. Int. J. Quantum Inform. **8**, 535–640 (2010). https://doi.org/10.1142/S0219749910006502
7. Eckart, C., Young, G.: A principal axis transformation for non-Hermitian matrices. Bull. Amer. Math. Soc. **45**(2), 118–121 (1939)
8. Fodor, I.K.: A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA, USA (2002)
9. Gibbons, K.S., Hoffman, M.J., Wootters, W.K.: Discrete phase space based on finite fields. Phys. Rev. A **70**, 1–23 (2004). https://link.aps.org/doi/10.1103/PhysRevA.70.062101
10. Kalman, D.: A singularly valuable decomposition: the SVD of a matrix. Coll. Math. J. **27**(1), 2–23 (1996)
11. Kumar, A., Maitra, S., Mukherjee, C.S.: On approximate real mutually unbiased bases in square dimension. Cryptogr. Commun. **13**(2), 321–329 (2021). https://doi.org/10.1007/s12095-020-00468-6
12. Kumar, A., Maitra, S.: Resolvable block designs in construction of approximate real MUBs that are sparse. Cryptogr. Commun. (2021). https://doi.org/10.1007/s12095-021-00537-4
13. Kumar A., Maitra, S., Roy, S.: Almost perfect real MUBs that are sparse, November 2021. Preprint
14. Raynal, P., Lu, X., Englert, B.: Mutually unbiased bases in six dimensions: the four most distant bases. Phys. Rev. **83**, 062303 (2011). https://link.aps.org/doi/10.1103/PhysRevA.83.062303
15. Sadek, R.A.: SVD based image processing applications: state of the art, contributions and research challenges. IJACSA - Int. J. Adv. Comput. Sci. Appl. (2012). https://arxiv.org/ftp/arxiv/papers/1211/1211.7102.pdf
16. Shparlinski, I.E., Winterhof, A.: Constructions of approximately mutually unbiased bases. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 793–799. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_72
17. Stewart, G.W.: On the early history of the singular value decomposition. SIAM Rev. **35**, 551–566 (1993). https://www.jstor.org/stable/2132388
18. Wootters, W.K., Fields, B.D.: Optimal state-determination by mutually unbiased measurements. Ann. Phys. **191**(2), 363–381 (1989). https://doi.org/10.1016/0003-4916(89)90322-9
19. `GitHub` Repository for Codes - https://bit.ly/3KPS9jk

Counter Mode for Long Messages and a Long Nonce

Shay Gueron[1,2](✉)

[1] University of Haifa, Haifa, Israel
shay.gueron@gmail.com
[2] Amazon, Seattle, USA

Abstract. This paper proposes "Compound-CTR" mode—a simple variation of Counter mode (CTR) with an n bits block cipher. Its goal is to increase the allowed length of a single message and the total number of messages that can be encrypted under a single key.

Compound-CTR encrypts a message and a (randomly chosen) nonce with length greater or equal n bits. It uses a master key to derive a nonce-based encryption key and subsequently uses it for encrypting the message in CTR mode.

We show how Compound-CTR mode achieves its goal and explain why it can be used as a valid variation of CTR mode that could be of interest in some practical scenarios. Compared to CTR mode, the overhead of Compound-CTR is only the per-message key derivation and one extra key expansion (for the block cipher). We show here key derivation options that require only a few extra block cipher calls.

Keywords: Block ciphers · Modes of operation · Counter mode

1 Introduction

Counter mode (CTR) is a block cipher mode of operation with multiple performance and convenience benefits. It is frequently used as a standalone confidentiality mode and as a component of a confidentiality-and-authentication construction. For example, it is the underlying encryption in AEAD modes such as AES-GCM [2] and CCM [6].

The standard CTR mode with an n bits block cipher E encrypts an input message M and a v bits initialization vector IV by XOR-ing M with a pseudorandom mask of the same length as M. The mask is generated by repeated invocations of E (with a given key) over n bits blocks (called "counter blocks") that consist of the IV concatenated with a running counter with $(n - v)$ bits. The IV values that are used across all of the encrypted messages must be distinct. When the IV values are sampled uniformly at random from the set of v bits strings, CTR construction introduces a tradeoff between: a) the maximal allowed number of blocks in a single message; b) the value of v that is necessary to keep the IV collision probability below a desired threshold. This affects the required key rotation rate.

We give the relevant example of the standard CTR mode with the 128 bits block cipher AES and a $v = 96$ bits IV.

– The counter block (that is encrypted) holds IV and can therefore accommodate a counter of only $128 - 96 = 32$ bits. Therefore, the maximal allowed length of a message is 2^{32} blocks (of 16 bytes, i.e., 2^{36} bytes). It is conceivable that modern use cases, especially for encryption at the cloud scale, could benefit from a higher maximum.
– If the IV is chosen uniformly at random from $\{0,1\}^{96}$, and the IV collision probability across all messages is required to remain below the safety margin of 2^{-32}, the number of messages under the same key is limited to 2^{32} (regardless of their lengths).

These limitations impose a (possibly too frequent) key rotation rate and a constraint on the possible message lengths.

In addition, note that the standard "birthday bound" limits the total number of blocks that can be encrypted under a given key to $2^{n/2-\zeta}$ where ζ is determined by the desired security margin.

This paper shows a simple method to alleviate these restrictions.

2 Notation and Definitions

We denote the set of strings of bits of length v by $\{0,1\}^v$ and the length of a string S by $|S|$. If $|S| \geq w > 0$, we denote the truncation of S to w bits (i.e., the string that consist of the w least significant bits of S) by $\mathsf{Truncate}(w, S)$.

If $a > 0$ is an integer and $i \in [0, 2^a - 1]$, we denote the a-bit string that encodes i as a binary number by $\mathsf{IntToStr}_a(i)$. For example, $\mathsf{IntToStr}_6(10) = 001010$ and $\mathsf{IntToStr}_8(100) = 01100100$. Concatenation of strings is denoted by $\|$. For example $\mathsf{IntToStr}_6(10) \parallel \mathsf{IntToStr}_8(100) = 00101001100100$. A string of s repeating 0 (1) bits is denoted by 0^s (1^s). For example, $0^6 = \mathsf{IntToStr}_6(0) = 000000$.

Hereafter, E denotes a block cipher for blocks of size n, and $E(K, P)$ denotes the encryption of the n bits block P under the key K. Here $|K|$ can be either n or $2n$, and this is implicit in the context of E. For convenience, we may assume that $n \geq 64$ and is divisible by 8. In a practical context E is AES (where $n = 128$) with key of either 128 or 256 bits.

Parsing a Message. Let M be a nonempty string of $p = |M|$ bits. $\mathsf{Parse}(M)$ is the encoding of M as the concatenation of $s = \text{ceil}(p/n)$ disjoint n bits blocks, and is denoted by

$$\mathsf{Parse}(M) = m1, m2, \ldots, ms*$$

where mj are complete blocks with $|mj| = n$ for $j = 1, 2, s - 1$, and the last block $ms*$ is potentially an incomplete block with $|ms*| = p - n(s-1) \leq n$.

For example (with $n = 128$), if $M = 0^{256}$ then $s = 2$, $\mathsf{Parse}(M) = m1, m2*$ where $m1 = 0^{128}$, $m2* = 0^{128}$ and both block are complete. If $M = 0^{128} \parallel 1^{128} \parallel 0^{64}$ then $s = \text{ceil}(320/128) = 3$, $\mathsf{Parse}(M) = m1, m2, m3s*$ where $m1 = 0^{128}$, $m2 = 1^{128}$ are complete blocks and $m3* = 0^{64}$ is incomplete.

3 CTR Mode and Its Limitations

The standard CTR mode with an n bits block cipher E encrypts an input message M and a v bits initialization vector IV by XOR-ing M with a pseudorandom mask of length $|M|$. Decryption is the same operation, and we therefore ignore it hereafter. The mask is generated by repeated calls to $E(K, \cdot)$ over *distinct* counter blocks of n bits that have form $IV \parallel \mathsf{IntToStr}_u(j)$ for some counter j. Here, j counts the blocks in the message to be encrypted.

To achieve confidentiality with this mode, it is crucial that the IV values that are used with the same key, across all the messages, are distinct. This guarantees the uniqueness of the counter blocks. CTR encryption is shown in Fig. 1.

```
CTR-ENC ($K$, $v$, $IV$, $M$)
Input: $K$ (key), $IV$ (with $|IV| = v$), message $M$.
 1. Set $|M| = p$, $s = \text{ceil}(p/n)$, $w = p - n(s-1)$, $u = n - v$.
 2. $\mathsf{Parse}(M) = m1, m2, \ldots, ms*$ where $|ms*| = w$.
 3. For j from 1 to s-1
    (a) $Tj = E(K, IV \parallel \mathsf{IntToStr}_u(j))$
    (b) $cj = mj \oplus Tj$
 4. $Ts = E(K, IV \parallel \mathsf{IntToStr}_u(s))$
 5. $cs* = ms* \oplus \mathsf{Truncate}(w, Ts)$
 6. $C = c1, c2, \ldots, cs*$
Output C    ($|C| = |M|$)
```

Fig. 1. CTR mode.

We point out that CTR mode provides only for confidentiality, and is considered "malleable": flipping a bit in the ciphertext would flip (upon decryption) the corresponding bit of the recovered plaintext. Therefore, for most applications, CTR encryption should be complemented with some means of authentication.

Our discussion focuses on the *stateless* encryption form of CTR mode where IV is chosen uniformly at random from $\{0,1\}^v$ for every encryption. For the analysis, we model the block cipher E as a random permutation.

Suppose that a key K is used for encrypting q messages M_i of respective lengths p_i, and denote $s_i = \text{ceil}(p_i/n)$, $i = 1, \ldots, q$. Then, the distinguishing advantage of a chosen plaintext adversary that observes q resulting ciphertexts is upper bounded by

$$Adv(q, p_1, \ldots, p_q) \leq \frac{q^2}{2^{v+1}} + \frac{\left(\sum_{i=1}^{q} s_i\right)^2}{2^{n+1}} \tag{1}$$

Denote the first term in (1) by $P1$ and the second term by $D1$. To explain (1), note that:

– $P1$ is an upper bound for the probability of the *bad* event where (at least) two IV values repeat across q random selections.
– The number of counter blocks encrypted for message M_i is s_i and hence the number of (distinct) blocks encrypted for the q messages is $(\sum_{i=1}^{q} s_i)$. Thus, the term $D1$ represents the standard PRF-PRP distinguishing advantage.

These bounds are essentially tight. We assume that the encrypted messages are not empty and therefore $s_i \geq 1$ for $i = 1, \ldots, q$. This implies that $P1 \geq D1$. If all the messages are short (e.g., $|M_i| = 128$) then $D1$ and $P1$ remain at the same order of magnitude. For long messages (and relatively small q) $P1$ dominates $D1$.

We can see that CTR mode imposes limits on the maximal number of encryptions allowed under the same key (q_{max}) and on the maximal length of a single messages (L_{max}). For example, consider the standard choice of $v = 96$ and the standard requirement that $P1 \leq 2^{-32}$. Since $u = n - v = 32$, we have $L_{max} \leq 2^{32} - 1$, and the requirement on $P1$ forces $q_{max} \leq 2^{32.5}$. Independently, $D1$ (or $D1 + P1$) also needs to be kept sufficiently small, and this translates to a limit on the total number of blocks ($B_{max} = \sum_{i=1}^{q} s_i$) that can be encrypted under the same key.

4 Compound-CTR

Our goal is to find a way that allows for more flexibility for q_{max}, L_{max}, and B_{max}, compared to CTR mode. To this end, we define the Compound-CTR mode that encrypts a message M and a nonce N with $|N| = \nu$, but supports the nonce length of $n \leq \nu < 2n - 16$. With no loss of generality, assume that ν is even. We choose $\nu < 2n - 16$ here for convenience, so that all the definitions apply to full bytes. We consider the stateless scenario where a new nonce is sampled, uniformly at random, from $\{0, 1\}^{\nu}$ for every message.

Splitting a Nonce. We use the notation $\mathsf{SplitNonce}(N)$ to describe the splitting of a string N into two equal-length (disjoint) halves, namely

$$\mathsf{SplitNonce}(N) = N1, N2 \text{ where } N = N1 \parallel N2 \text{ and } |N1| = |N2| = |N|/2.$$

For example, if $N = 0^{64} \parallel 1^{64} \parallel 0^{32}$ ($|N| = 160$), then $\mathsf{SplitNonce}(N) = N1, N2$ where $N1 = 0^{64} \parallel 1^{16}$, $N2 = 1^{48} \parallel 0^{32}$ (and $|N1| = |N2| = 160$).

Figures 2, 3 and 4 below show three methods to derive a pseudorandom encryption key (KE) of length n bits ($\mathsf{Derive0}$ and $\mathsf{Derive1}$) and of $2n$ bits ($\mathsf{Derive2}$) from a (long) nonce N with $n \leq |N| < 2n - 16$. They commence with $\mathsf{SplitNonce}(N)$ and operate on the resulting halves $N1$, $N2$. Figure 5 shows the Compound-CTR mode that takes a message (M) and a (long) nonce, derives a fresh encryption key (KE) using one of the derivation schemes, and encrypts M with CTR (encryption) mode using KE and a zero IV. Explanations and design rationale follow the figures in Sect. 4.1.

```
Derive0($K, \nu, N$)
Input: $K$ $N$
 1. SplitNonce($N$) $= N1, N2$
 2. $X0 = 0 \parallel 1 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 3. $Y0 = 1 \parallel 0 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 4. $KE = E(K, X0) \oplus E(K, Y0)$
 5. Output $KE$
```

Fig. 2. Derive0. Deriving an n bits encryption key.

```
Derive1($K, \nu, N$)
Input: $K$ $N$
 1. SplitNonce($N$) $= N1, N2$
 2. $X0 = 0 \parallel 1 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 3. $Y0 = 1 \parallel 0 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 4. $U1 = 0 \parallel 1 \parallel 0 \parallel 1 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 5. $U2 = 1 \parallel 0 \parallel 0 \parallel 1 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 6. $KE = E(K, X0) \oplus E(K, U1) \oplus E(K, Y0) \oplus E(K, U2)$
 7. Output $KE$
```

Fig. 3. Derive1. Deriving an n bits encryption key.

```
Derive2($K, \nu, N$)
Input: $K$ $N$
 1. SplitNonce($N$) $= N1, N2$
 2. $X0 = 0 \parallel 1 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 3. $Y0 = 1 \parallel 0 \parallel 0 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 4. $U1 = 0 \parallel 1 \parallel 0 \parallel 1 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 5. $V1 = 0 \parallel 1 \parallel 1 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N1$
 6. $U2 = 1 \parallel 0 \parallel 0 \parallel 1 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 7. $V2 = 0 \parallel 1 \parallel 1 \parallel 0 \parallel 0^4 \parallel 0^{n-8-\nu/2} \parallel N2$
 8. $K' = E(K, X0) \oplus E(K, U1) \oplus E(K, Y0) \oplus E(K, U2)$
 9. $K'' = E(K, X0) \oplus E(K, V1) \oplus E(K, Y0) \oplus E(K, V2)$
10. Output $KE = K' \parallel K''$
```

Fig. 4. Derive2. Deriving a $2n$ bits encryption key.

```
Compound-CTR-ENC-$j$ ($K$, $\nu$, $N$, $M$)
Input: $K$ (key), $N$ (with $|N| = \nu$), message $M$.
 1. $KE = \mathsf{Derive}\text{-}j(K, N)$
 2. $C$ = CTR-ENC ($KE$, $n/2$, $0^{n/2}$, $M$)
Output $C$    ($|C| = |M|$)
```

Fig. 5. Compound-CTR (encryption) mode. The definition refers to three options that correspond to choosing one of the key derivation methods Derive0, Derive1, Derive2 in step #1. The notation Derive-j ($j \in \{0, 1, 2\}$) indicates the derivation choice and Compound-CTR-ENC-j indicates the corresponding variant.

4.1 Explanation, Design Rationale and Properties

To explain the properties that Compound-CTR-ENC offers, suppose that it is used with the key K for encrypting q messages M_i of respective lengths p_i, and denote s_i = ceil(p_i/n), $i = 1, \ldots, q$. Here, q nonces are sampled, uniformly at random from $\{0, 1\}^\nu$. These nonces are input to Derive-j ($j \in \{0, 1, 2\}$) in order to derive q encryption keys from $\{0, 1\}^n$ (or from $\{0, 1\}^{2n}$). Let KE_i be the i-th derived key that is used for encrypting the message M_i $i = 1, \ldots, q$.

Collisions. We start with this assumption on the quality of Derive-j as a pseudorandom function: the result of the q key derivations is indistinguishable from a uniform random sampling from the respective set ($\{0, 1\}^n$ or $\{0, 1\}^{2n}$).

Now, consider two possible *bad* events: a) two nonces collide. The probability for this event is at most $q^2/2^{\nu+1}$; b) two derived keys collide although the nonces do not collide. The probability for this event is at most $q^2/2^{n+1}$ (or $q^2/2^{2n+1}$).

Thus, with probability at most $q^2/2^{\nu+1} + q^2/2^{n+1}$ (or $q^2/2^{\nu+1} + q^2/2^{2n+1}$), we can assume that all the derived encryption keys are distinct.

The Use of CTR Mode with the Derived Keys. Since KE_i is used for only one message we can invoke CTR encryption with an arbitrary IV (which is, by definition, unique for this key). In particular, we can choose $IV = 0^{n/2}$. Now, the counter blocks are of the form $0^{n/2} \parallel \mathsf{IntToStr}_{n/2}(\cdot)$ and can accommodate a counter of $n/2$ bits for the corresponding blocks of M_i. This alleviates the restriction on the maximum message length that is imposed by CTR mode.

The distinguishing advantage of a chosen plaintext adversary, observing ciphertexts from M_i, encrypted under KE_i, depends on an assumption on the block cipher E in a multi-key setting. Modeling E as a random permutation in this setting, the advantage is upper bounded by

$$\sum_{i=1}^{q} \frac{{s_i}^2}{2^{n+1}} \tag{2}$$

The Key Derivation Functionality Derive. The purpose of Derive-j (i.e., one of Derive0, Derive1, Derive2) is to generate a fresh pseudorandom encryption key for every message, using a nonce of length $\nu \geq n$. First, note that if N and N' are distinct nonces, $\mathsf{SplitNonce}(N) = N1, N2$ and $\mathsf{SplitNonce}(N') = N1', N2'$, then we have at least $N1 \neq N1'$ or $N2 \neq N2'$. Note also the domain separations across the constructions of the blocks $X0$, $Y0$, $U1$, $U2$, $V1$, $V2$ (Figs. 2, 3 and 4). We point out that the use of 4 zero bits (0^4) in these blocks is only a convenient choice that completes the first four bits to a full byte. The Derive versions use simple constructions to generate a pseudorandom function from a permutation. Their PRF indistinguishability quality can be explained as follows.

– Derive0 evaluates the permutation E over 2 blocks per message, altogether $2q$ times (at least q are distinct blocks). Due to the PRP-PRF property, the PRF distinguishing advantage is at most $4q^2/2^{n+1}$.
– Derive1 uses 4 evaluations of E per message. It includes the XOR of at least two evaluations over distinct blocks. Here, the PRF distinguishing advantage is at most $16q/2^n$ [1,5].
– Derive2 uses 6 evaluations of E per message, in order to generate a key of $2n$ bits. It applies the CENC construction [3,4] twice. Here, the PRF distinguishing advantage is at most $18q/2^n$.

The Full Construction: Adversary Advantage Against Compound-CTR-ENC-*j*. Suppose that an adversary views Compound-CTR-ENC-j ciphertexts of q chosen messages M_i of respective lengths p_i (denote $s_i = \text{ceil}(p_i/n)$), and makes Q_E offline chosen-key queries (to guess one of KE_i) $i = 1, \ldots, q$. Suppose that E is modeled as a random permutation in a multi-key settings. Here, $j \in \{0, 1, 2\}$ controls the length of the derived keys: n bits for $j = 0, 1$ and $2n$ bits for $j = 2$.

Then, the adversary's advantage in distinguishing the Compound-CTR-ENC ciphertexts from q uniform random strings of lengths p_i is upper bounded by the sum of:

a) the PRF distinguishing advantage of Derive, when it is called q times;
b) the probability that there is no collision in the derived keys;
c) the distinguishing advantage of q CTR ciphertexts that encrypt every message M_i under a (uniqe) uniform random key;
d) The key guessing probability, which is $Q_E/2^n$ for Derive0, Derive1, and $Q_E/2^{2n}$ for Derive2.

5 Discussion

The Compound-CTR-ENC mode is designed to increase the allowed length of a single message and the total number of messages that can be encrypted under a single key, compared to the standard CTR mode. The construction is very simple, and uses only a block cipher as the cryptographic building block. The per-message overhead involved with Compound-CTR-ENC, compared to CTR mode, is at most 6 extra invocations of E plus one key expansion (as it is in the case where the block cipher is AES). We recommend the Compound-CTR-ENC-2 variant that uses Derive2 for deriving a key of $2n$ bits, and a nonce length $\nu = 5n/4$. This provides very comfortable security bounds. For consistency, the block cipher E should also use a $2n$ bits.

We conclude with the following example that illustrates the use of Compound-CTR-ENC-2. Let the main key K have $|K| = 256$, and let E be AES with a 256 bits key. With Derive2, every derived key KE has $|KE| = 256$. We choose the nonce length $\nu = 160$ bits. Suppose that an adversary can compute Q_E evaluations of E with a chosen key (at a chosen value). Suppose that an adversary uses the scheme for encrypting q (chosen) equal-length messages of length 2^{36} blocks (2^{40} byes). Note that these messages are all longer than the limit imposed by CTR mode. The adversary distinguishing advantage is upper bounded by:

$$\frac{1}{2}\frac{q^2}{2^{160}} + \frac{1}{2}\frac{q^2}{2^{256}} + \frac{18q}{2^{128}} + \frac{q}{2^{57}} + \frac{Q_E}{2^{256}}$$

Here, K was used for processing a total of $2^{40}q$ bytes. This amount crosses the birthday bound already at $q = 2^{28}$. Yet, for all conceivable values of Q_E, the advantage is dominated by $q/2^{57} = 2^{-29}$.

Acknowledgments. This research was supported by: NSF-BSF Grant 2018640; The Israel Science Foundation (grant No. 3380/19); The Center for Cyber Law and Policy at the University of Haifa, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

References

1. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS, pp. 394–403. IEEE Computer Society (1997)
2. Dworkin, M.: SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST, November 2007. https://csrc.nist.gov/publications/detail/sp/800-38d/final
3. Iwata, T.: New blockcipher modes of operation with beyond the birthday bound security. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 310–327. Springer, Heidelberg (2006). https://doi.org/10.1007/11799313_20
4. Iwata, T., Mennink, B., Vizár, D.: CENC is optimally secure. Cryptology ePrint Archive, Report 2016/1087 (2016). https://ia.cr/2016/1087
5. Lucks, S.: The sum of PRPs Is a secure PRF. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 470–484. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_34
6. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). RFC 3610, September 2003. https://doi.org/10.17487/RFC3610, https://www.rfc-editor.org/info/rfc3610

Transfer Learning for Time Series Classification Using Synthetic Data Generation

Yarden Rotem, Nathaniel Shimoni(✉), Lior Rokach, and Bracha Shapira

Ben-Gurion University of the Negev, Be'er Sheva, Israel
{rotemyar,nathanie,liorrk,bshapira}@post.bgu.ac.il

Abstract. In this paper, we propose an innovative Transfer learning for Time series classification method. Instead of using an existing dataset from the UCR archive as the source dataset, we generated a 15,000,000 synthetic univariate time series dataset that was created using our unique synthetic time series generator algorithm which can generate data with diverse patterns and angles and different sequence lengths. Furthermore, instead of using classification tasks provided by the UCR archive as the source task as previous studies did, we used our own 55 regression tasks as the source tasks, which produced better results than selecting classification tasks from the UCR archive.

Keywords: Transfer learning · Time series classification · Synthetic data

1 Introduction

Transfer learning (TL) is a machine learning (ML) technique that tries to utilize knowledge learned from a source domain in a relevant target domain. The relevant knowledge is applied to the target domain in order to improve the performance of the prediction function of the target domain [21]. The need for sufficient training data exists in most ML tasks, but obtaining labeled data can be expensive, time-consuming, or in some cases - infeasible. TL is a promising technique which can address this problem by transferring the knowledge across domains, preventing the need for labeled data in sparse domains [22].

TL has also shown to be effective at addressing some of the challenges with training a deep learning model which typically is time-consuming and requires high computational resources. Moreover, when lacking training data, ML models encounter the overfitting problem [24].

Transfer learning has also been widely used in computer vision, with state-of-the-art neural network (NN) models such as AlexNet [3] and ViT-G/14 [9], which is the current leader in terms of top-1 accuracy on the ImageNet [4] dataset. Evaluation of models pretrained on ImageNet show that accuracy is improved when using TL on new target datasets as opposed to training with the same architecture from scratch [8]. TL has also been used effectively for natural

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 232–246, 2022.
https://doi.org/10.1007/978-3-031-07689-3_18

language processing (NLP) tasks with pretrained models using word2vec and BERT models, and BERT's later versions were considered state of the art [5]. Many studies used pretrained NLP models (such as BERT) to serve as a good starting point for new target datasets. However, for time series (TS) tasks, limited effort has been invested in developing a state-of-the-art, generic, and robust pretrained model that provides a good starting point for a new task.

A **time series** is a series of data samples in a time-based domain, which are typically sampled at a uniform time interval [20]. There are two main types of TS: univariate time series (UTS) and multivariate time series (MTS) [19]. An MTS is an M-dimensional TS where each data sample consists of M real values, e.g., an MTS can be data acquired by measuring multiple climate sensors, such as temperature, humidity, and wind speed, once an hour; this is an $M = 3$ MTS. A UTS is simply an MTS where $M = 1$; a UTS can be data acquired by sampling the heartbeat of a patient every 10 s [10]. In this paper, we focus only on UTS data. TS data is relevant for many domains, including the analysis of financial transactions [18], monitoring network traffic [17], the analysis of time-based medical events [16]. In fact, TS data mining was mentioned as one of the top 10 data mining problems by Yang and Wu [15].

Time series data analysis is a highly focused research domain that has a number of different applications. The three main applications are: **time series classification (TSC)** - the task of training a classifier to map a given input to a probability over the possible class values (labels) [13], time series forecasting - the task of predicting future values of a given sequence using previous data [12], and time series clustering - the task of dividing a set of TS data into groups, where similar TS samples are put in the same cluster [11]. In this work, we focus only on time series classification.

TL for TSC has not been extensively studied, and a generic, robust, and scalable pretrained model that can serve as a good starting point for new datasets is needed, especially when there is insufficient labeled data. Due to the time-consuming process of collecting and labeling data, the availability of such a pretrained model is essential and would reduce the training time and cost, and in some cases, these models could lead to better overall results.

In this study, we propose an innovative, generic, scalable, and architecture-agnostic TL for TSC method based on (1) our new algorithm for generating synthetic data and (2) 55 corresponding regression tasks. Our method can be applied to any deep learning CNN-based architecture. As opposed to previously proposed TL for TSC methods, our model only needs to be pretrained once, and there is no need to search for the optimal source dataset for every new target dataset. Using our unique algorithm, 15,000,000 synthetic samples of UTS data with various angles, sequence lengths, and patterns were used to pretrain our CNN (convolutional neural network) model.

Using 85 datasets from the UCR archive as target datasets, we perform a comprehensive evaluation of our method. For datasets with seasonal characteristics, when the amount of training data was reduced to 10%, our method outperforms all other TSC methods (both TL and non-TL methods) on 17 of the 34 seasonal datasets in the UCR archive, whereas the second-best methods outperform on only seven of the 34 seasonal datasets.

Additionally, using our method improves the test set's accuracy while reducing training time by 85%, without compromising performance. We thus believe that our method can serve as a good starting point for any new target dataset.

The contributions of this paper are as follows:

1. ***Synthetic UTS data and regression task generator algorithm:*** In this paper, we contribute a new architecture-agnostic TL for TSC method. Unlike previously proposed TL for TSC methods which use an existing source dataset and classification task from the UCR archive [29], we propose a new algorithm which generates synthetic UTS data and creates 55 corresponding regression tasks which can be used as a source dataset and task.
   Using existing datasets from the UCR archive as the source dataset has some limitations that our synthetic data overcomes. First, given a target dataset, datasets from the UCR archive may not always be similar or generic enough to serve as a good source dataset. Since our synthetic 15,000,000 sample dataset has a wide variety of patterns, angles, and sequence lengths, it could be a more generic source dataset and therefore be a better fit. A second limitation is that using UCR datasets is not scalable: each update to the UCR archive requires that TL for TSC methods perform a new pretraining procedure to incorporate the new datasets. Since our method relies on the synthetic 15,000,000 sample dataset as a source dataset, no additional pretraining is necessary. Finally, searching for the optimal source dataset from the UCR archive can be time- and resource- consuming. Since we do not use datasets from the UCR but instead use our synthetic dataset, no such search is needed.
   In this paper, we demonstrate the superiority of a dataset consisting of synthetic data over existing datasets from the UCR archive, by addressing all of the above mentioned issues.
2. ***Code contribution:*** Our code[1] includes the following:
   - ***UTS data and regression tasks generator:*** We created an algorithm to generate synthetic UTS data with a wide range of UTS patterns, angles, and sequence lengths that can serve as a source dataset.
   - ***Complete framework:*** a comprehensive easy-to-use framework that covers data and regression task generation through fine-tuning the pretrained model on a new target dataset and task.
3. We publish both the synthetic dataset with 15,000,000 UTS samples and the pretrained CNN model with the *CTN* architecture that was pretrained on that dataset, making them available for use by researchers and the entire ML community.

The remainder of the paper is structured as follows: In Sect. 2, we provide the necessary background and introduce related work on TSC and TL for TSC methods. Following this, in Sect. 3, we describe our method, from data generation through fine-tuning the pretrained model on a new target dataset and task. In

[1] Code availble at: https://github.com/YR234/TL-for-TSC.

Sect. 4, we describe the experimental setup, while Sect. 5 presents our results. Finally, in Sect. 6, we present our conclusions and plans for future work.

2 Background and Related Work

In this section, we first discuss on related work regarding TSC and TL for TSC methods, and we highlight the differences between those methods and ours.

2.1 TSC Related Work

In this subsection, we discuss previously proposed TSC methods.

MultiRocket [1] is a TSC method that achieved SOTA (state-of-the-art) results on the entire UCR archive [29] at a rate orders of magnitude faster than any other competing method.

MultiRocket is, in practice, a single-layer convolutional neural network, where the transformed features from the convolutional kernels form the input for a linear classifier.

MultiRocket uses as many as 10,000 convolutional kernels with a wide range of length, padding, dilation, and random weights. After the kernels are generated, each kernel is applied to each input time series, resulting in a feature map. MultiRocket then computes a set of features from the feature map that includes PPV (portion of positive values) plus a randomly selected features from a set of five candidate features. These features serve as the input for a linear classifier, such as a ridge regression classifier or logistic regression.

MultiRocket does not use a nonlinear function or have any hidden layers, thus allowing it to be orders of magnitude faster than any other method.

OS-CNN [28] is a TSC method that uses omni-scale (OS) blocks, which does not need to tune the feature extraction scales. Usually, a core challenge of a CNN is to determine the proper scales of feature extraction. This method uses OS blocks which are made up of OS layers that can be configured automatically from the input size based on a list of kernel sizes; by stacking those layers, this method can achieve full receptive field coverage of the total length of the input (sequence length) [14].

InceptionTime [?] is a TSC method that uses an ensemble of five deep CNN models, which was inspired by the Inception-V4 [27] architecture. This architecture includes several techniques commonly used when constructing a CNN model, such as residual block with shortcut connections [14] and inception modules [27]. Each of the five models is given equal weight in the final prediction decision.

Because of our TL-based approach, our method differs entirely from the TSC methods mentioned above. In the absence of sufficient labeled data, TL techniques are useful. In this paper, we leverage this by reducing the amount of labeled training data to 10%. Our experimental results indicate that when it comes to seasonal datasets, our method outperforms all other methods, and with all datasets (both seasonal and non-seasonal) our method is only second to MultiRocket, however the difference in the performance of the two methods was not shown to be significant when the Nemenyi statistical test was performed (Fig. 1).

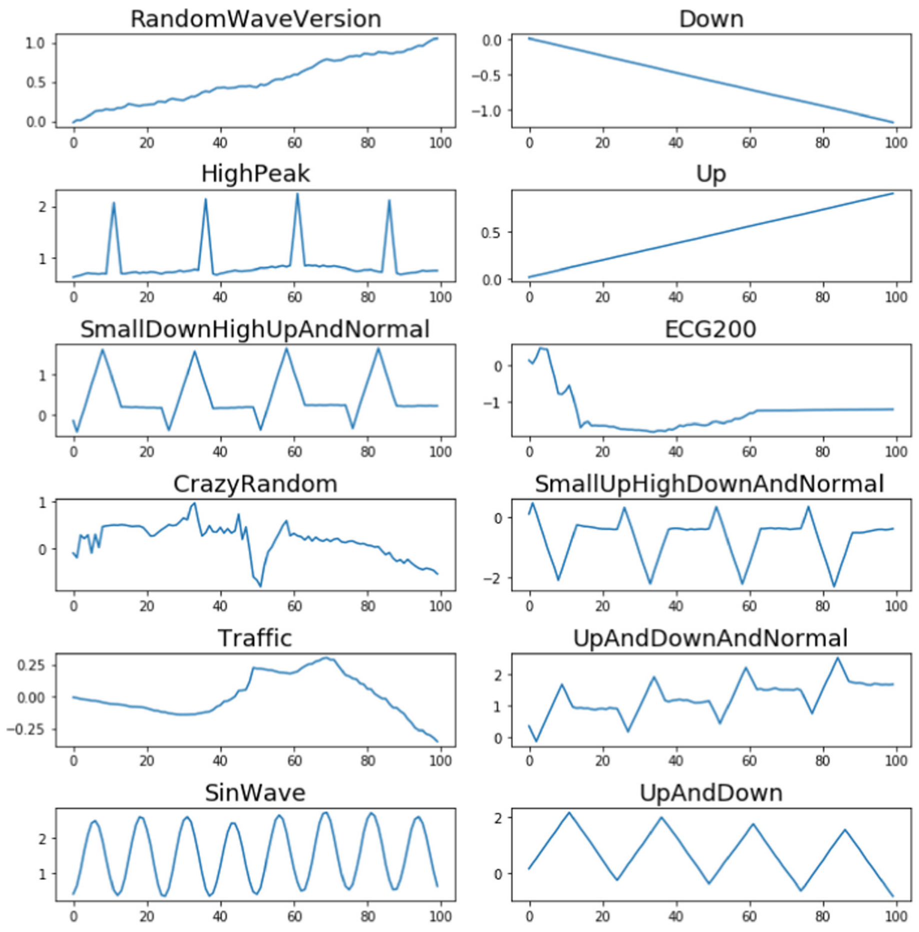


Fig. 1. The 12 UTS patterns generated in our work.

2.2 TL for TSC Related Work

The use of TL for TSC has been proposed in a number of studies. In this subsection, we discuss the existing TL for TSC methods and how our TL for TSC method differs from these methods.

An overview of the general TL for TSC process is presented in Fig. 2. This process consists of the following five steps: First, a source dataset is selected. Second, a source task is selected. In step 3, the model's architecture is chosen. In step 4, the model chosen in step 3 is pretrained on the source dataset and task selected respectively in steps 1 and 2. The final step consists of fine-tuning the pretrained model from step 4 on a new target dataset and task.

While all previous TL for TSC studies used existing datasets and classification tasks from the UCR archive as the source dataset and task for steps 1 and 2, in this paper, we generate synthetic data for the source dataset and use regression tasks instead of classification as the source task, and demonstrate how those two decisions can result in better generalization while eliminating the need for an exhaustive search for the best source dataset.

Fawaz et al. [6] suggested using DTW (dynamic time warping), a technique for finding the optimal alignment between two given time series sequences [7], as a similarity measure for finding the most similar source dataset from the UCR archive. The source task is chosen according to the source dataset (provided by the UCR archive).

While our method may only differ from the method of Fawaz et al. in terms of steps 1 and 2 of the TL for TSC process, our novel approach for creating the source dataset and task from synthetic data and regression tasks instead of using an existing dataset and classification task from the UCR archive addresses other issues that we will discuss later in the paper.

Our experimental results on the UCR archive showed that the method proposed by Fawaz et al. performed positive transfer learning on 71/85 datasets, however this approach has some disadvantages.

Kashiparekh et al. [30] suggested using a convolutional neural network (CNN) based architecture with a multi-head approach for training a given S source dataset (D_S) and corresponding S classification tasks (T_S) from the UCR archive.

The CNN core architecture consists of convolutional layers followed by skip connections [14], which make this architecture a deep one. However, instead of standard fully connected layers followed by a dense layer with the softmax activation function, the authors used S fully connected layers and S dense layers with the softmax activation function - one for each source dataset and task.

The authors randomly selected $S = 24$ datasets from the UCR archive for training and validation, and the remaining 41 datasets were used as test sets (the authors used sequence lengths up to 512, and therefore not all 85 datasets of the UCR archive were evaluated).

As noted earlier, none of these methods provides a real solution when it comes to real-world problems in the domain of TL for TSC. Since they are limited to the available datasets in the UCR archive, they may not always be able to find the optimal source dataset. Not only that, when using the method proposed by Fawaz et al., one would have to perform an exhaustive search to find the most similar source dataset for a new target dataset and task.

In contrast to prior work, our method does not require an exhaustive search, and it is not restricted to datasets available in the UCR archives or any specific sequence length. Since it is based on diverse synthetic data that was generated by our new algorithm, it can be applied to a variety of new target datasets and tasks.

3 Method

In this section, We will discuss on our five-step TL for TSC method (see Fig. 2). The first two steps describe the process of generating the source dataset and regression source task. We then describe steps 3–5 where we select and pretrain the CNN architecture and fine-tune the pretrained model on a new target dataset and task.

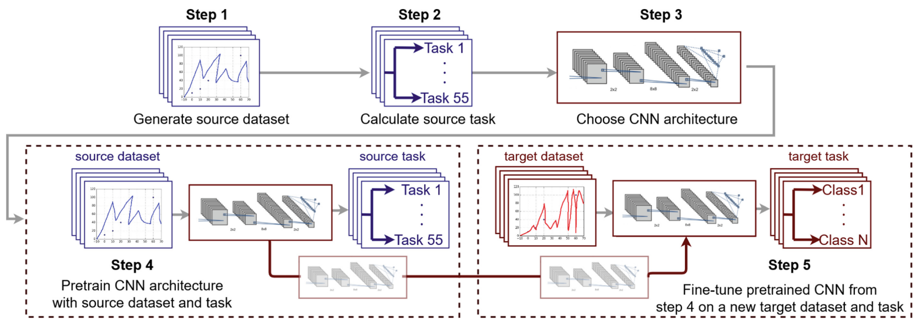


Fig. 2. Method overview: In step 1, we generate a 15,000,000 sample UTS source dataset using our Algorithm. After that, we calculate 55 regression tasks for each UTS in the source dataset to be our source tasks. In step 3, we select the CNN architecture (we chose to use *CTN*). Then in step 4, we train the CNN with the source dataset and task. Finally, in step 5, we fine-tune the pretrained CNN model on a new target dataset and task.

3.1 Data Generation - Source Dataset

We created a synthetic UTS data generator algorithm; using this algorithm, we created a 15,000,000 sample dataset that contains a wide range of UTS with different segment patterns, angles, sequence lengths. This dataset will serve as our source dataset D_S.

In our study, we generated only 12 UTS patterns. However, using our algorithm, many more patterns can be generated.

3.2 Data Generation - Source Tasks

Upon generating the source dataset, we proceed to the source task. Because our target task is TSC (classification), it would be natural to use classification as our source task, however when comparing classification and regression as source tasks, we found that regression achieves more accurate results, and therefore it was chosen as our source task.

1. The 55 tasks are:
    1. **Maximum (Task 1):** Given an input UTS, the purpose of the task is to accurately predict the maximum value of the UTS.

2. **Minimum (Task 2):** Given an input UTS, the purpose of the task is to accurately predict the minimum value of the UTS.
3. **STD (Task 3):** Given an input UTS, the purpose of the task is to accurately predict the STD (standard deviation) value of the UTS.
4. **Peaks (Task 4):** Given an input UTS, the purpose of the task is to accurately predict the number of high and low peaks.
5. **Cross median (Task 5):** Given an input UTS, the purpose of the task is to accurately predict the number of times the UTS crosses the median value from up to down and vice versa.
6. **10 splits (Tasks 6–55):** Given an input UTS, we first divide the UTS into 10 equal length segments. For each segment we calculate tasks 1–5 and concatenate them into a 50 value task (10 segments * 5 tasks).

3.3 CNN Model's Architecture

Our method is architecture-agnostic, meaning that any deep learning network with a convolutional layer based architecture (CNN) can be used. In our research we used the same CNN architecture as Kashiparekh et al. [30] whose work showed it to be an effective architecture for TL. Just one change was made to their architecture; unlike the multi-head approach used by Kashiparekh et al., we used only one dense layer with the sotfmax activation function. This architecture will be denoted as CTN.

3.4 CNN Pretraining

The next step of our method is pretraining the CTN model on the source dataset D_S with the source task T_S.

To create the training and validation sets, we randomly divided the source dataset D_S into an 80%–20% split. We pretrained the CTN model for 100 epochs with a batch size of 128, while performing early stopping on the validation loss. We used the *Adam* [25] optimizer and MSE (mean square error) as the loss function.

The pretraining process was lengthy, taking almost 10 days on a single GPU processor, however this process only needs to happen once. Once the CTN model has been pretrained, we save the weights of the model's core layers. These weights are used later to fine-tune new target datasets and tasks.

3.5 Fine-Tuning a New Target Dataset

The final step of our method is to fine-tune the pretrained CTN model on a new target dataset D_T with a new target task T_T.

We start by initializing the CTN model's core layers with the pretrained weights that were saved in the last step. Once initialization is complete, newly added fully connected layers can be adjusted so that they better fit the new target dataset and task.

4 Experimental Setup

In this section, we describe our experimental setup, including the datasets, preprocessing, and settings used, as well as the methods we compare our method to.

4.1 Datasets

To evaluate our method, we used the UCR archive, which is the benchmark archive for TSC. In 2002, the archive contained 45 datasets; this increased to 85 datasets in 2015, and as of 2019, 128 datasets are available. All of the datasets contain UTS samples. The archive's datasets vary in terms of the time series domain covered; the domains include traffic, sound, sensors, motion, image, HAR (human activity recognition), financial, medical, and more. Furthermore, they are diverse in terms of the sequence length (ranging from 8–5,000), number of classes (2–60), number of training samples (12–139,000), and number of test samples (15–139,000). Due to running time considerations, we evaluated our results on just the 85 dataset version of the UCR archive and not on the most update version that includes 128 datasets.

4.2 Data Preprocessing (Reducing the Labeled Training Data to 10%)

The original train-test split provided by the UCR archive was used. However, instead of using all of the training data, we reduced each dataset's training data to only 10% of the original, while keeping the same class distribution, e.g., given a dataset of 100 training samples with 70 samples of class 1 and 30 samples of class 2, we reduced the training data to 10 samples, with seven class 1 samples and three class 2 samples. Therefore, the 70–30% class distribution was maintained.

The following are some key points regarding the reduction process:

1. The remaining 10% of the training samples were chosen at random.
2. The reduction process was only performed once.
3. All of the test data samples were evaluated (the test data was not reduced).

This reduction process was used to emphasize the importance of transfer learning, since when there is a lack of labeled data, the pretraining process (steps 1–4 of our method) is expected to provide a better start for learning a new target dataset and task than learning from scratch.

4.3 Methods Used for Comparison

We compare our method, which will now be denoted as CTN_our, to the methods covered in the related work section: Fawaz et al. [6] (denoted as $Fawaz$), Kashiparekh et al. [30] (denoted as $ConvTime$), MultiRocket [1], OS-CNN [28], and InceptionTime [10]; we also examine the CTN architecture without the pretraining phase (steps 1–4 in our method), which is denoted as CTN_S (no transfer learning was applied). We included CTN_S, so we can examine our results in terms of positive and negative transfer learning and more.

4.4 Hyperparameters and Other Settings

For all deep learning methods (Fawaz et al., Kashiparekh et al., OS-CNN, Inception-Time), including ours, we trained each dataset with 2,000 epochs, using cross-entropy [2] as the loss function and Adam [25] as the optimizer.

For MultiRocket (a linear classifier), we used the default parameters provided by the authors.

4.5 The Evaluation Process

The evaluation process includes applying all of the methods on the datasets with the necessary data preprocessing and with the hyperparameters - all this was described at this section.

A summary of the results is provided in the next section.

5 Results

This section begins with a brief summary of the results. We then explore each aspect mentioned in the brief summary in more detail.

5.1 Results Appendices

A summary of the results can be seen in Table 1 and a more visual representation can be found in Fig. 3.

5.2 Brief Summary of the Results

1. Thirty-four of the 85 UCR archive datasets have seasonality characteristics. Of these datasets, our method outperforms all other examined methods on 17 datasets; the second best method only outperforms all other methods on seven datasets.

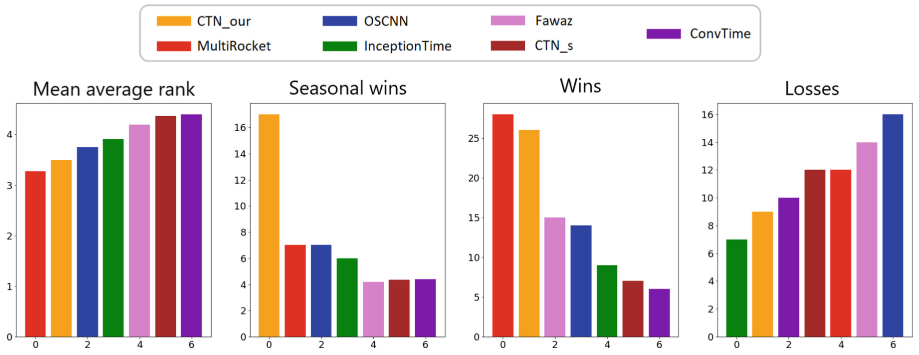


Fig. 3. Empirical results, from left to right: mean average rank, seasonal wins, wins, losses. Each method is associated with a colored bar.

Table 1. Summary of the results in terms of the number of wins, number of losses, seasonal wins, and mean average rank

Method	Wins	Losses	Seasonal wins	Mean avg rank
ConvTime	6	10	3	4.4
CTN_S	7	12	4	4.365
Fawaz	15	14	4	4.2
InceptionTime	9	***7***	6	3.906
OSCNN	14	16	7	3.753
CTN_our	26	9	***17***	3.494
MultiRocket	***28***	12	7	***3.271***

2. Positive transfer learning occurs in all 2,000 epochs except the first seven epochs, and using our method can save 85% of the training time while achieving the same results (see Fig. 4).
3. Our method obtains a mean average rank of 3.494, which is second only to MultiRocket with a mean average rank of 3.271 (the difference between the two values is not significant. In terms of the win/lose rate, our method obtains a 26/9 rate, while MultiRocket's rate is 28/12; MultiRocket has two more wins, but it also has three more loses than our proposed method.
4. As can been Fig. 3, our method comes in at least second place in each case, something no other method achieved.

5.3 Seasonality Evaluation

We first evaluate the results in terms of seasonality. The autocorrelation values of a given UTS are in the range of $[-1, 1]$. Generally, as the autocorrelation values approach zero there is no seasonality and vice versa. So, in this paper, we took the absolute value of the autocorrelation function, and the values will eventually be in the range of $[0, 1]$. Higher absolute values of the autocorrelation function indicate strong seasonality and vice versa.

To empirically define datasets with seasonality we calculated a seasonality metric for each dataset, which is denoted as SM (seasonality metric).

All datasets with SM => 0.5 will be considered as seasonal datasets. The results show that 34 of the 85 UCR archive datasets are seasonal datasets. Of these 34 datasets, *CTN_our* outperforms all other methods on 17 datasets. The second best methods are OSCNN and MultiRocket which outperform other methods on only seven of the 34 seasonal datasets. These results are presented in Table 1 and more visually in Fig. 3. These results indicate that our method is superior when it comes to seasonal datasets.

5.4 Positive Transfer Learning

To demonstrate that transfer learning using our method can improve the performance of a given CNN architecture, we compered the following two methods:

CTN_our and *CTN_S*. The comparison was made while considered the following three aspects: positive transfer learning after each epoch, training time, and final accuracy. The results can be seen in Fig. 4.

A more detailed evaluation for each of the aspects mentioned above is provided below:

1. ***Positive transfer learning:*** Except for the first seven epochs, *CTN_our* outperforms *CTN_S*. Accordingly, we can conclude that our method's performance improves when training continues beyond the first few initial training epochs.
2. ***85% Less training time:*** It took *CTN_S* 1,515 epochs to achieve its highest level of accuracy on all 85 datasets (indicated by the grey 'X' in Fig. 4), whereas *CTN_our* attains the same level of accuracy after only 193 epochs. In other words, using our method can save as much as 85% of the training time and still achieve the same generalization.
3. ***Accuracy:*** When considering the average accuracy after 2,000 epochs across the 85 datasets from the UCR archive, *CTN_our* outperforms *CTN_S*, as can be see in Fig. 4.

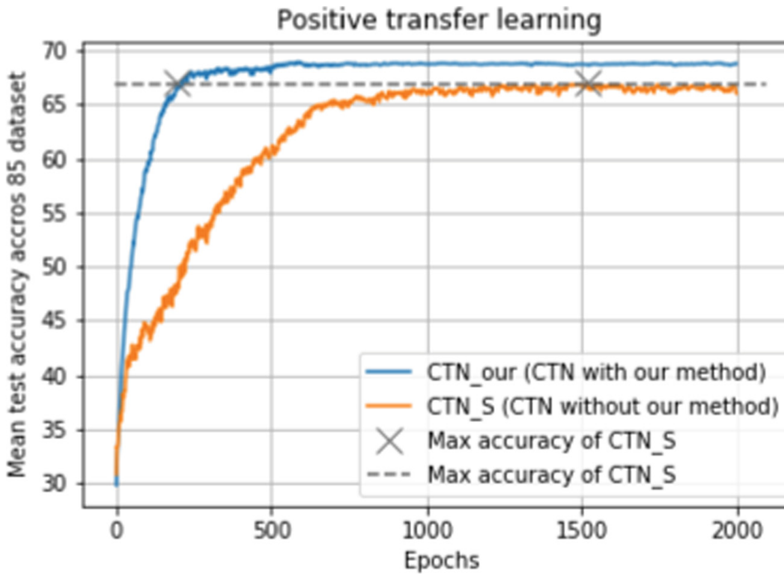


Fig. 4. Performance of the *CTN* architecture on all of the test sets from the UCR archive after each epoch, with and without our method.

5.5 Mean Average Rank and Win/Lose Rate

In terms of the mean average rank, *CTN_our* achieved a score of **3.494**, which is only second to MultiRocket with a score of 3.271; the Nemenyi statistical test indicated that there was no significant difference in the results of the two methods

(see Table 1 which presents the mean average rank of all of the examined methods).

In terms of the win/lose rate, our method obtains a **26/9** win/lose rate. While MultiRocket achieves a 28/12 win/lose rate, winning two more times but losing three more times. InceptionTime has the fewest losses, with a win/lose rate of 9/7.

Our method comes in at least second place on every empirical measure (mean average rank, seasonal wins, win, lose), which no other method is capable of (the results of all of the examined methods can be seen in Table 1).

6 Conclusion and Future Work

In this paper, we introduced:

1. A novel architecture-agnostic TL for TSC method. In contrast to previous TL for TSC methods using existing UCR's datasets and tasks as the source dataset and task, in this paper, we introduce a new algorithm that generates UTS data and creates 55 corresponding regression tasks to be used as a source dataset and task.
2. Open-source code for generating custom synthetic data, producing regression tasks, pretraining, and fine-tuning on a new target dataset and task.
   Our 15,000,000 sample synthetic dataset, the 55 regression tasks, and the pretrained model with the CTN architecture are published for further use by the ML community.

Our study shows that the use of our method can not only improve the performance of a given CNN architecture but also decreases training time by 85%.

When it comes to seasonal datasets, our method outperforms all other existing TSC methods.

For future work, we would like to do the following:

1. Explore other regression tasks beside the 55 tasks we used. The new tasks could include Fourier transform, autocorrelation, etc.
2. Examine a new architecture for transfer learning based on both CNN and LSTM layers which will be trained on our source dataset and task, as suggest by Wang et al. [23].
3. Expand our source dataset to include more patterns than the 12 patterns we generated.
4. Extend our method so that it will be suitable for MTS (multivariate time series) datasets.
5. Explore the use of a generative adversarial network (GAN) to automatically generate synthetic data that is more similar to a specific given dataset.

References

1. Tan, C.W., Dempster, A., Bergmeir, C., Webb, G.I.: MultiRocket: effective summary statistics for convolutional outputs in time series classification arXiv preprint arXiv:2102.00457 (2021)
2. De Boer, P.-T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Ann. Oper. Res. **134**, 19–67 (2005)
3. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
4. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009)
5. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding arXiv preprint arXiv:1810.04805 (2018)
6. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.-A.: Transfer learning for time series classification. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 1367–1376. IEEE (2018)
7. Müller, M.: Dynamic time warping. In: Bundy, A., Wallen, L. (eds.) Information Retrieval for Music and Motion, pp. 69–84. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-642-96868-6_63
8. Menegola, A., Fornaciali, M., Pires, R., Bittencourt, F.V., Avila, S., Valle, E.: Knowledge transfer for melanoma screening with deep learning. In: 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017), pp. 297–300. IEEE (2017)
9. Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. arXiv preprint arXiv:2106.04560 (2021)
10. Ismail Fawaz, H., et al.: InceptionTime: finding alexnet for time series classification, ArXix (2019)
11. Ferreira, L.N., Zhao, L.: Time series clustering via community detection in networks. Inf. Sci. **326**, 227–242 (2016)
12. Sagheer, A., Kotb, M.: Time series forecasting of petroleum production using deep LSTM recurrent networks. Neurocomputing **323**, 203–213 (2019)
13. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.-A.: Deep learning for time series classification: a review. Data Min. Knowl. Disc. **33**(4), 917–963 (2019). https://doi.org/10.1007/s10618-019-00619-1
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
15. Yang, Q., Wu, X.: 10 challenging problems in data mining research. Int. J. Inf. Technol. Decis. Mak. **5**(04), 597–604 (2006)
16. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Proceedings 2001 IEEE International Conference on Data Mining, pp. 289–296. IEEE (2001)
17. Papadimitriou, S., Yu, P.: Optimal multi-scale patterns in time series streams. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 647–658 (2006)
18. Zhu, Y., Shasha, D.: Statstream: statistical monitoring of thousands of data streams in real time. In: VLDB 2002: Proceedings of the 28th International Conference on Very Large Databases, pp. 358–369. Elsevier (2002)

19. Wang, L., Wang, Z., Liu, S.: An effective multivariate time series classification approach using echo state network and adaptive differential evolution algorithm. Expert Syst. Appl. **43**, 237–249 (2016)
20. Karim, F., Majumdar, S., Darabi, H., Harford, S.: Multivariate LSTM-FCNs for time series classification. Neural Netw. **116**, 237–245 (2019)
21. Zhao, W.: Research on the deep learning of the small sample data based on transfer learning. In: AIP Conference Proceedings, vol. 1864, no. 1, p. 020018. AIP Publishing LLC (2017)
22. Zhuang, F., et al.: A comprehensive survey on transfer learning arXiv preprint arXiv:1911.02685 (2019)
23. Wang, J., Wang, W., Wei, S., Zeng, Y., Luo, F.: Time series sequences classification with inception and LSTM module. In: 2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA). IEEE, pp. 51–55 (2019)
24. Hawkins, D.M.: The problem of overfitting. J. Chem. Inf. Comput. Sci. **44**(1), 1–12 (2004)
25. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014)
26. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
27. Ismail Fawaz, H., et al.: InceptionTime: finding AlexNet for time series classification. Data Min. Knowl. Disc. **34**(6), 1936–1962 (2020). https://doi.org/10.1007/s10618-020-00710-y
28. Tang, W., Long, G., Liu, L., Zhou, T., Jiang, J., Blumenstein, M.: Rethinking 1d-CNN for time series classification: a stronger baseline arXiv preprint arXiv:2002.10061 (2020)
29. Dau, H.A., et al.: The UCR time series archive. IEEE/CAA J. Automatica Sinica **6**, 1293–1305 (2019)
30. Kashiparekh, K., Narwariya, J., Malhotra, P., Vig, L., Shroff, G.: ConvTimeNet: a pre-trained deep convolutional neural network for time series classification. In: 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2019)

Non-stopping Junctions via Traffic Scheduling

Shlomi Dolev, Ehud Gudes, and Hannah Yair(✉)

Ben-Gurion University of the Negev, Be'er Sheva, Israel
{dolev,ehud}@cs.bgu.ac.il, hannaya@post.bgu.ac.il

Abstract. Emergency situations involve massive movements of (logistic and units) platoons to and from focal locations. Platoons may move in different directions and can be blocking each other in junctions causing even deadlocks. The possibility to minimize the delay in junctions, in particular, non-stopping and waiting for a (virtual) green light, may avoid the chain phenomena of cascade stopping and cascade starting to move again when all cars wait for the car in front of them to gain enough velocity. The remote driving system is an opportunity to stream all platoons driving in different directions without stopping, by spacing vehicles to allow conflicting traffic to move in the space between vehicles. In this work, we present briefly the algorithms to identify and control platoons and focus on the real-time junction scheduling towards the non-stopping junction(s). We demonstrate the results that imply road safety as actions are remotely controlled, by using the SUMO simulator [7].

Keywords: Scheduling · Platoon · Junction · Virtual traffic light · Autonomous vehicles

1 Introduction

Managing and scheduling transportation at crossroads efficiently while minimizing the travel time is a challenging task and is a task that is re-investigated in the scope of autonomous and computer (distant) controlled vehicles. One of the bold challenges is to find novel policies and algorithms to schedule vehicles through junctions, without waiting for a green light at the junction that causes traffic jams and delays the overall flow of traffic on the road.

The assistance of a computer system can leverage the capabilities of single procedures to be applied to a group of autonomous vehicle's procedures that when executed on the controlling computer initiate commands to many autonomous vehicles in parallel. In particular, such procedures are based on the concept of *platoon* and *traffic policies*. A platoon is defined as a group of cars moving in the same direction as a selected car, called, the *representative vehicle*, and at a distance of each other which is less than a given *threshold*.

This research was partially funded by the Andromeda MAGNET Consortium, by the Lynne and William Frankel Center for Computer Science and by Rita Altura chair in computer science.

The advantages of creating a platoon are well-known [12] and include increased safety and fuel saving. Traffic policies are liable for managing and scheduling transportation. The contributions of our work are the new dynamic and history agnostic definition of *platoons* and the control of the transportation by *virtual traffic policies* that focus on *virtual traffic lights policy.* The meaning behind *virtual traffic policies* is to give remote instructions to autonomous vehicles on the road, these instructions fall into two categories (weak and strong), where a weak instruction is an autonomous instruction in which the vehicle from the moment of the instruction knows how to implement it by its internal autonomous systems, compared to a strong instruction that is a complete remote control of all vehicle systems (as can be seen later in Sect. 4) to ensure full safety and cyber security. In the sequel, we present our on-the-fly dynamic definition of platoons and the outline of the algorithm. We also demonstrate the flexible and robust platoon definition using, SUMO, a well-known vehicles simulator [7].

The rest of this paper is structured as follows. Section 2 shortly discusses platoon identification problem including the algorithm explanation and results. Section 3 describes the traffic policies and motivation to solve them. Section 4 presents the virtual traffic light (VTL) policy, and related work. Section 5 presents the implementation of the VTL algorithm, and the analysis of the algorithm. Section 5.1 presents an extension of the concept of the algorithm for scheduling autonomous vehicles in all the parts of a road together. Lastly, Sect. 6 concludes the paper.

2 Automatic Real-Time Platoon Problem and Algorithm

As previously mentioned, a platoon is defined as a group of vehicles moving in the same direction around a selected representative vehicle. The platoon consists of every vehicle that is at a distance less than a threshold distance from a vehicle that belongs to the platoon.

Outline of the Algorithm: First, we build a scheme that represents the map area as a directed graph whose nodes are junctions and the edges represent the lanes connecting the junctions. This scheme is built by utilizing the transformation part of the SUMO simulator [7], which can be based on google maps to a directed graph. When the representative vehicle R is received, the information of R and the vehicles that are on the same edge of R is collected. This information is then sorted into two groups *pos*, *neg* where *pos* is the group of vehicles in front of R and *neg* includes the cars that are behind R. These groups were built in an iterative fashion, adding all vehicles that are within the threshold distance from the last added vehicles. Once all the vehicles on the current edge are considered, the platoon calculation propagates along the edges that are adjacent to the edge R resides on. The full implementation and analysis of this algorithm was presented in our previous work [4].

In Fig. 1, the illustration shows the algorithm operation by three arrows that describe the calculation of the platoon members. One straight arrow with the red vehicle direction describes the calculation operation of the members in front of the red vehicle, and the two other arrows describe the calculation operation of the members that are behind of the red vehicle.

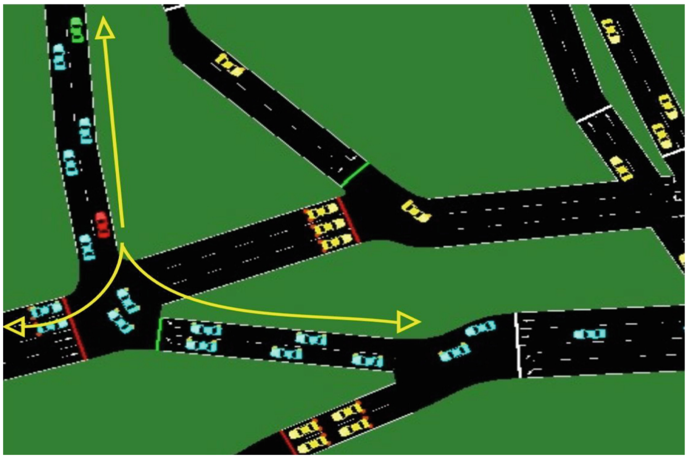


Fig. 1. Simulator snapshot (Color figure online)

3 Traffic Signs Policies

In this section, we present shortly several virtual traffic signs/policies and their possible usage in practice. The policies can be used by a traffic controller at a control center, which receives information on road conditions, accidents, traffic jams, etc., throughout an entire region. The virtual signs can control the platoon's movements and enable a smoother traffic flow overall by solving the unusual conditions. In the next section, we detail the algorithm used to implement the VTL policy. The full implementation of the other traffic policies algorithm is presented in our previous work [4].

Policies List

– Follow the leader:

$$pltnSign(Follow, p) = the\ vehicles\ in\ platoon\ p\ follow\ the\ leader \tag{1}$$

– Speed limit for a platoon:

$$pltnSign(maxSpeed, p) = speed\ limit\ vehicles\ in\ platoon\ p \tag{2}$$

– Speed limit on a lane for a platoon:

$$pltnSign(maxSpeed, p, l) = speed\ limit\ vehicles\ in\ platoon\ p\ on\ lane\ l \tag{3}$$

– Lane block for all kind of vehicles:

$$laneSign(block, l) = no\ entry\ to\ lane\ l \tag{4}$$

– Distance keeping among vehicles in a platoon:

$$pltnSign(minGap, p) = safe\ distance\ limit\ vehicles\ in\ platoon\ p \tag{5}$$

– Virtual Traffic Lights:

$$VTLSign(junction, type) = schedule\ the\ up\ coming\ vehicles \tag{6}$$

The traffic policies will be controlled and changed according to the scenario by several algorithms [4]. The next section will discuss the traffic lights policy problem in detail.

4 Virtual Traffic Lights

The main idea of the Virtual Traffic Lights is to synchronize the vehicles coming to the junction in such a way that vehicles do not stop at all. They will not stop to let others to cross the junction or to wait for others to evacuate the junction. Under the assumption that all the vehicles in the road are autonomous vehicles and computer controlled, the solution for Virtual Traffic Lights is early handling of junction management. That is, calculating the timing at a reasonable distance D before the junction and maintaining a sufficient distance between the vehicles before the junction, allowing flexibility of deceleration or acceleration of the vehicles in order for them to cross the junction as safely and quickly as possible.

The timing principle is "first come, first served". That is, the vehicles that are closer to the junction by a given distance D are the ones that are timed before crossing the junction. Additionally, the vehicles arriving at the junction will cross it at high speed to evacuate the junction as quickly as possible. Thus there is no case of excessive deceleration that could cause an abnormal traffic load in a particular lane entering the junction. In other words, the order of priority between the lanes entering the junction is equal. That is why there is no starvation; this principle produces a junction in which the vehicles cross in analog to a zipper.

Note that for the platoons mentioned above that cross the junction in the policy presented now, the only vehicles leaving the platoon are vehicles that turned in a different direction than the direction of the representative vehicle. Because giving priority to crossing the junction is equal between all lanes and vehicles cannot stop on the road, the distance between the vehicles in the platoon that turn in the same direction of the representative vehicle will increases slightly. This will prevent the platoon to disintegrate.

In Fig. 2, a snapshot shown from the algorithm for vehicle synchronization running at a non-stop junction that slightly describes the zipper situation, and it can be seen that the lights in all directions at the junction are green. This figure depicts the equal order of priority between all lanes entering the junction. Note that in practice, there is no need for a physical traffic light. A demonstration video can be found in [1].

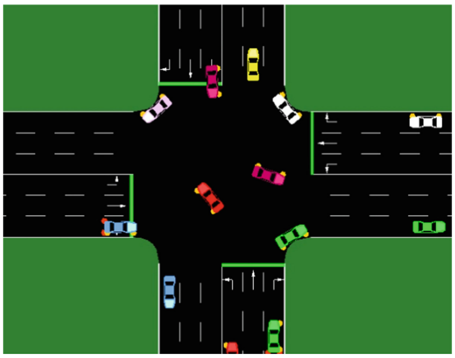


Fig. 2. Simulator snapshot (Color figure online)

Related Work. We reviewed several recent Virtual Traffic Lights (VTL) and scheduling traffic projects. These are described and compared briefly in the following table. Table 1 summarizes the main differences among recent VTL and scheduling traffic projects.

Table 1. Comparison of VTL projects, Projects (NA stands for Not-Applicable)

Reference	Communication type	Vehicle type	Junction type	Purpose	Off/On line problem	Scheduling algorithm
Juan et al. [10]	NA	Autonomous vehicle	One-way, Y merges, Multiway merges, and Two-way crossing	Providing polynomial-time algorithms for the cases of a k-way merge (for constant k) and for a crossing involving two-way traffic for platoons	Offline	Based on dynamic programming and parametric search techniques
Olaverri-Monreal et al. [9]	vehicle-to-vehicle	VTL support equipped vehicles	Signalized intersections	Improve traffic flow and reduce collisions	Online	Standard rules
Dolev et al. [5]	NA	NA	NA	Schedule packets in high-speed networks avoids any possibility of packet collision at a switch or a link	NA	Based on Euler tour and tokens, when the token moves without collision according to Euler tour
Kok et al. [8]	Intervehicle	NA	Any type	The authors proposed platoon interaction algorithms to the LWR-IM model to describe platoon interactions in urban arterial	Online	Algorithm (LinkModelMJ)
Ning et al. [6]	Vehicle infrastructure	Autonomous Vehicle	Signalized intersections	Improved platoon-based adaptive control strategy to provide multimodal traffic management for signalized intersections, assuming that the connected vehicle information is available online	Online	A mixed-integer linear programming (MILP) model is proposed to optimize signal timings in a real-time manner

(*continued*)

Table 1. (*continued*)

Reference	Communication type	Vehicle type	Junction type	Purpose	Off/On line problem	Scheduling algorithm
Hugo et al. [3]	Vehicle-to-vehicle, VTL protocol	VTL support equipped vehicles	Single unsignalized intersection single lane/approach 125 m	Redesign of the VTL protocol to include exterior lights on equipped vehicles can solve the problem of the co-existence with non-equipped vehicles	Online	Standard rules
Alessandro et al. [2]	Vehicle-to-vehicle	VTL support equipped vehicle	NA	To design a mechanism that allows vehicles to autonomously solve priorities at road junctions in the absence of fixed infrastructures (i.e., conventional traffic lights)	Online	Standard rules
Wantanee et al. [11]	Vehicle-to-vehicle	VTL support not/equipped vehicles	Signalized intersections	Outstanding issue by proposing a transition model in which VTL-vehicles benefit from the VTL technology while co-existing and sharing the streets with current non-VTL vehicles	Online	Vehicles self-organize to select a leader which serves as a virtual traffic lights to decide the right of way at that intersection
Our work	Vehicle infrastructure, using GPS	Semi/fully autonomous vehicles	Any type	To schedule vehicle to intersection without stopping	Off/Online	Based on conflict graph and timing indicator

According to the table, it can be seen that the contribution in our project compared to the existing projects is the calculated manner of scheduling, different from the standard transition rules at a traffic light junction, but in a zippered manner. In addition, it can be noted that the algorithm does not require much data and unique models to implement it, nor does it require physical systems at the junction to communicate with vehicles.

The next section presents the new algorithm that we have developed for the traffic lights policy problem.

5 Virtual Traffic Light Algorithm

The algorithm idea is to utilize the junction as much as possible i.e., as many vehicles as possible cross the junction together and cross the junction fast as possible, and the most important thing does not to cause vehicles to stop before or at a junction but only slightly slow.

5.1 Outline of the Algorithm

First, a scheme was build that represents the map area as a directed graph whose nodes are junctions and the edges represent the lanes connecting the junctions, we can transform such map from Google Maps. At each given time unit, all of the vehicles who are close to the junction at distance D are collected. A conflicts graph is built based on the given location of the vehicles in the directed graph and by the junction type. Then, with this information a schedule is built with the necessary crossing time for each vehicle. As long as the conflicts graph is not empty, we list the maximum of nodes that do not conflict and calculate their crossing time according to the parameter *clearIndicator* that indicates when the junction is empty. Then we update the list's vehicles velocity in accordance with the distance and the parameter *clearIndicator*. Finally, the parameter *clearindicator* will be updated as per the new time.

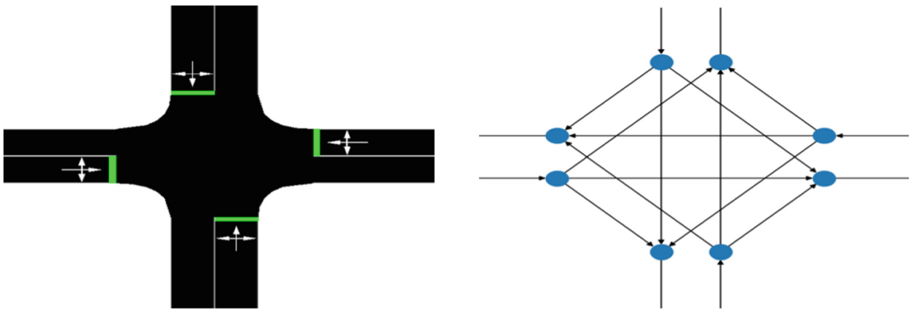


Fig. 3. Map transformation

In Fig. 3, the transformation of part of the SUMO simulation [7] map is presented (SUMO, can be based on Google Maps). This is mapped into a directed graph in order to calculate the scheduling time for the upcoming vehicles. The lanes are represented by directed edges and the nodes represent the location just before and just after the junction for each lane.

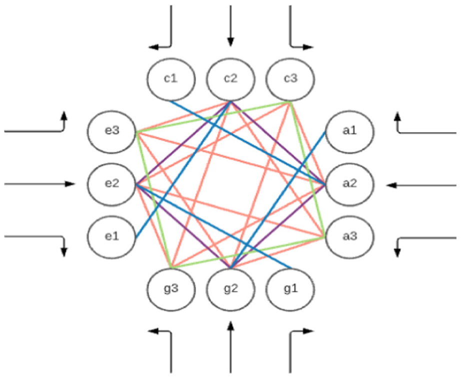


Fig. 4. Conflicts graph transformation

In Fig. 4, we present a complete conflicts graph to the junction as was described in Fig. 3. This graph describes all the optional crashing situations in the junction, while the nodes named a, c, e, g describe all the entries options of the junction according to the arrows. And, every node index x_1, x_2, x_3 describes the options of turning right, straight, or left respectively. Note that edges that do not exist indicate no conflict, for example there is no edge $(c1, g1)$ because there is no conflict between them.

5.2 Implementation of the Algorithm

Algorithm 1 builds the conflict graph according to the type of the junction and the list of the closest vehicles to the junction. In the first line of the algorithm (line 1), we build a complete conflict graph according to the type of the junction. In the second part of the algorithm (lines 2–8), we update the values of the vertexes according to the list of the closest vehicles to the junction and return this current conflicts graph.

Algorithm 1: buildConflictGraph(*vehicleList*, *type*):

```
   Result: Build a conflict graph
1  $G = db$.getConflictGraph($type$);
2  for $v$ in $vehicleList$ do
3      $lane = v$.getLane();
4      $vertex = G$.getVertex($lane$);
5      $vertex$.addVehicle($v$);
6  end
7  $G$.deleteAllTheEmptyVertexes();
8  return $G$;
```

Algorithm 2 schedules a junction according to the specific conflict graph. Every time unit, the algorithm decides which vehicles cross the junction and when. In the algorithm's first part (lines 3–4), built a conflict graph as described above according to the list of the closest vehicles to the junction that called *batch*. In the second part of the algorithm (lines 5–10), the arrival time will be scheduled according to the conflict graph, the parameter *clearIndicator* that describes when the junction is empty, and the *layer* is the largest set of non-conflict vehicles that can cross the junction safety. In the middle of the second part of the algorithm (lines 7–8), updates the parameter *clearIndicator* when α is the time of crossing the road and $changeVelocity(layer)$ return the time that takes all the vehicles to pass the distance D.

Algorithm 2: scheduleJunction(*junction*, *type*):

```
   Result: schedule a junction according to the specific conflict graph
1  $clearIndicator$=currentTime; \\ when the junction is empty
2  for $time$ in $timeUnit$ do
3      $batch$ =collect all the closest vehicles;
4      $G$=buildConflictGraph($batch, type$);
5      while $G \neq \emptyset$ do
6          $layer$=getNonConflictItems($G$);
7          $\alpha = Junction$.getMaxLenght()/$junction$.getMaxVelocity();
8          $clearIndicator = clearIndicator$+changeVelocity($layer$)+$\alpha$;
9          $G = G \setminus \{layer\}$;
10     end
11 end
```

Algorithm 3 calculates the new velocity for all the vehicles that are coming up to the junction and do not have conflicts. In the first line of the algorithm (line 1) we calculate the latest arrival time among all vehicles in the *layer*. The next lines (lines 2–4) update the parameter *maxInd* to keep the value of the safe arrival time i.e., when the junction is empty from vehicles. In the last lines (lines 5–9) the algorithm updates the velocity of all the vehicles in the layer according to the safe arrival time and return this value to update the next layer.

The algorithm above will work also in case that the vehicle currently in the junction is part of a *platoon*. There are several ways to preserve the platoon. One way is to increase the possible distance between two consecutive vehicles in a platoon. Another way is to decrease the velocity of the vehicles when they are in the proximity of a junction so that the next vehicle will still arrive at the allowed distance (a combination of both options is also possible). The algorithm details are omitted from this version.

Algorithm 3: changeVelocity($layer, indicator$):

```
Result: Calculate the new velocity
1 $maxInd$=max{($item$.getDistance()/$item$.maxVelocity())+time | $item \in layer$};
2 if $indicator > maxInd$ then
3     $maxInd = indicator$;
4 end
5 for $item$ in $layer$ do
6     $velocity = item$.getDistance()/($maxInd - time$);
7     $item$.changeVelocity($velocity$);
8 end
9 return $maxInd$;
```

5.3 Analysis of Our Algorithm

Assuming that there are vehicles in every entrance lane to the junction.
Objects Definitions:

l is the maximum number of entering lanes from some direction.
$G = (V, E)$ is the conflicts graph.
$|V| = 3l$ from a lane there are 3 options of turning at most.
$|E| = l^2$ all the options of conflicts.

The time complexity of scheduling the batch per time-unit:

- $build\ the\ conflicts\ graph.\ O(l^2)$
  - $get\ all\ the\ non-conflict\ vehicles\ in\ the\ graph.\ O(l)$
  - $update\ the\ conflict\ graph.\ \Theta(l)$
  - $update\ the\ velocity\ for\ the\ non-conflict\ vehicles.\ O(l)$

Total time complexity is: $O(l^2)$

5.4 Experimental Results

Figure 5 shows the execution result during a 1000 units time. The red graph describes the maximum number of vehicles in the layers per unit time, and the green graph describes the waiting time in seconds between two consecutive batches per unit time. Note that the waiting time also includes the execution time of the algorithm (the simulation was carried over an Intel® Core^{TM} i5-8500 CPU @3.00 GHz PC, with 8.0 GB memory).

From the figure above, we see that the correlation between the number of vehicles and the waiting time between two consecutive batches is tendentious; when the maximum number of vehicles in the layers of a batch is going down, the waiting time of the next batch is high and vice versa. However, the highest waiting time is 30 ms. Note that when the map is bigger the calculation time becomes also bigger. Therefore, running the algorithm on a world map requires a computer with a stronger and faster computing power than the computer mentioned above.

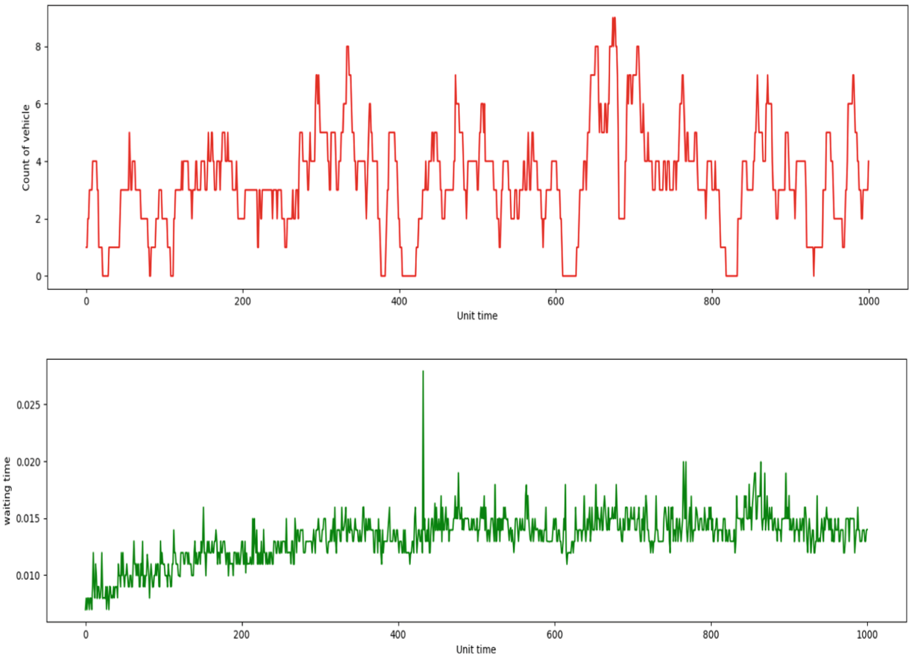


Fig. 5. Measurement of count of vehicles and waiting time (Color figure online)

6 Conclusions

In conclusion, this research discusses a platoon identification algorithm that is based on a road map, a representative vehicle, and a GPS location. Additionally this algorithm can identify platoons in any kind of road, straight or curved. This research presented real-time scheduling algorithms that are based on a conflict graph and an indication parameter for emptying the junction to the next vehicle for autonomous vehicles in different parts of a road that includes different type of junctions, and an extension of the concept of the algorithm for scheduling autonomous vehicles in all the parts of a road together. This work can be extended to the problem of scheduling a starting time driving for a vehicle in a path P (a path is defined as a series of junctions) from source to destination without collisions or changing velocity that is unnecessary during the driving.

References

1. Demonstration video (2022). https://drive.google.com/file/d/1-R2n7-aqiLFBNtfJe3FH4NDB4uMZuz31/view?usp=sharing
2. Bazzi, A., Zanella, A., Masini, B.M., Pasolini, G.: A distributed algorithm for virtual traffic lights with IEEE 802.11p. In: European Conference on Networks and Communications, EuCNC 2014, Bologna, Italy, 23–26 June 2014. pp. 1–5. IEEE (2014)
3. Conceição, H., Ferreira, M., Steenkiste, P.: Virtual traffic lights in partial deployment scenarios. In: 2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast City, Australia, 23–26 June 2013, pp. 988–993. IEEE (2013)
4. Dolev, S., Gudes, E., Yair, H.: Automatic real time platoon formation using the road graph. In: 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA), November 2021, pp. 1–4. IEEE Computer Society, Los Alamitos, CA, USA (2021)
5. Dolev, S., Kranakis, E., Krizanc, D.: Baked-potato routing. J. Algorithms **30**(2), 379–399 (1999)
6. Li, N., Chen, S., Zhu, J., Sun, D.J.: A platoon-based adaptive signal control method with connected vehicle technology. Comput. Intell. Neurosci. **2020**, 2764576:1–2764576:10 (2020)
7. Lopez, P.A., et al.: Microscopic traffic simulation using sumo. In: The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE (2018)
8. Ng, K.M., Reaz, M.B.I.: Platoon interactions and real-world traffic simulation and validation based on the LWR-IM. PLoS ONE **11**(1), 1–17 (2016)
9. Olaverri-Monreal, C., Gomes, P., Silvéria, M.K., Ferreira, M.: In-vehicle virtual traffic lights: a graphical user interface. In: 7th Iberian Conference on Information Systems and Technologies, CISTI 2012, pp. 1–6 (2012)
10. Vial, J.J.B., Devanny, W.E., Eppstein, D., Goodrich, M.T.: Scheduling autonomous vehicle platoons through an unregulated intersection. In: Goerigk, M., Werneck, R. (eds.) 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2016. OpenAccess Series in Informatics (OASIcs), vol. 54, pp. 5:1–5:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016)

11. Viriyasitavat, W., Roldan, J.M., Tonguz, O.K.: Accelerating the adoption of virtual traffic lights through policy decisions. In: International Conference on Connected Vehicles and Expo, ICCVE 2012, Las Vegas, NV, USA, pp. 443–444, 2–6 December 2013. IEEE (2013)
12. Zhao, W., Ngoduy, D., Shepherd, S., Liu, R., Papageorgiou, M.: A platoon based cooperative eco-driving model for mixed automated and human-driven vehicles at a signalised intersection. Transp. Res. Part C Emerg. Technol. **95**, 802–821 (2018)

Predicting Subscriber Usage: Analyzing Multidimensional Time-Series Using Convolutional Neural Networks

Benjamin Azaria[1] and Lee-Ad Gottlieb[2](✉)

[1] Cloudinary, Santa Clara, USA
benji@cloudinary.com
[2] Ariel University, Ariel, Israel
leead@ariel.ac.il

Abstract. Companies operating under the subscription model typically invest significant resources attempting to predict customers' future usage. These predictions can be used to fuel growth: Companies can use them to target individual customers – for example to convert non-paying consumers to begin paying for enhanced services – or to identify customers not maximizing their subscription product. This can allow the company to avoid an increase in the churn rate, and to increase the usage of some customers.

In this work, we develop a deep learning model to predict the product usage of a given consumer, based on historical usage. We adapt a Convolutional Neural Network with auxiliary input to time-series data, and demonstrate that this enhanced model effectively predicts future change in usage.

Keywords: Multidimensional time-series · Convolutional neural networks · Usage prediction

1 Introduction

Historically, companies interested in usage prediction have retained the services of expert analysts. These experts may examine past behavior over multiple features, estimate future usage in the short-term, and translate this knowledge into actionable tasks. Yet for companies which boast thousands of customers and have an increased need for accurate research forecasting, this non-scalable approach quickly becomes infeasible. Hence, the development of an automated model is an obvious necessity [7].

Future usage prediction is especially crucial for subscriber-based companies, for which churn rate has especially significant impact on profitability. It is crucial as well for companies valuated based on Annual Recurring Revenue (ARR) which reflects income derived from customers. For these companies, future usage prediction is a prerequisite for predicting the future valuation of the company.

Usage data sets are often represented as multi-dimensional time-series data, and indeed we have at our disposal two large multi-dimensional time-series, which constitute high quality labeled data sets, each produced in a controlled environment. However, successfully predicting multi-dimensional time-series values is itself a non-trivial task [3]. Indeed, automated multi-dimensional time series prediction is a relatively new field, and although different strategies have been proposed in the literature, currently there seems to be no consensus approach to the problem. Here, machine learning and deep learning algorithms are typically used for time-series data *classification*, which produces a binary decision value (for example, whether the customer usage will grow or not). The problem of *regression*, which produces a continuous value describing customer growth the extent of customer growth, is clearly much more challenging to accurately predict.

Our Contribution. For the regression problem of multi-dimensional time-series prediction, we propose to utilize both common machine learning techniques and novel methods. Specifically, we shall leverage one-dimensional convolutional neural network (CNN), in conjunction with auxiliary output, to predict future utilization based on historical usage. In this model, an auxiliary layer produces a prediction for the future in the short-term. This prediction is fed into the next auxiliary layer, which produces a prediction for a future prediction for a slightly later period, until the final auxiliary layer produces a prediction for the desired longer-term period. We demonstrate experimentally that this enhanced model effectively predicts future usage, and gives superior results when compared to other commonly utilized models in the field.

We present the data sets in Sect. 3.1, describe our learning model in detail in Sect. 4, and demonstrate promising empirical results in Sect. 5. In Sect. 6, we give concluding remarks and discuss future work.

Related Work. See [4] for a comparison of different time series regression models, such as gradient boosting and random forest. [10] showed that many cutting-edge regression methods fail to distinguish between periods of high and low quality data, and further are not able to generalise well to other data sets. In [12], Support Vector Machine (SVM) were compared with Artificial Neural Networks (ANN) on several distinct one-dimensional time-series for the task of regression, and they indicated that SVM was more efficient than ANN on these.

In [15] it was shown that classification using CNN on multi-channel human activity recognition time-series outperformed all other algorithms tested. [5] suggested eliminating the feature extractor, and feeding the network with raw inputs, while also relying on back-propagation. [9] combinated RNN and CNN for classification problems such as activity recognition, and [13] considered classification for Human Action Recognition, and evaluated multiple machine learning models including LSTM and CNN.

For using CNN to address multivariate time-series regression and comparison with other approaches, see [1,8] and [6]. A newer approach for solving multi-dimensional time-series is Temporal Convolutional Network (TCN): This was used in [14] to predict Earth's inter-annual climate variability, and [11] compared TCN to seven different models, finding that it achieved more accurate results on most data sets tested.

2 Problem Statement

We are given a set of multi-dimensional time-series, each with a continuous label-vector representing usage. Our goal is to predict the label of a query time-series, that is its future increase or decrease in usage. There is an imbalance in this model, in that the maximum possible decrease is bounded by 100%, while the maximum possible increase is unbounded. To address this, let P_A be the average usage in the observed time-frame, and P_B the average usage in the time-frame to be predicted. Setting $\delta = \frac{P_B}{P_A} - 1$, we define our usage increase function as:

$$f(\delta) = \begin{cases} 1 - \frac{1}{1+\delta}, & \text{if } \delta > 0 \\ \delta, & \text{otherwise} \end{cases} \tag{1}$$

The above function maps the increase or decrease onto a real value in the range $[-1, 1]$. Values tending to -1 imply significant decrease in usage, values close to 0 imply no change in usage, and values close to 1 imply extreme growth. (See also its use in Sect. 3.1 below.)

3 Data and Processing

Before presenting the model, we describe our data sets and their selected features, and detail our data processing. The data processing includes the creation of a modified label-vector for the time-series, which will be necessary for running the model.

3.1 Data Sets and Features

Our main data set is a proprietary set provided by Cloudinary, which is a software-as-a-service (SAAS) company providing media end-to-end solutions in the cloud. The data contains daily records of the media transformations undertaken by each customer, as well as delivery over the network, recorded as delivery requests and utilized bandwidth. Another important feature is the total data storage used by each customer through uploading media to Cloudinary's distributed shards; this includes original assets and assets derived via transformations. The data set also records each customer's subscription plan – these can be standard or custom, and monthly or annual. Note that a customer in this data set is often a company consisting of many distinct users.

We also utilized the public Bike-Share Usage in London and Taipei Network data set[1], which is a spatio-temporal urban transport data set from a network of bike stations. London and Taipei are both very large cities with large bike-share systems. At each location where bikes are picked up, the individual bikes are tracked, so that each rental generates a digital footprint: which bike, from where, to where, for how long, at what date and time, and by whom. The data collects events from 2016 until the first half of 2020.

[1] https://www.kaggle.com/ajohrn/bikeshare-usage-in-london-and-taipei-network.

For both data sets, care was taken to determine which features should be extracted. We avoided sparse features, as well as pairs of correlated features, or features unrelated to usage. A detailed list of features extracted from the Cloudinary data set is deferred to the full version. From the Bike Sharing data set we extracted the following fields: Start date, bike ID, duration, and startStation ID. We have also filtered out customers who are outliers, meaning their usage of the product is atypically small or large.

3.2 Data Processing

For each data set we computed the time-series associated with each individual user, and further spawned new user time-series at every one-week interval. While this could cause individual customers to appear as multiple time-series in the data set, we ensured that the same customer can appear (possibly multiple times) in only one of the train, validation or test groups.

Having segmented the data into individual time-series, we normalize each one: We first log-scale the time-series features, and then normalize each feature twice:

1. Self-comparison: Each customer feature x_j is compared to its time-series, mapped to $z_j = (\frac{x_j - \mu}{\sigma})$, where μ is the mean and σ is the variance of this feature over this time-series.
2. Global comparison: Each customer feature z_j is compared to all other customers, mapped to $w_j = (\frac{z_j - \mu}{\sigma})$, where μ is the mean and σ is the variance of this feature compare to all other customers.

Labels. It remains to calculate the usage label, as previously discussed in Sect. 2. We preprocess the Cloudinary data set as follows: Recall that P_A is the average usage in the observation period, which we now define to be the first ten weeks of the series, that is days 1–70. For the i-th time-series, we create a label vector y_i as follows: For entry $y_{i,j}$, we define $P_{B,j}$ to be the average usage in the 30 day period ending at day $70+j$. We then compute $\delta_j = \frac{P_{B,j}}{P_A} - 1$ and set $y_{i,j} = f(\delta_j)$ (as defined in Sect. 2), and obtain the distribution of our label. See Figs. 1 and 2 for an illustration of the computation of the label for $j = 60$ and the distribution of these labels in the Cloudinary data set.

The Bike-share data set is preprocessed in the same way, except that we take $P_{B,j}$ to be the average usage in the 21 day period (instead of a 30 day period) ending at day $70 + j$. Our choice for this period length reflects the fact that the behavior captured in the Bike-share data set is strongly influenced by the weekly cycle, as typically usage is determined by the week day. While the Cloudinary data set is also influenced by a weekly cycle, the standard approach in the corporate world is to compute usage in monthly blocks, making this the more appropriate measure for this data set.

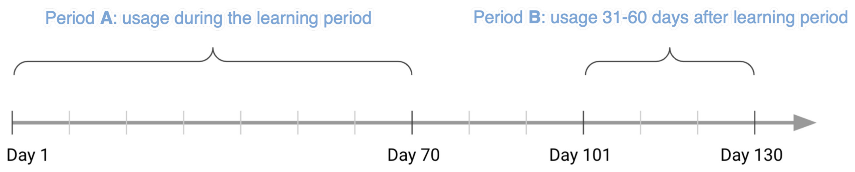

Period A: usage during the learning period
Period B: usage 31-60 days after learning period
Day 1
Day 70
Day 101
Day 130

Fig. 1. Label of data

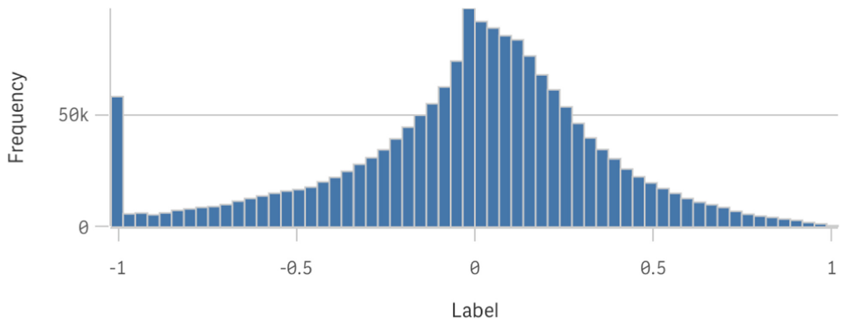


Fig. 2. Label frequency

4 CNN Construction

We model our data using Convolutional Neural Networks. These are known to return superior results for time-series classification when compared to other models [2,9,15], and when combined with auxiliary output are known to give predictions with 50% improved accuracy compared to state of the art models [16].

Our exact model depends on the period to be predicted. In what follows, we initially describe the model for the Cloudinary data set and predicting future usage of 60 days (which represents 2 months cycle of a customer), meaning predicting the average usage in the 30-day period including days 31–60 after the initial 70 days of the time-series. Hence we stipulated above (Sects. 2,3.2) that the time-series in the training set are labelled with a scalar in $[-1, 1]$ representing the average of this period.

Considering a single feature in a sequence (as in Fig. 3), the 1-dimensional convolution uses a kernel that weights adjacent observations, and by using pooling and a second convolution, enables the model to learn the trend of this specific feature. In our case we have multiple features, so we combine their output into a fully connected layer.

The structure of our model is as follows: It is composed of two convolutions (with 40 and 50 filters, respectively), followed by a fully connected (dense) layer of 150 neurons, using RELU as the activation. (see Fig. 4). Following this unit are an additional six fully connected layers of auxiliary output (see Fig. 5; the first auxiliary layer is also illustrated in Fig. 4). The goal of each auxiliary layer

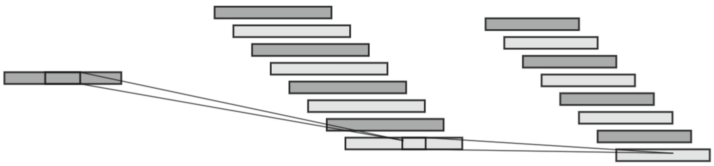


Fig. 3. One dimensional convolution on a single feature

is to predict future usage in the short term, and to pass this prediction to the next auxiliary layer. This next auxiliary layer will predict future usage for a slightly later period, until the final auxiliary layer yields a long-term prediction. In particular, we view each layer as predicting an additional ten days in advance, leading to a final prediction of sixty days. For each of the six auxiliary layers we utilize a dropout of 50% to avoid over-fitting.

We used grid search on the validation data set to determine the optimal number of convolutions, number of filters in each convolution, size of the fully connected layer, and the drop-out rate; the best results determined the parameters adopted in the above model. We also tried reducing the drop-out rate for later auxiliary level, but found that this did not yield appreciably better results. For computing the optimal number of auxiliary layers, we added layers until the improvement was negligible.

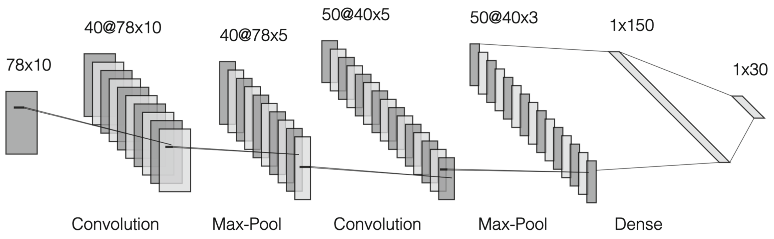


Fig. 4. The first part of the model includes two layers of 1-dimensional CNN followed by a fully connected layer. At the far right of the figure is the first auxiliary layer.

To each auxiliary layer $k \in \{1, 2, 3, 4, 5, 6\}$ we associate a mean-squared error (MSE) loss function (as in Fig. 5). Recall that the motivation behind auxiliary layer k was to produce a future prediction for $10k$ days ahead. Then

$$\text{loss}_k = \frac{1}{n} \sum_i (y_{i,10k} - \tanh(\bar{y}_{i,10k}))^2 \tag{2}$$

where $y_{i,10k}$ is the true label of future day $10k$ in time-series i (as defined in Sect. 3.2) , and $\bar{y}_{i,10k}$ denotes the k-th auxiliary layer's prediction for this value. The tanh function is used to restrict the range of $\bar{y}_{i,10k}$ to $[-1, 1]$, while having only a minor effect on values already within this range.

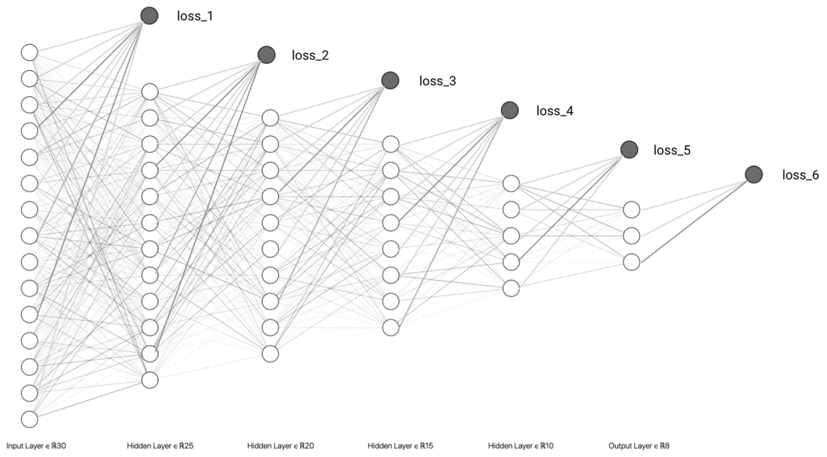


Fig. 5. The second part of the model includes 6 auxiliary layers.

The final model attempts to minimize a sum of the six individual loss functions. However, because each coordinate in label-vector y_i is determined by averaging time-series entries over the previous 30-day period, a simple sum would overvalue the time-series entries in the earlier part of the evaluation period, and undervalue the later time-series entries in that period. For example, the time-series value of day 31 after the observation period figures into all labels $y_{i,j}$ for $j \in [31, 60]$, while the time series value of day 60 only figures only into the value of $y_{i,60}$ in this range. To address this issue, we take a weighted sum instead, and the goal becomes to minimize

$$\text{joint-loss} = \frac{1}{6}\sum_{k=1}^{6}\left(\frac{\text{loss}_k}{\delta_k}\right) \tag{3}$$

for moderately decreasing values δ_k chosen via experimentation.

We stop the training after encountering 20 consecutive epochs without improvement in validation loss, as presented in the following algorithm:

```
$best\_validation\_loss\_score \leftarrow \infty$;
$remaining\_epochs \leftarrow 20$;
while $remaining\_epochs > 0$ do
    Train model one epoch and calculate $current\_validation\_loss\_score$ ;
    if $best\_validation\_loss\_score \geq current\_validation\_loss\_score$
     then
        $best\_validation\_loss\_score \leftarrow current\_validation\_loss\_score$;
        $remaining\_epochs \leftarrow 20$;
    else
        $remaining\_epochs \leftarrow remaining\_epochs - 1$;
    end
end
```

Algorithm 1: Stopping the training process

Extension to Other Prediction Periods. The above description gives our model for a 60-day prediction. In this model, each auxiliary layer corresponded to a forward prediction of 10 days. Hence, to adapt this model to different forward value predictions – specifically, 30 day and 90 day predictions – we simply modify the model to contain three or nine auxiliary layers, respectively, instead of the original six.

For the Bike-share data set, the experiment below attempts a forward prediction of nine weeks. Recall that for this data set we average periods of 21 days (instead of 30), and therefore we will view each auxiliary layer as giving a forward prediction of one week (instead of 10 days). It follows that for a forward prediction of nine weeks we can use a model of nine auxiliary layers.

5 Experiments

We run the following common machine learning models on our data sets: Random forest, Fully Connected Artificial Neural Network (ANN), Recurring Neural Network (RNN), and our augmented Conventional Neural Network (CNN) model. For both data sets, we divided the data as follows: 70% of the customers in the training set, 20% of the customers in the validation set, and 10% of the customers in the test set.

Cloudinary Data Set. Here we attempted to predict future usage of 30 days, 60 days and 90 days (meaning the respective averages of the three 30-day periods $[1, 30], [31, 60], [61, 90]$), and reported the success rates for each period in a designated table. The computed error values for each model is $\frac{1}{n}\sum_{i=0}^{n} \mathbf{1}_{|y_{i,j} - \overline{y}_{i,j}| > d}$ where $y_{i,j}$ is the true label (of Sect. 3.2), $\overline{y}_{i,j}$ is the output prediction of the model, $\mathbf{1}$ is the indicator function, and j takes values in $\{30, 60, 90\}$. The value d is a threshold on the difference between these two models; if the difference exceeds the threshold, the prediction is taken to be an error, and otherwise the prediction is viewed as successful. This is a common regression measure, and is especially appropriate for our setting due to commercial rationale behind the model: Companies seek to maximize the customers correctly served. We tried values $d \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ Comparisons for 30, 60, and 90 days are presented in Tables 1, 2 and 3.

The results show that our method almost always gives improved results when compared to the others. This is especially true as the forward prediction period is increased, making prediction more difficult.

Table 1. Performance of predicting behavior at 30 days in Cloudinary data

Value of d	Random Forest	ANN	RNN (LSTM)	Our model (CNN)
5%	21.62%	21.52%	26.66%	**28.33%**
10%	39.78%	43.22%	47.51%	**50.40%**
15%	55.56%	59.25%	63.41%	**66.40%**
20%	67.20%	70.25%	74.80%	**76.93%**
25%	76.29%	78.21%	82.17%	**84.09%**
30%	82.16%	84.37%	87.04%	**88.90%**

Table 2. Performance of predicting behavior at 60 days in Cloudinary data

Value of d	Random Forest	ANN	RNN (LSTM)	Our model (CNN)
5%	16.42%	18.01%	18.27%	**19.09%**
10%	32.73%	34.66%	35.52%	**36.17%**
15%	47.10%	48.45%	**50.96%**	50.88%
20%	59.77%	59.92%	63.29%	**63.73%**
25%	69.44%	70.77%	73.34%	**74.57%**
30%	77.92%	80.13%	80.84%	**81.71%**

Bike Rental Data Set. For this data set, we attempted to predict the total duration of usage in each station. As before the training period was 10 weeks (weeks 1–10), but now we attempt to predict usage at 9 weeks (week 19). As above, the computed error is taken to be $\frac{1}{n}\sum_{i=0}^{n}\mathbf{1}_{|y_{i,j}-\overline{y}_{i,j}|>d}$, but with $d \in \{0.01, 0.025, 0.05, 0.075, 0.1\}$. The lower values of d taken here in comparison to those taken for the Cloudinary data set are due to this data set being easier to predict – indeed, the practices of individuals as recorded in the bike-share data set are simpler and more predictable than those of the clients in the Cloudinary data set, which are usually large companies of many users. This is also the reason we sufficed here with only a single experiment, utilizing the most advances (nine layer) model. The results are reported in Table 4.

Table 3. Performance of predicting behavior at 90 days in Cloudinary data

Value of d	Random Forest	ANN	RNN (LSTM)	Our model (CNN)
5%	15.20%	15.68%	14.27%	**17.09%**
10%	28.37%	29.95%	28.27%	**31.56%**
15%	40.16%	42.31%	41.03%	**45.42%**
20%	50.42%	52.87%	51.40%	**55.90%**
25%	59.54%	61.52%	60.81%	**63.65%**
30%	67.16%	68.06%	68.40%	**70.28%**

Table 4. Performance of predicting behavior at 9 weeks in Bike-share data

Value of d	Random Forest	ANN	RNN (LSTM)	Our model (CNN)
1%	35.14%	45.68%	53.87%	**59.50%**
2.5%	74.79%	82.31%	88.22%	**92.20%**
5%	82.14%	87.41%	95.81%	**98.49%**
7.5%	95.03%	96.52%	98.61%	**99.18%**
10%	97.87%	98.06%	99.47%	**99.86%**

While all methods performed well for high values of d. Our method returned better results. For low values of d our method is appreciably better.

6 Conclusions

We attempted to predict future customer behavior using multi-dimensional time-series as our input. We utilized two data sets: The first was the Cloudinary data set, where customer are often large companies. Our second data set was a public data set containing behavior of individuals using the Bike-share in London and Taipei. We suggested a CNN-based model with auxiliary output, and compared this model to other standard approaches. Our approach achieved the best results in almost all of the comparisons undertaken. Our model performed better in the Bike-share data, and we believe this was because individual behavior is easier to predict than that of large companies.

We note that we also ran our model on less structured data, such as the Exchange rate per country data set[2]. Our model returned poor results for this set, and we believe this is because its behavior is erratic

References

1. Antsfeld, L., Chidlovskii, B., Borisov, D.: Magnetic sensor based indoor positioning by multi-channel deep regression. In: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 707–708 (2020)
2. Canizo, M., Triguero, I., Conde, A., Onieva, E.: Multi-head CNN-RNN for multi-time series anomaly detection: an industrial case study. Neurocomputing **363**, 246–260 (2019)
3. Cerqueira, V., Torgo, L., Mozetic, I.: Evaluating time series forecasting models: An empirical study on performance estimation methods. arXiv:1905.11744 (2019)
4. Goldsmith, J., Scheipl, F.: Estimator selection and combination in scalar-on-function regression. Comput. Stat. Data Anal. **70**, 362–372 (2014)
5. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. In: The Handbook of Brain Theory and Neural Networks, vol. 3361, no. 10 (1995)

[2] https://data.oecd.org/conversion/exchange-rates.htm.

6. Mehtab, S., Sen, J., Dasgupta, S.: Analysis and forecasting of financial time series using CNN and LSTM-based deep learning models. arXiv:2011.08011 (2020)
7. Miller, A., Vonwiller, B., Weed, P.: Grow fast or die slow: Focusing on customer success to drive growth (2021). https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/grow-fast-or-die-slow-focusing-on-customer-success-to-drive-growth. Accessed 5 Oct 2021
8. Mode, G.R., Hoque, K.A.: Adversarial examples in deep learning for multivariate time series regression. arXiv preprint arXiv:2009.11911 (2020)
9. Okita, T., Inoue, S.: Recognition of multiple overlapping activities using compositional CNN-LSTM model. In: UbiComp 2017, pp. 165–168 (2017)
10. Pimentel, M.A.F., Charlton, P.H., Clifton, D.A.: Probabilistic estimation of respiratory rate from wearable sensors. In: Mukhopadhyay, S.C. (ed.) Wearable Electronics Sensors. SSMI, vol. 15, pp. 241–262. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18191-2_10
11. Rasul, K., Sheikh, A.S., Schuster, I., Bergmann, U., Vollgraf, R.: Multivariate probabilistic time series forecasting via conditioned normalizing flows. arXiv:2002.06103 (2020)
12. Samsudin, R., Shabri, A., Saad, P.: A comparison of time series forecasting using support vector machine and artificial neural network model. J. Appl. Sci. **10**(11), 950–958 (2010)
13. Wang, J., Long, Q., Liu, K., Xie, Y., et al.: Human action recognition on cellphone using compositional Bidir-LSTM-CNN networks. In: CNCI 2019 (2019)
14. Yan, J., Mu, L., Wang, L., Ranjan, R., Zomaya, A.Y.: Temporal convolutional networks for the advance prediction of ENSO. Sci. Rep. **10**(1), 1–15 (2020)
15. Yang, J., Nguyen, M.N., San, P.P., Li, X.L., Krishnaswamy, S.: Deep convolutional neural networks on multichannel time series for human activity recognition. In: 24th International Joint Conference on Artificial Intelligence (2015)
16. Zhang, Y., Chang, F., Wang, M., Zhang, F., Han, C.: Auxiliary learning for crowd counting via count-net. Neurocomputing **273**, 190–198 (2018)

Smart Cybercrime Classification for Digital Forensics with Small Datasets

Isfaque Al Kaderi Tuhin, Peter Loh(✉), and Zhengkui Wang(✉)

Information and Communications Technology, Singapore Institute of Technology, Singapore, Singapore
16sic013j@sit.singaporetech.edu.sg,
{peter.loh,zhengkui.wang}@singaporetech.edu.sg

Abstract. In forensic investigations, automation support for cybercrime classification is crucial for efficient and effective resolution. Although numerous technologies have been developed to assist digital forensic investigators and incident responders in data analysis, investigation still requires a significant amount of manual effort, which is time-consuming and costly. Machine learning has been proven to be a promising strategy for automated data classification in recent studies. However, in order to train the model, most, if not all, machine learning algorithms rely substantially on a large number of annotated datasets. In this paper, based on a few historical case records, we propose an effective and efficient approach to classifying cybercrime, starting with small datasets. Our proposed method uses a Siamese Network Architecture with two identical Convolutional Neural subnetworks, as well as a Deep Learning Model. To classify novel graph data, a similarity metric forecasting method is applied, which permits the categorization of new data from a limited number of cybercrime records. In addition, to improve the accuracy of our Deep Learning Model, we apply forensic knowledge graph technology. This technique assembles pieces of information gathered from security logs into a vast, connected graph to expose a case's contextual background. Even with a small number of labeled samples trained in the Deep Learning model, our proposed approach is capable of identifying cybercrime data reliably and automatically from our experimental assessments. As a result, our approach demonstrates that automated cybercrime investigations can be both successful and feasible.

Keywords: Digital forensics and cybercrime classification · Knowledge graph · Small data · Deep learning · Neural network

1 Introduction

Around the world, cybercrime is increasing more rapidly than it is being resolved [1]. Cybercriminals are using more sophisticated techniques than ever to carry out malicious activities in the era of big data [2], causing unprecedented damage. Various digital forensics models [3] and technologies have been used, such

This project was supported in part by the NRF National Cybersecurity R&D Grant.

as the use of knowledge graph analysis to represent attack scenarios using graph database tools [4]. However, as S. Papastergiou et al. [5] expressed that most of them lack the ability to provide a comprehensive picture of a complex attack; they only view it from a high-level, abstract perspective. Furthermore, the investigation is manually carried out by the domain expert to come to a conclusion which is time consuming, error prone, and costly [6]. This contributes to the presence of cybercrimes undetected and unhindered in continuous attacks, leading to wasted time and revenue loss.

In order to automate the investigation process, several AI technologies have been adopted. A survey by S. Iqbal and S. A. Alharbi [6], compared various Machine Learning and Deep Learning approaches for data-driven digital forensic investigation and its significance. Even with AI approaches, an issue that is prevalent is the lack of forensics data as a reference to validate such approaches [7] as AI automation requires heap loads of historical data to train [8]. A study conducted by the Google Research Center and Carnegie Mellon University [9] measured prediction errors as they gradually trained with an increasing number of data in a deep neural network. They found that the accuracy increases as the volume of training data is expended. In other words, the more data, the better the AI performance.

Availability of the dataset, especially in the digital forensics community, remains scarce. C. Grajeda et al. [10] outlined the difficulty faced by the digital forensics community to find an openly available dataset in forensics research. Only 36.7% of the real-world data are estimated to be openly available. A large number of datasets are either private, outdated, or synthetic. In fact, an online article by Andrew Ng [11] states that, often in industry, the available data sets are much smaller than a large consumer Internet company such as Google.

Law enforcers in the field of digital forensics should be able to use AI technology and examine large amounts of data in a timely manner in search of important evidence during the incidence of cyber attacks. However, due to the lack of a readily available real-world data set of actual cyber attacks, it hinders the opportunity to fully utilize AI technology to conclude and classify cybercrime cases.

In this regard, we explore an approach that employs the Siamese Network to train a Deep Learning model and generate accurate predictions using a small number of contextual graph data, also known as a knowledge graph. Knowledge graphs are widely used in digital forensics, which are a transformation of logs, to derive various analyses using graph theory [12], making it the best data structure for a contextual classification such as our proposed method. It has also inspired us to use similarity learning, which examines the similarity metric between two pairs of knowledge graphs. In order to create such an architecture, the Siamese Network architecture was used to create our very own model, the Siamese Convolutional Neural Network (SCNN), which is defined by two identical Convolutional Neural Networks (CNN) that produce feature vectors simultaneously to compare with each other. As a result, it develops a useful approach for classifying new data simply by comparing it with small samples of existing data.

The major contributions of the paper are summarized as follows. First, we present the concept of using a knowledge graph with a Deep Learning model to create an effective classification model. Second, we propose a method that combines the robustness of knowledge graph data and the deep learning approach to classify a cybercrime case with a small number of trained datasets. Third, we evaluated our proposed approach against existing classification methods and the evaluation results indicate that our proposed method significantly outperformed the existing baseline method.

The rest of the paper is organized as follows. We provide the related work in Sect. 2, followed by a detailed description of the framework of the proposed methodology in Sect. 3. Section 4 outlines the performance and precision of the proposed method, and lastly, Sect. 5 provides the conclusion.

2 Related Work

In the field of digital forensics, numerous methods for analyzing and evaluating cybercrime have been offered in the past. The approaches, on the other hand, are constrained in a variety of ways, which will be discussed in this section.

In a cloud network context, A. A. Sharma and S. Sharma developed a hybrid approach for attack detection [13]. This method can detect attacks such as ICMP assaults, TCP Sync Attacks, and UDP Attacks in simulated network traffic. In general, this method relies on large amounts of data and works only with digital evidence from the cloud network environment.

Mohammad [14] presented a study that aims to improve the analysis of incriminating digital evidence by inspecting historical actions that affect the status of the file system with the objective of recognizing whether a malevolent application program has accessed, updated, or deleted any file system using the enhanced multiclass support vector machine model (EMSVM). This model uses a training dataset of about 126,831 system event logs that was simulated. Again, it shows that a large amount of data is required. Furthermore, mainly structured data were used to normalize and embed the data in ML.

N. Kanimozhi et al. [15] proposed a crime pattern analysis to predict and classify crime types based on various factors, such as the location, weather condition, period, etc. This was modeled using the Nave Bayes classifier, which provided great accuracy in predicting the type of crime and the occurrences of physical crimes. Similarly, digital forensics can adopt such techniques to predict any type of crime as and when it happens.

Silva et al. [16] used Support Vector machines (SVM) to automate the IT incident management process. It is an approach to building data around incoming incidents for automatic incident categorization. This enables fast incident recognition without the involvement of a human specialist. This is the kind of technology that digital forensics can adopt to automate the investigation process.

Rupa et al. [17] analyzed various classifiers, such as LinearSVC, logistic regression, multinomial Nave Bayes and random forest, and developed a computational system that could classify cybercrime offenses. For this method, cybercrime analysis reports are used to classify it into different cybercrime. This does not include the raw security log data that is crucial in detecting patterns of interest in a cybercrime case.

There were common techniques that are commonly used when there are no labeled data available are unsupervised clustering. K-means [18] and DBscan [19]. K-means is widely used by security analysis to help in gaining important insights about the known and unknown attack patterns as mentioned by A. Shrestha Chitrakar and S. Petrović [20]. However, it is not helpful, as it still requires human intervention to understand the pattern of interest and is prone to less accurate results [21].

Based on previous studies, it is clear that there is no single solution that allows flexibility in the type of data to be classified, the size of the training dataset required, the requirement of structured data, the use of no known contextual graph data, and the majority of unsupervised classification still requires a significant amount of manual human input.

As a result, we must design a strategy that takes advantage of AI models that are not data-hungry in order to improve the efficiency and automation of cybercrime detection and classification. Our proposed solution attempted to address the problem by allowing the categorization of different types of datasets, needing a smaller training sample for accurate classification, leveraging unstructured data, and, most crucially, automating the entire process.

3 Proposed Methodology

Our proposed framework consists of a three-step process as seen in Fig. 1 below. The prerequisite of this framework is the preparation of the dataset by transforming artifacts into a knowledge graph representation [22] which can be done using SIEM tools like Linkurious [23]. Secondly, graph embedding features are extracted [24] and then followed by training and validating a deep learning model using Siamese Convolutional Neural Network(SCNN). The following section contains the detailed description of each stage of the proposed framework.

3.1 Converting Case Records into a Knowledge Graph

Knowledge graphs have been widely used to represent the tracking and analyzing of a particular case. One of such existing ideas is CyGraph [25], which uses a unified graph-based cybersecurity model to capture the complex relationship that allows greater views for decision support and situational awareness. Even graphs are used in event analysis of cybercrimes [26].

There growing implementation of knowledge graph construction features in commercially available SIEM such as IBM's QRadar and Linkurious which are able to ingest large amount of security logs to form a graph network of corelated

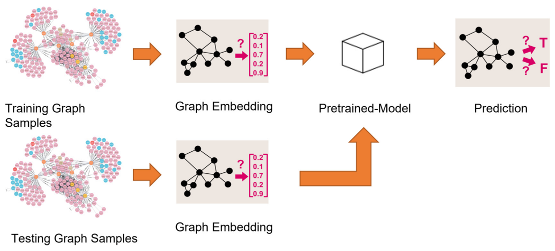


Fig. 1. Overall process flow of the proposed framework.

activities. This helps the Digital Forensic Investigator (DFI) or Post-Incidence Responder (PIR) gain an overview of the security event and take the necessary steps to analyze it using graph theory [27]. Due to such existing technologies being used commonly, it inspired us to utilize such knowledge graph technology to represent an entire case.

Table 1. Statistics of the benchmark dataset.

Dataset	No. of graphs	No. of classes	No. of vertices	No. of edges
PROTEINS	1113	2	39.1	72.8
IMDB BINERY	2000	2	429.6	497.8
REDDIT BINARY	1000	2	19.8	96.5

For this research, we picked benchmark graph datasets, which are REDDIT, IMDB, and PROTEIN from Morris TUDataset [28]. The reason of using the unrelated graph dataset is due to the lack of readily available knowledge graph datasets generated from real world cybercrime cases. Therefore, the benchmark data are used to hypothetically demonstrate the core functionality and result of the proposed approach, since different types of cybercrime would produce different knowledge graph structures [29]. In Table 1 the statistics for the reference data set used. Essentially, each instance of the dataset is a network graph that defines the topological properties for every instances (see Fig. 2).

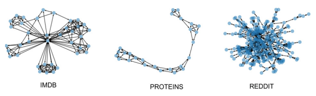


Fig. 2. Benchmark dataset Graph structures.

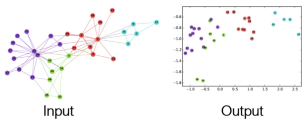


Fig. 3. Visualization of the graph embeddings.

3.2 Graph Embedding

To prepare the knowledge graph to be used in any AI model, it requires one to undergo a transformation of its nodes, edges, and their features into a small-dimensional vector space while preserving properties such as information and graph structure (see Fig. 3). Graph embeddings are the main essence that tries to capture the entire data representation within the graph so that it is in a "computationally digestible" format [30].

There are several embedding algorithms that can provide embeddings at the node, edge, and whole graph level. Since in our approach we want to classify multiple cybercriminal knowledge graphs, whole-graph embedding was used. A whole graph level embedding algorithm used for this approach is graph2vec [31] for its best performance in representing the whole graph quickly and accurately, it follows Gensim's Doc2Vec approach that uses the skip-gram network. It receives an ID of the document on the input and is trained to maximize the probability of predicting random words from the document.

3.3 Siamese Convolutional Neural Network Model

CNNs are a class of deep, feed-forward artificial neural networks where connections do not form a cycle. CNNs are generally used in computer vision, but they have shown promising results applied to other fields as well. Because it involves tasks where feature detection in text is more important, it is able to detect patterns in the context of textual data. W. Yin et al. [32] stated that the advantages in using CNN is also due to its fast performance and low memory usage capability.

Siamese Network by G. Koch et al. [33] proposed unique neural network structure that employs twin networks(hence Siamese) to make use of its powerful discriminative features to generalize the prediction for new data and data of a novel class.

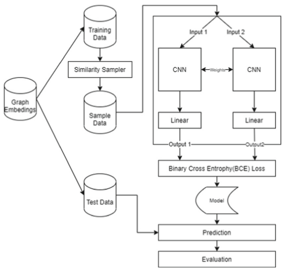


Fig. 4. SCNN architecture.

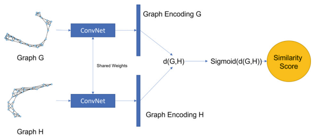


Fig. 5. Depiction of the similarity learning in SCNN.

In order to provide the best of both world, CNN for detecting better contextual pattern and Siamese to handle deep learning for small data, both of these techniques were combined to formulate a Convolutional Simese Neural Network. The model used here is a slight variation from the paper by F. López [34] and G. Koch et al. [33]. Figure 4 shows the architecture of SCNN.

To begin, the permutation of pairs of graph samples is calculated using the combination formula below, where n is the total number of graph samples, r is the number of combinations and Y indicates whether the pairs of samples belong to the same class as shown in (1).

$$ {}_nC_r = \frac{n!}{r!(n-r)!} \quad \Rightarrow \mathbf{Y} \tag{1} $$

The SCNN model is then used to train the above. During the training phase, the pairs of samples move through a sequence of convolutional networks simultaneously to generate an encoding for the graph. The encoding is then passed into a similarity encoding function. The similarity encoding function is the Euclidean pairwise distance (d (G, H)), which is calculated using the pairwise distance between two points of the graph encoding of the first pair, G, and the second pair, H, in a 2D vector space of the graph as shown in (2). After that, a round of the sigmoid activation function is performed on the pairwise distance D to output a binary classification where 0 is dissimilar and 1 is similar. This is then iterated a certain number of times until the prediction improves to match the similarity label of the permutated combination of the training pairs.

$$ d\left(G,H\right) = \sqrt{\sum_{i=1}^{n} \left(H_i - G_i\right)^2 \Rightarrow \mathbf{D}} \tag{2} $$

To derive the similarity score (see Fig. 5, a new sample from the benchmark dataset P with class y is compared with the trained sample G with class Y using as a similarity metric after the model has been trained. If SCNN (G, P) has a high threshold T score, P is labeled as class G. If the score is less than T, it will check with another trained sample G, and the process will repeat until the results are acceptable.

4 Experiments and Results

In this section, we begin by experimenting with hyperparameters to determine the best hyperparameters for similarity learning across a variety of graph data to determine the optimum model configuration. As noted in the previous section, the benchmark graph datasets, REDDIT, IMDB, and PROTEIN from Morris TUDataset [28] were used. The model performance is then compared using an accuracy metric to that of various baseline classification techniques, such as Kmeans, Multinomial Naive Bayes, and K-Nearest Neighbors, which are all from the Scikit Learn Library [35], while the GCN+CNN is from a paper by F.López [34]. The baseline graphs were chosen because they closely resemble our model's goal, and since, according to W. A. Al-Khater et al. [36] In comparative research, there is no standard approach to the classification of a few shot graphs.

Firstly, each dataset was initially randomized, and then a balance train set for each class (total of 4 samples) and a test set of four samples for each class(total of 8 samples) were chosen at random. The graph embeddings for each train and test sample were then extracted using the graph2vec embedding technique with dimension size d. We used the SCNN architecture to create a hidden dimension h with an output size of 2 due to the hot encoding nature of the prediction. Adam optimizer was used with its default hyperparameter and a common learning rate of *lr=1e-3*. The training was iterated for 100 rounds. Subsequently, the model is evaluated using the test sample using an F1 score, which is the distributed accuracy of precision and recall. Three different sizes of d and h were carried out to test which dimension size is capable of capturing the context of the knowledge graph for better accuracy. In Table 2 shows the score based on the different hyperparameters. Given the result in Fig. 6, it is apparent that the most optimal hyperparameters are d and h are 16 and 8, respectively.

Table 2. Comparison results the proposed SCNN model and other common classification algorithms.

Dataset	d=8,h=4	d=16,h=8	d=32,h=16	d=64,h=32
PROTEINS	0.649	0.792	0.899	0.801
IMDB BINERY	0.753	1.0	0.899	0.899
REDDIT BINARY	0.584	0.898	0.792	0.697

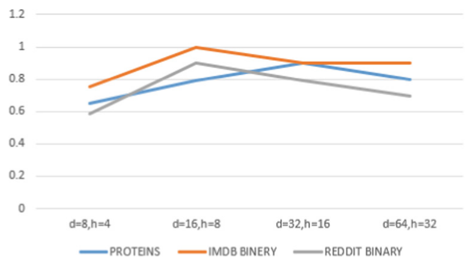


Fig. 6. A comparison graph of the score for different hyperparameters selected

We then compared the effectiveness of our technique with six different methods, which included both supervised and unsupervised procedures, using the equivalent hyperparameters. The evaluation metric used for this result is the F1 score metric, since we are concerned about the distributed accuracy. The results are shown in Table 3. We can obtain these by utilizing the confusion matrix we got from our model, and we can get the average F1 score for the results among the three different datasets. The result indicates that our model correctly classifies the data with an average F1 score of 0.89, demonstrating that the SCNN outperforms the other classification models. The PROTEIN and IMDB datasets

Table 3. Comparison results the proposed SCNN model and other common classification algorithms.

Method	PROTEIN	IMDB	REDDIT	Average
GRAPH+CNN+SIM(SCNN)	1.0	1.0	0.67	0.89
GRAPH+CNN	0.899	0.792	0.375	0.689
Multinomial NB	0.899	0.899	0.697	0.832
Logistic Regression	0.899	0.899	0.451	0.75
K-Means	0.293	0.293	0.375	0.32
K-Neighbours Classifier	0.697	0.697	0.333	0.576

receives similar F1 Score among the other classification models, except for the largely linked graph data like the REDDIT Dataset. However, this is not the case for SCNN, as it was able to accurately classify the REDDIT dataset with an F1 score of 0.67.

Other models, such as Multinomial Nayev Bayes, Kmeans, and K-Nearest Neighbors, do not perform well for a wide range of graph topologies, which could be due to a lack of appropriate datasets. It indicates that, with only a few training samples, our model is capable not only of performing well, but also of tailoring to a wide diversity of graph data formats.

5 Conclusion

In this paper, we proposed a solution in this research for dealing with the shortage of datasets in the digital forensic domain. The framework we established is crucial for the development of a technique that can aid in the classification of cybercrime in a Digital Forensics Investigation. This uses a similarity learning approach to automatically understand attack patterns, which uses deep learning technology to forecast accurately without the need for a significant amount of data. Our proposed approach enables forensics professionals to predict the type of cybercrime with which they are dealing quickly and efficiently to investigate cybercrime offenses. The results show that the established approach can save time and eliminate the need for manual reporting. It aids in early detection of a crime, even with limited data. This is an important mechanism that investigators can use to forecast cases and begin taking action. This could also potentially shorten the learning curve for new cybercrime detectives.

This paper has led to many interesting future works. For example, it is interesting to extend the approach by including real-world cyber-forensic data that can be converted into a knowledge graph. Expert investigators can also use the human-in-the-loop approach to correct incorrectly predicted classifications. This will be a fantastic contribution to the cyberforensic world as the number of data coming in increases. Finally, to forecast unseen crimetypes, an approach such as detection of novel categorisation can be used.

Acknowledgement. The research work described here is funded, in part, by Singapore's National Research Foundation(NRF) National Cybersecurity R&D Grant GC2018-NCR-0008.

References

1. Montasari, R.: CDFIPM. University of Derby, United Kingdom (2021)
2. Nouh, M., Nurse, J.R., Webb, H., Goldsmith, M.: Cybercrime investigators are users too! (2019). arXiv preprint, arXiv:1902.06961
3. Nadeem Ali, M.: Crime detection using digital forensic technology. Int. J. Comput. Sci. Inf. Secur. **14**, 01 (2016)
4. Henseler, H., Hyde, J.: Technology assisted analysis of timeline and connections in digital forensic investigations. In: LegalAIIA@ ICAIL, pp. 32–37 (2019)
5. Papastergiou, S., Mouratidis, H., Kalogeraki, E.-M.: Handling of advanced persistent threats and complex incidents in healthcare, transportation and energy ICT infrastructures. Evol. Syst. **12**(1), 91–108 (2021)
6. Iqbal, S., Alharbi, S.A.: Advancing automation in digital forensic investigations using machine learning forensics. Digit. Forensic Sci. 3 (2020)
7. Hughes, N., Karabiyik, U.: Towards reliable digital forensics investigations through measurement science. Wiley Interdisc. Rev. Forensic Sci. **2**(4), e1367 (2020)
8. Polachowska, K.: 12 challenges of AI adoption, January 2022
9. Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 843–852 (2017)
10. Grajeda, C., Breitinger, F., Baggili, I.: Availability of datasets for digital forensics-and what is missing. Digit. Investig. **22**, S94–S105 (2017)
11. Andrew, N.: Ai doesn't have to be too complicated or expensive for your business, August 2021
12. Studiawan, H., Sohel, F., Payne, C.: A survey on forensic investigation of operating system logs. Digit. Investig. **29**, 1–20 (2019)
13. Sharma, A., Sharma, S.: An intelligent analysis of web crime data using data mining. Int. J. Eng. Innovative Technol. (IJEIT) **2**(3) (2012)
14. Mohammad, R.M.A.: An enhanced multiclass support vector machine model and its application to classifying file systems affected by a digital crime. J. King Saud Univ. Comput. Inf. Sci. **34**(2), 179–190 (2019)
15. Kanimozhi, N., Keerthana, N., Pavithra, G., Ranjitha, G., Yuvarani, S.: Crime type and occurrence prediction using machine learning algorithm. In: 2021 International Conference on Artificial Intelligence and Smart Systems, ICAIS, pp. 266–273. IEEE (2021)
16. Silva, S., Pereira, R., Ribeiro, R.: Machine learning in incident categorization automation. In: 2018 13th Iberian Conference on Information Systems and Technologies, CISTI, pp. 1–6. IEEE (2018)
17. Ch, R., Gadekallu, T.R., Abidi, M.H., Al-Ahmari, A.: Computational system to classify cyber crime offenses using machine learning. Sustainability **12**(10), 4087 (2020)
18. Jin, X., Han, J.: K-Means Clustering, pp. 695–697. Springer, US (2017)
19. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, vol. 96, pp. 226–231 (1996)

20. Shrestha Chitrakar, A., Petrović, S.: Efficient k-means using triangle inequality on spark for cyber security analytics. In: Proceedings of the ACM International Workshop on Security and Privacy Analytics, pp. 37–45 (2019)
21. Joy, A.: Pros and cons of unsupervised learning, August 2021
22. Ehrlinger, L., Wöß, W.: Towards a definition of knowledge graphs. SEMANTiCS (Posters, Demos, SuCCESS) **48**(1–4), 2 (2016)
23. Linkurious, Graph visualization: why it matters (2022). https://linkurious.com/blog/why-graph-visualization-matters/, (accessed 12 January 2022)
24. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: a survey of approaches and applications. IEEE Trans. Knowl. Data Eng. **29**(12), 2724–2743 (2017)
25. Noel, S., Harley, E., Tam, K.H., Limiero, M., Share, M.: Cygraph. In: Handbook of Statistics, vol. 35, pp. 117–167. Elsevier (2016)
26. Adderley, N.A.: Graph-based temporal analysis in digital forensics. Theses and Dissertations, 2241 (2019)
27. Pinheiro, M.: Graph theory for cybercrime: a note. Mod. Int. J. Pure Appli. Math. **1**(2), 30–37 (2017)
28. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tudataset (2020). arXiv preprint, arXiv:2007.08663
29. Eberle, W., Holder, L., Graves, J.: Using a graph-based approach for discovering cybercrime. In: FLAIRS, January 2010
30. Tong, F.: Graph embedding for deep learning (2021). https://towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings-4305c10ad4a4, (accessed 12 January 2022)
31. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: Graph2vec (2017). arXiv preprint, arXiv:1707.05005
32. Yin, W., Kann, K., Yu, M., Schütze, H.: Comparative study of CNN and RNN for natural language processing (2017). arXiv preprint, arXiv:1702.01923
33. Koch, G., Zemel, R., Salakhutdinov, R., et al.: Siamese neural networks for one-shot image recognition. In: ICML Deep Learning Workshop, Lille, vol. 2 (2015)
34. López, F.: Text classification with CNNS in Pytorch, December 2021
35. Unknown, "scikit-learn" (2022)
36. Al-Khater, W.A., Al-Maadeed, S., Ahmed, A.A., Sadiq, A.S., Khan, M.K.: Comprehensive review of cybercrime detection techniques. IEEE Access **8**, 137293–137311 (2020)

Auditable, Available and Resilient Private Computation on the Blockchain via MPC

Christopher Cordi[1], Michael P. Frank[1], Kasimir Gabert[1], Carollan Helinski[1], Ryan C. Kao[1], Vladimir Kolesnikov[2](✉), Abrahim Ladha[2], and Nicholas Pattengale[1]

[1] Sandia National Laboratories, Albuquerque, NM, USA
[2] Georgia Institute of Technology, Atlanta, GA, USA
kolesnikov@gatech.edu

Abstract. Simple but mission-critical internet-based applications that require extremely high reliability, availability, and verifiability (e.g., auditability) could benefit from running on robust public programmable blockchain platforms such as Ethereum. Unfortunately, program code running on such blockchains is normally publicly viewable, rendering these platforms unsuitable for applications requiring strict privacy of application code, data, and results.

In this work, we investigate using MPC techniques to protect the privacy of a blockchain computation. While our main goal is to hide both the data and the computed function itself, we also consider the standard MPC setting where the function is public.

We describe GABLE (Garbled Autonomous Bots Leveraging Ethereum), a blockchain MPC architecture and system. The GABLE architecture specifies the roles and capabilities of the players. GABLE includes two approaches for implementing MPC over blockchain: Garbled Circuits (GC), evaluating universal circuits, and Garbled Finite State Automata (GFSA).

We **formally model and prove** the security of GABLE implemented over garbling schemes, a popular abstraction of GC and GFSA from (Bellare et al., CCS 2012).

We analyze in detail the performance (including Ethereum gas costs) of both approaches and discuss the trade-offs. We implement a simple prototype of GABLE and report on the implementation issues and experience.

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for NNSA under contract DE-NA0003525. This report describes objective technical results and analysis. Any subjective views or opinions that might be expressed in this report do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Approved for public release SAND2022-3532 C.

1 Introduction

Current programmable blockchain platforms such as Ethereum provide a "global computer," an always-on public computing resource. They guarantee reliability, availability and auditability for computations implemented as smart contracts, which are posted to the blockchain and subsequently executed. Even in the event of a widespread disaster, any still-functioning part of the Ethereum network renders this computing resource *available*. Organizations can take advantage of such a robustly available computing facility to execute particularly mission-critical computational tasks, if this computation can be done *privately*.

We note that the performance of secure multiparty computation (MPC) has been steadily improving and is practical today even for large functions/inputs.

In this work, we explore the motivation, use cases, architectures, and constructions for securing (i.e. ensuring privacy of) a sensitive computation done on sensitive inputs on a public blockchain network using MPC techniques.

We present GABLE (Garbled Autonomous Bots Leveraging Ethereum), an architecture and a system for running MPC on the Ethereum blockchain. We consider two different base approaches, garbled finite state machines/automata (GFSA), and garbled circuits (GC); we present both in a unified manner as garbling schemes. We show how functions can be computed privately. At a high level, in our architecture there are four types of players, or participant roles:

1. *Garbler* is a player who generates encrypted function and input encodings, publishes the former on Ethereum, and distributes the latter to other players. Garbler may be a single trusted entity or run distributedly, e.g., by an MPC over a private chain.
2. *Input Provider* or *Writer* is a player who contributes (encrypted) inputs to the computation.
3. *Input Unlocker* or just *Unlocker* is a designated player that manages encrypted inputs, preventing input providers from learning unauthorized information about the computation and adapting their input based on it.
4. *Output Recipient* or *Reader* is a player who may obtain a designated output from the computation.

We emphasize that players may play several of these logical roles simultaneously (with some exceptions; cf. trust model Sect. 2.3). For example, an input provider may also be an output recipient.

1.1 Motivation

Existing public programmable smart-contract blockchains such as Ethereum provide a highly robust and accessible computing platform. An entity whose operations may require execution of business or strategic logic that needs to function (using inputs from various sources) with a high degree of assurance of its reliability and availability may implement that functionality as a smart contract running on a blockchain. Another desirable blockchain property is auditability, due to the generally immutable nature of data committed to a consensus chain.

However, the deploying entity may wish to keep private the following computation data and metadata:

- Input values provided to the computation, including their semantics.
- The nature of the computation that is being performed on the inputs. Most generally, we may wish for all information about the structure and function of the computation to be obscured.
- Any internal data within the computation.
- Semantics of any intermediate or final outputs produced by the computation.

Most of these features are implied by standard MPC properties, while hiding the computed function is the goal of Private Function Evaluation (PFE) [25].

Use Cases. We outline two use-case scenarios which illustrate the potential usefulness and practicality of our system. Of course, many others are possible.

Sealed-Bid Auctions: Auditability and Security. Consider a simple sealed-bid auction with simultaneous bidding, where the soundness of the computation may be fully audited after the fact, if needed. This may be useful, e.g., in public-interest auctions, such as auctioning airwaves to cellular providers, or electricity delivery contracts to utilities. Other cases (e.g., auctioning of a privately-held company to a select set of bidders) may demand strict privacy.

While such auctions may be easily run via an MPC, running them on a blockchain offers a much higher level of transparency and trust (and user engagement) than more traditional means of execution. We report some analysis of costs for our methods on simple multi-bidder auctions in Sect. 5.

Transactive Energy. Refers to market-based mechanisms for managing the exchange of energy in a wide-area electrical grid [32]. Participants in such a market, e.g., utility companies, may wish to engage in automatic electronic negotiation without revealing their strategies. A capability such as GABLE could provide a solution. Each participant deploys their own garbled bot expressing their private negotiation strategy; these bots then negotiate with each other on the public blockchain, while hiding their internal negotiation strategies.

Private-Public Chain Integration. The encrypted function may be generated (garbled) by an MPC executed over a private network. This assures trust in the garbling, facilitates future opening of the secure computation (auditing) if needed (cf. Sect. 2.3), and integrates public and private chains.

1.2 Contribution

Blockchain and MPC technologies provide essentially complementary features to a computation. A blockchain assures availability (ability to provide input, compute, and retrieve the output when needed) and reliability (including assurance that only authorized players can submit input and that the output of the computation is correct), even if direct communication channels among the parties are cut. MPC ensures privacy and correctness, guaranteeing partial reliability (output correctness) but nothing about availability.

Our system, GABLE (Garbled Autonomous Bots Leveraging Ethereum) combines the best of the two technologies. In our security model, we require not only the data, but, optionally, also the computed function to be private (standard MPC evaluates publicly known functions). In this work, we:

– Design a general architecture for available and resilient MPC over the Ethereum blockchain. It allows authorized players to submit inputs at any time, and to retrieve outputs as soon as they are available. Our design specifies the roles and capabilities of the players.
– Design two approaches for implementing MPC on the blockchain: Garbled Circuits (GC), including GC evaluating universal circuits, and Garbled Finite State Automata (GFSA).
– *Formally state and prove* the security of our system against malicious players. Note, we *do not formally define* the properties of auditability, availability and resilience of the computation. They are derived from the corresponding intuitive properties of the underlying Ethereum blockchain. We specifically discuss achieving auditability in Sect. 2.3.
– Analyze in detail the performance characteristics (including gas costs) of both approaches and discuss the trade-offs.
– Implement a simple prototype and several demonstration applications, and report on the implementation issues and experience.[1]

1.3 Results and Evaluation

We prototyped GABLE using GFSA and experimented on an Ethereum testnet. See Sect. 5 for details of our experiments and results; here, we summarize our findings. First, a simple provenance-tracking application scenario was modeled using 5 cycles of reactive functionality in a simple 6-state FSA; the cost to run this demo on the Ethereum mainnet would have been less than US$3.00 on the day of the test. Next, we demonstrated a 5-state machine implementing MPC for the classic 2-party "Millionaire's Problem" for configurable input lengths based on a bit-serial comparison algorithm; the cost of that demo would have been about $0.50 per bit of input, and can be further reduced. Finally, we compared costs for GFSA versus a garbled universal circuit (GUC) implementation on a multi-bit auction problem, showing that the cost overhead of GUC grows only modestly (polylogarithmically) with the number of bidders, as expected, and becomes less expensive than GFSA for $B > 7$ bidders, at which point the cost is about $20 per bit of price data. Note, GC *without* functional privacy would cost far less.

[1] A reference implementation of our initial prototype has already been publicly released [21], and wider licensing/availability of a more complete code base is under consideration.

1.4 Outline of the Paper

We presented the motivation and several use cases above in Sect. 1.1, and previewed our contribution and results in Sect. 1.2 and Sect. 1.3. We present preliminaries in Sect. 1.5 and discuss related work in Sect. 1.6. We provide a high-level overview of our approach in Sect. 2, including defining the logical players, the trust model, and security intuition. A generic security statement and theorem are outlined in Sect. 3, and specialized for our main GC-based protocol in Sect. 4. We discuss our prototype and demo implementations and present results and cost analysis in Sect. 5 and conclude in Sect. 6.

The full version of the paper [18] also includes several appendices that present some technical details of the GFSA approach used in our early implementations, the proof of the main security theorem, and additional details of our prototype implementations and test results.

A much more detailed report on the system design, implementation and demos, including reference source code for the initial prototype, is available as a Sandia National Laboratories technical report [21].

1.5 Preliminaries

Blockchain Technology. Bitcoin [28] is a stunningly successful technology, which generalizes public timestamping (ledger) and smart contracts work and uses proof-of-work, concepts considered before [19,23,29]. Bitcoin has limited support for programming digital smart contracts, i.e. it is not Turing-complete. This is in part due to the possibility of intentional or accidental resource overuse or even exhaustion as a result of programming issues. The need for a rich programming language for contracts was nevertheless recognized, and in 2013, the Ethereum blockchain was proposed [14]. Ethereum addressed the issues stemming from language Turing-completeness by putting the onus on the programmer/contract creator, and requiring payment per storage unit and execution step. We implement our system for Ethereum; our higher-level design is general and will fit most natural blockchain architectures.

MPC (including Two-Party Computation, 2PC, and the general p-Party Computation), is an area of cryptography that studies protocols which compute functions over players' private inputs while revealing nothing except the function output. MPC has improved dramatically over the past 15 years. The first proof-of-concept 2PC implementation, Fairplay [27], evaluated only 200 Boolean gates per second. Today, 2PC implementations can process up to 2–5 million gates/s [36]. Improvements in the malicious model and in 3PC and MPC are even more impressive. Recent work reports 3PC techniques that can evaluate as many as *seven billion* Boolean gates/second [6]. Research on algorithms and implementations has firmly transitioned MPC from a theoretical curiosity to a subject of practical engineering.

We note that 2PC protocols, and specifically Yao's garbed circuit (GC) techniques [34], are most suitable for use in our setting, because the blockchain itself can naturally serve as the GC evaluator.

FSA (finite-state automata) comprise a standard but simple model of computation that differs from Boolean circuits. The FSA model is weaker (some functions require exponentially larger representation in FSA as compared to circuits). The primary benefit of GFSA, in the case of sufficiently low-complexity applications, is simply that obscuring the structure of the computation becomes relatively trivial, since all computations reduce to a linear sequence of state transition table lookups, and relatively simple techniques suffice to obscure the topology of the state graph. As a result, the overhead to achieve functional privacy in GFSA becomes less than that of garbled (universal) circuits for computations operating on sufficiently small numbers of bits.

Garbling Schemes and Garbled Functions (GF). We build our approach around GC and GFSA. They are special cases of garbling schemes, as defined by Bellare et al. [10]. Informally, we will refer to garbling schemes as *garbled functions* (GF), similarly to how "GC" refers to both the GC garbling scheme and the GC approach. We will use the terms "garbled" and "encrypted" function interchangeably in this work. The BHR framework defines a garbling scheme as a tuple of algorithms $\mathcal{G} = (\mathsf{ev}, \mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$[2] and requires that they satisfy the following properties:

In addition to the correctness property, BHR define relevant notions of security for garbled functions: privacy, obliviousness and authenticity. We refer the reader to [10] for precise definitions of these standard notions. Here, we informally summarize these notions.

GF *correctness* guarantees correct evaluation if all players behave honestly.

GF *privacy* guarantees that an adversary $\texttt{Adv}$ who sees the garbled function (e.g. the GC), the encoded inputs and the output decoding information, does not learn anything beyond the result of the computation.

GF *obliviousness* guarantees that an $\texttt{Adv}$ who sees the garbled function and the encoded inputs, does not learn anything. This notion is different from privacy, which gives $\texttt{Adv}$ the decoding information and allows it to obtain the output of the computation (and nothing else).

GF *authenticity* captures $\texttt{Adv}$'s inability to create from a GF F and its garbled input X a garbled output $Y \neq F(X)$ that will be deemed authentic.

Note, we only need correctness and privacy. The authenticity requirement can be avoided if we choose to rely on the blockchain to honestly evaluate GF. Obliviousness becomes unnecessary if the output decoding information d is always published (e.g., d is provided to output receivers, who may be corrupted by $\texttt{Adv}$), in which case we are never in the setting without d available to $\texttt{Adv}$.

MPC from GF. Bellare et al. [10] do not systematize the ways one can obtain an MPC protocol from GF. However, the following (informally presented) natural 2PC construction works, and is proven secure against semi-honest adversaries in [10], assuming GF is private:

[2] In the BHR notation, ev is a reference evaluator for plaintext functions, Gb is the Garbler, En is the input encoder, Ev is the garbled function evaluator, and De is the output decoder.

Construction 1 (2PC from GF, informal). `Gen` *generates GF F, encoding information e, and decoding information d by running* Gb*.* `Gen` *sends F, d to* `Eval`*.* `Gen` *and* `Eval` *securely (e.g. via Oblivious Transfer (OT)), deliver to* `Eval` *the labels of e corresponding to players' inputs.* `Eval` *evaluates GF by running* Ev*, and obtains the plaintext output from garbled output labels and d by running* De*.*

We note that the above construction is secure against malicious `Eval`, as long as label delivery remains secure against a malicious `Eval`. We will use this property in our security argument.

1.6 Related Work

We briefly discussed relevant MPC and FSA preliminaries in Sect. 1.5. In this section, we review several systems addressing privacy on the blockchain and compare them to our approach.

MPC+Blockchain. As we discuss next, many works explore the interplay of blockchain and MPC. To our knowledge, only YOSO (You Only Speak Once) MPC [11,22] formally models a *public* blockchain executing MPC. YOSO deviates from the typical blockchain architecture (e.g., of Ethereum) of all nodes sharing the same view. Instead, YOSO nodes have private data and are selected to perform MPC subtasks. If sufficiently large fraction of selected players are honest, MPC is secure. To protect against adversarial corruption, these players are hidden: they are unpredictably self-selected (e.g., via mining-like process), and each MPC subtask consists of computing and sending a *single* message, after which they erase their relevant private state. The main technical challenge of YOSO MPC is sending encrypted messages (e.g. containing internal state and subtask computation output) to *unidentified* players who are self-selected in the future. While YOSO MPC has attractive asymptotic complexity, unfortunately, it is concretely prohibitively expensive due to the cost of its building blocks. Our solution, at the cost of much stronger corruption and trust model (e.g., we only handle non-adaptive corruptions, while YOSO supports adaptive), is far more efficient and aligns with Ethereum architecture.

On the other hand, permissioned networks, such as Hyperledger, may be run by a small number of semi-trusted servers, and MPC can naturally be executed among the servers to achieve full privacy of transactions and contracts. This direction is explored in [12]. Their approach does not extend to public blockchains, since an arbitrary number of adversarial nodes may participate in the public network. Our work can be seen as general MPC on a public ledger for a restricted use case, where the encrypted function is generated by an organization trusted by the participants (and whose honesty can be later audited).

Hawk [26] is an architecture for a blockchain that can support private data. It handles private data using a trusted *manager*, realized using trusted hardware, such as Intel SGX. The trusted enclave *may* be implemented via MPC. It is not clear who would be the MPC principals to achieve a reasonable trust model; further (and [26] acknowledges this) this would cause an impractical overhead.

The Enigma system [37] uses MPC protocols to implement support for private data on a blockchain. They use MPC off-chain to perform computation on shares of data. We aim to run MPC on-chain for resilience, availability and auditability; Enigma's techniques will not achieve these properties.

A line of work explores the interplay of blockchain and (separately executed) MPC to achieve fairness in MPC or connect MPC to financial mechanisms directly [5,13,17]. Works such as [33] use blockchain to manage encrypted inputs to MPC performed by a separate trusted network. Reference [16] considers a blockchain-hybrid MPC model (plain model with available ledger), and addresses foundational issues of MPC, such as concurrent composability, in this model. In contrast, in our work, the blockchain itself executes MPC.

Zero-knowledge proofs (ZKP) are widely used both in MPC and in blockchain. We note that public ledger nodes never prove anything (indeed, the underlying secret would then be known to everyone). Instead, ZKPs are used by off-chain entities, such as wallets, to prove correctness of their actions. Several ledgers, such as ZCash, provide transaction privacy based on ZKPs. This line of work is ortohogonal to the privacy protection work we consider.

Solidus [15] uses a publicly verifiable ORAM machine to generalize and scale up the ZKPs for the use case where financial institutions representing many accounts interact with a ledger.

Blockstream CA [31] use simple ZKPs in conjunction with additive homomorphic commitments to manipulate secret data on the ledger. Partial privacy can be achieved for very simple functionalities (for efficiency, we are constrained by additively-homomorphic encryption).

In contrast to the above approaches, our solution is general MPC.

Trusted Enclaves. As in the Hawk example above, privacy can be achieved if one is willing to entrust hardware enclaves, such as SGX. Nodes of the blockchain network may be equipped with enclaves, which would execute encrypted contracts on encrypted data. Several other systems, such as Secret Network [4], also implement this approach. We note that enclave security is a cat-and-mouse game; in this work, we do not rely on secure enclaves.

2 Overview: Approach and Trust Model

As discussed in Sect. 1, we wish to add privacy of both computations and data to the process of contract execution on the Ethereum network. Data and function privacy is normally achieved using an appropriate secure computation protocol. However, in the public blockchain setting, the number of network nodes is unspecified, and MPC privacy guarantees cannot be achieved. Instead, we take the following approach:

2.1 Logical Players and Evaluation Pattern

We consider several logical players:

– The *Contract creator* or *Garbler* sets up encryptions of functions and inputs. It initializes the contract and sends encrypted labels to corresponding input providers. Garbler can be run by an MPC, e.g. over a private chain.
– *Input provider* or *writer*. This player is authorized to interact with a published contract (which implements a GF) and provide (garbled) input into the contract based on the plaintext input it has.
– *(Input) unlocker*. This player facilitates secure input provision by establishing an extra decryption step (performed by the unlocker) of the submitted garbled input. This prevents input providers, who posses *both* input labels on each input wire, from decrypting the internals of the encrypted computation. Effectively, use of the unlocker (who we assume does not collude with input providers) implements a secure OT of the input label based on the input.
– *Evaluator*. This player (implemented by the blockchain itself) evaluates the GF by executing the contract created by the contract creator on garbled inputs provided by the input providers. (By its nature, the blockchain also generates an indelible public archive of the contract's execution, including garbled inputs and outputs.)
– *Output recipient* or *reader*. This player is authorized to receive the output of the computation. It is also possible to make the output available to all.

2.2 Approach

In our approach, the blockchain network itself plays the role of the *Evaluator* `Eval` of the GF (either a garbled circuit or a garbled FSA, in this work).

GF Generation and Contract Publishing. The computed function is first represented as a Boolean circuit or FSA. Then it is garbled within the BHR framework [10], resulting in a GF (e.g., GC or GFSA).

The GF is assumed to be *honestly* generated by an agent of a contract creator, `Gen`. We note that `Gen` possesses all secrets of the encrypted function and therefore is able to infer the internal state of the (plaintext) computation, should it ever gain access to the encrypted evaluation. Therefore we assume that all the secrets of the (small and self-contained) computation performed by `Gen` are *securely deleted*[3]. That is, we assume that `Gen` produces GF F, encoding information e and decoding information d. Upon delivery (as we discuss next) of e and d to the blockchain network players, and of F to the contract, `Gen` securely erases all its state (perhaps except F and d). We note that secure deletion of `Gen`'s state is not needed if audit may be desired or it is allowable for `Gen` to inspect the details of the evaluation, such as inputs, intermediate states, etc.

[3] If we require auditability of MPC, this information must be securely stored instead of being deleted. Then, upon audit, the generated GC can be reconstructed, and its correctness and correctness of MPC execution verified. See Sect. 2.3 for details.

Input Provision. As plaintext input becomes available to input providers, they may enter the corresponding garbled input labels into the contract. To do this, in the GC case, they must have access to *both* garbled labels for each Boolean input. This would present a serious security problem if not addressed. A player who knows more than one label of a wire may infer unallowed plaintext information. In addition to passively learning private information, the attacker may adaptively substitute its input, thereby affecting the correctness of the computation as well.

We address this issue by introducing and using *unlockers*, logical players who help manage input labels. Thus, the process of input provision proceeds as follows (we specify it for the case of GC; the GFSA case is analogous):

1. `Gen` generates GF and corresponding input labels, w_i^0 and w_i^1, representing two labels for each Boolean input wire W_i. `Gen` encrypts these labels with unlocker key k_u. For each input wire W_i, `Gen` gives the two encryptions $Enc_{k_u}(w_i^0), Enc_{k_u}(w_i^1)$ to the input provider responsible for the wire W_i. `Gen` gives the unlocker key k_u to the unlocker associated with W_i.
2. When the input provider is ready to submit the (encrypted) input $b \in \{0,1\}$ on wire W_i, it publishes to the contract $Enc_{k_u}(w_i^b)$, the encrypted label corresponding to its input b, received from `Gen`.
3. When notified (e.g., off-chain or by the blockchain, or in response to monitoring the blockchain), the unlocker retrieves $Enc_{k_u}(w_i^b)$ from the contract, decrypts it with the key k_u received from `Gen`, and publishes w_i^b to the contract.

Secure Evaluation and Output Delivery. Once all inputs are provided to the contract, the contract is evaluated by the blockchain and the (encrypted) output is produced. Anyone may inspect the encrypted output, and only authorized players (those who received d, or corresponding portions of d from `Gen`) may decrypt and obtain the plaintext output.

Reactive Functionalities. We stress that the computation need not be one-shot. It is natural to consider multi-staged evaluation, where intermediate outputs may be provided to output recipients, and function state propagated across the stages. This is easy to achieve with obvious variations of GF evaluation. One approach to this is illustrated in Fig. 1. We prove security only for one-shot functionalities. Proofs can be naturally extended to the reactive case.

2.3 Trust Model

After having described the players and their actions, we are now ready to specify the trust model. There are two main assumptions:

– We assume that the contract generator acts honestly and *securely erases* its state after completion of its task. Note, this is immediately achieved if garbler is implemented as MPC e.g., run on a private chain.
– Input providers *do not collude* with corresponding unlockers. That is, we allow arbitrary collusions of players, but a set of colluding parties may not include an input provider and an unlocker for the same wire/GFSA step.

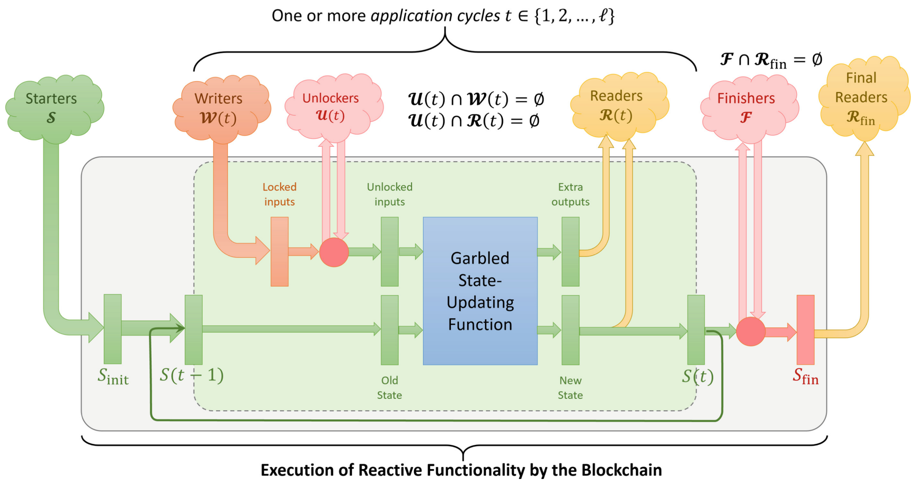

One or more *application cycles* $t \in \{1, 2, \dots, \ell\}$
Starters $\mathcal{S}$
Writers $\mathcal{W}(t)$
Unlockers $\mathcal{U}(t)$
$\mathcal{U}(t) \cap \mathcal{W}(t) = \emptyset$
$\mathcal{U}(t) \cap \mathcal{R}(t) = \emptyset$
Readers $\mathcal{R}(t)$
$\mathcal{F} \cap \mathcal{R}_{\text{fin}} = \emptyset$
Finishers $\mathcal{F}$
Final Readers $\mathcal{R}_{\text{fin}}$
Locked inputs
Unlocked inputs
Extra outputs
Garbled State-Updating Function
S_{init}
$S(t-1)$
Old State
New State
$S(t)$
S_{fin}
Execution of Reactive Functionality by the Blockchain

Fig. 1. ***Reactive execution of garbled machines by blockchain contracts.*** The gray region represents the full machine execution, which may require one or several contracts. The green sub-region represents operation within one *application cycle*, each of which may accept new inputs and produce new intermediate outputs. The blue rectangle represents a garbled state-update function that maps (old state, input) to (new state, output); this block can be implemented either using a (monolithic) garbled FSA transition table, or as a traditional GC (the latter requiring a *projective* or bit-vectorized state encoding). It must be garbled separately for each cycle. In this conception, $\mathcal{S}$ denotes a set of players called *Starters* authorized to configure the initial state, and each cycle t may have its own sets of authorized input providers $\mathcal{W}(t)$, unlockers $\mathcal{U}(t)$, and output recipients $\mathcal{R}(t)$. The Finishers $\mathcal{F}$ and Final Readers $\mathcal{R}_{\text{fin}}$ are only required if there is a final output that is supposed to be visible to a broad audience including the other players, but where the other players may have a disincentive to reveal the specific output value to that audience. (Color figure online)

Security Against Cheating `Gen`*: Audits, Covert* [8] *and PVC* [7,24] *Security.* We assume that `Gen` behaves honestly and, further, erases its state. In some scenarios, it may be desired to open the computation at a later stage, e.g., for audit purposes. This can increase trust in `Gen` and the transparency of the process. Of course, the auditor (or the public, if the computation is opened to the public) will learn the inputs of all players. Release of this information may be acceptable, e.g., in situations where inputs are sensitive only for a certain duration of time.

Auditing of `Gen` is easily achieved by requiring `Gen` to generate everything from a PRG seed and to securely store the seed. During audit, the seed is revealed and the auditor verifies that all actions of `Gen` are consistent with the seed (this may require participation of unlockers and input providers). Because GC is secure against a malicious evaluator, honest generation of GC implies correctness and security against malicious players in the collusion model described above.

In the case when the function is public, we can also easily achieve covert [8] or even publicly verifiable covert (PVC) [7,24] security. Following the ideas on [8], covert security can be achieved by requiring `Gen` to produce two GFs and sets of inputs; the blockchain network, e.g. via a randomness beacon, challenges to open one of them, verifies its correctness, and evaluates the unopened GF. PVC, a strengthening of the covert model introduced by [7], requires the ability to prove cheating, in case a cheater was caught. Because the GF and all inputs are published on the chain, it is easy to collect evidence of cheating. Firstly, we can require `Gen` to publish a seed (failure to do so will automatically imply guilt). Further, it is easy to verify that `Gen`'s actions are consistent with the seed and punish it (e.g. via funds slashing) if a violation is detected.

3 Generic Security Statement and Proof

We state the general security theorem for functions implemented as garbled functions and present the proof. The security of our specific construction presented next in Sect. 4 is an immediate corollary of this general theorem.

Let $\mathcal{G} = (\mathsf{ev}, \mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme, satisfying correctness and privacy as defined by [10] (as noted in Sect. 1.5, obliviousness and authenticity are not needed). We additionally require that the decoding information d is projective[4], and decoding each bit calls a hash function, modeled as a *Random Oracle* (RO). Note, standard GC constructions in fact do implement d this way: output wire's plaintext value, for example, can be obtained by computing $\text{low\_bit}[H(w_i)]$, where w_i is the output label[5]. Similarly, other garbling schemes, such as GFSA, can have a decoding function d incorporate a call to RO. We will use the RO programmability [20] in our simulation.

Theorem 1. *Let $\mathcal{G} = (\mathsf{ev}, \mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme as above. Let $(y_0, ..., y_p) = f(x_0, ..., x_q)$ be the function desired to be computed, such that each bit of the function output depends on all inputs*[6]*. Let* `Gen` *be the contract generator, $IP_1, ..., IP_n$ be the input providers, $U_1, ..., U_m$ be the unlockers, and $R_1, ..., R_\ell$ be the output receivers. Assume* `Gen` *is honest and generates $(F, e, d) = \mathcal{G}.\mathsf{Gb}$ and distributes (F, e, d) to players as described above. Let $I \subset \{IP_i, U_j, R_k\}$ be the set of*

[4] As defined in [10], in a projective garbling scheme, the encoding information is represented as a list of tokens, one denoting 0, and one denoting 1, for each bit of the input; an encoding of a player's input is a collection of the tokens corresponding to its plaintext input. Similarly, for the output decoding, we say it is projective if the plaintext output is decoded bitwise in a similar manner.

[5] To use $\text{low\_bit}[H(w_i)]$ as the decoding function, `Gen` needs to ensure that $\text{low\_bit}[H(w_i^b)] = b$. This is easy to do by choosing the output labels from corresponding domains. We stress that this is but one way of implementing d with these properties.

[6] While some functions of interest do not meet this requirement, the functions we consider in this work will: indeed, universal circuit and FSA function outputs depend on all their inputs.

colluding malicious players, such that for no input wire W_i both its input provider and unlocker are in I.

Then blockchain evaluation of f which computes $\mathcal{G}.\mathsf{ev}$ as described above, is secure against a malicous adversary corrupting I.

Proof. For lack of space, we present the proof in the full version of this paper [18].

Remark 1. If an unlocker U_j colludes with a reader R_k, together they can learn the output of the computation and abort based on it. This is not considered a vulnerability in the standard notion of simulation-based security. Note, if we wish to avoid such adaptive abort, we can require that no unlocker colludes with any reader.

4 Instantiations and Security Proofs

Construction 2 (UC GC-based)**.** *Our main construction is the instantiation of the generic GF-based construction described above in Sect. 2 based on the following choice of underlying primitives/schemes:*

Let f be a function to be computed on the blockchain. Let C be a Universal Circuit computing f. Let $\mathcal{G}$ be the classic Yao GC garbling scheme with point-and-permute [9] and projective decoding function as specified in assumptions of Theorem 1.

Having proven a generic security theorem (Theorem 1) for computing functions represented by arbitrary garbled functions, the proof of security of our main protocol, which is GC-based, is an immediate corollary of Theorem 1.

Theorem 2. *Assume all assumptuions of Theorem 1 hold, including the collusion assumptions. Then Construction 2 is secure in the malicious model against collusions specified in Theorem 1.*

Proof. Proof is an immediate corollary of Theorem 1 and the fact that the underlying GC scheme used in 2 satisfies the required assumptions of Theorem 1. □

Other Instantiations and Proofs are Analogous. In particular, GC-based instantiation is the same as UC GC with the exception of garbling the circuit C, and not necessarily a UC. A garbled FSA offers a reasonable performance for simple functions compared to UC GC. In an appendix of the full version [18] we cast a one-shot evaluation of GFSA as a GF in the [10] notation. A GFSA-based GF satisfying privacy and correctness can be used as a basis of our general construction.

5 Prototype Implementations and Test Results

To illustrate our approach and assess its real-world cost, we implemented a simple prototype and several demonstration applications. Specifically, an implementation of the GFSA approach (see full version [18]) was developed, for simplicity, and applied to several demo applications represented as finite state automata.

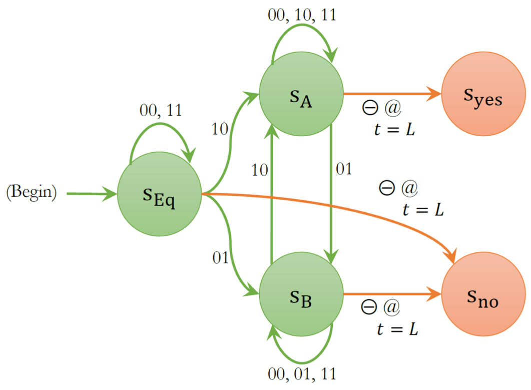


Fig. 2. ***Base FSA used in Millionaire's Problem demo.*** In this version, players AB supply bits of their input values simultaneously, least-significant bit first, on successive cycles. In the final cycle, after L time steps have passed, a Finisher (as in Fig. 1) supplies a special "Finish" symbol ⊖ which makes the final result readable by both parties.

The prototype Garbler, implemented in Python, takes a simple JSON-format description of an FSA transition function and translates it to a sequence of garbled tables, one for each state update cycle (time step). After garbling, another Python script translates the garbled machine data to source code in the Solidity programming language for a smart contract for the Ethereum platform; this contract includes the garbled tables as static data, together with a generic *Executor* which accepts garbled input values from input providers and evaluates the garbled machine, producing garbled outputs which can then be interpreted by authorized output recipients.

We used the popular Truffle tool suite, which provides a framework for Ethereum development, to develop, test and deploy (on a private test network, and later on the Ethereum mainnet) several prototypes and demonstrations, which we now discuss.

We implemented a provenance tracking demo, presented in detail in the full version [18]. Next we discuss our implementation of the millionaires problem.

5.1 Millionaires' Problem Demo

This demo executed a 5-state machine (Fig. 2) implementing MPC for Yao's classic 2-party "Millionaires' Problem" [35] with bit-serial inputs. Before we had added Unlockers, to achieve MPC fairness (the last player to move possesses an informational advantage due to his ability to look ahead at final outputs), we invoked a special extra player "Finisher" (separate from the 2 normal parties) that acts to reveal the result. (See also Fig. 1).

Extending this line of argument, we observe that every step (input provision and corresponding GFSA state update) of the GFSA execution may exhibit the following similar vulnerability: the input provider may see the immediate effect of its input, such as such as whether the next FSA state depends on its input. This issue can be resolved by state-graph transformations, which increase the size of the state machine. In one version that we tested, each additional bit of input length cost almost exactly 2 million gas units to store the additional garbled machine data (since each time step has to be garbled separately), which was about \$0.50 worth of Ethereum on the day of that test. With some overhead, the total cost to run an FSA for a 32-bit, two-party Millionaire's Problem was 75 million gas, corresponding to roughly US\$75 or so given typical prices at that time. That demo required spreading out the GFSA data over multiple smart contracts, due to Ethereum contract size limits.[7] Reimplementing this same demo using Unlockers for each input and an optimized 2-state FSA allowed us to reduce the gas cost to $\sim$\$12.

5.2 Configurable Garbled Universal Circuit (GUC) Method

To let us handle applications of a complexity beyond the reach of the GFSA approach in future implementations of GABLE, a simple approach was designed to implement a garbled circuit (GC) for (configurable) universal circuits (UCs). Figure 3 illustrates our basic UC approach. Input values here are activated using Unlockers (not shown), as we described earlier in the paper.

Although implementation of this method is still in progress, careful analysis of the approach allowed us to already compare its costs to those of the existing GFSA technique for an example problem, a multi-party auction (generalized from the Millionaire's Problem). Figure 4 shows comparison results. As we expected, cost scales up exponentially with the number of bidders B for GFSA, but only as $\Theta(B \log^2 B)$ for the GUC. (Circuit width scales as $\Theta(B)$, circuit depth scales as $\Theta(\log B)$, and the depth of the Thompson network for each application circuit layer also scales as $\Theta(\log B)$.) The break-even point with our implementation falls at $B = 7$ bidders, where the cost of both approaches is roughly \$20 per input bit.

[7] Per EIP-170 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md), a contract's deployed bytecode size cannot exceed 24,576 bytes.

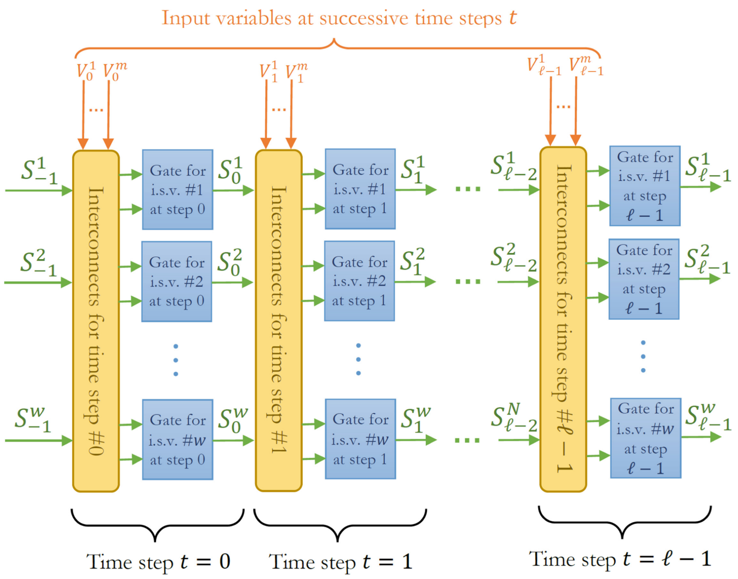

Input variables at successive time steps t
V_0^1 V_0^m
V_1^1 V_1^m
$V_{\ell-1}^1$ $V_{\ell-1}^m$
S_{-1}^1
S_{-1}^2
S_{-1}^w
Interconnects for time step #0
Interconnects for time step #1
Interconnects for time step #$\ell-1$
Gate for i.s.v. #1 at step 0
Gate for i.s.v. #2 at step 0
Gate for i.s.v. #w at step 0
Gate for i.s.v. #1 at step 1
Gate for i.s.v. #2 at step 1
Gate for i.s.v. #w at step 1
Gate for i.s.v. #1 at step $\ell-1$
Gate for i.s.v. #2 at step $\ell-1$
Gate for i.s.v. #w at step $\ell-1$
S_0^1 S_0^2 S_0^w
S_1^1 S_1^2 S_1^w
$S_{\ell-2}^1$ $S_{\ell-2}^2$ $S_{\ell-2}^N$
$S_{\ell-1}^1$ $S_{\ell-1}^2$ $S_{\ell-1}^w$
Time step $t = 0$
Time step $t = 1$
Time step $t = \ell - 1$

Fig. 3. ***Concept for configurable universal circuits in multi-step computations.*** In this approach, for each layer of application logic, a generalized connection network such as a Thompson network [30] obscures the interconnect topology. The elements of that network, together with generic application gates for computing new values of internal state variables (i.s.v.'s), are configured via truth tables during the garbling process. Thus, no separate programming input is required for this type of UC, yet the function of the network remains obscured.

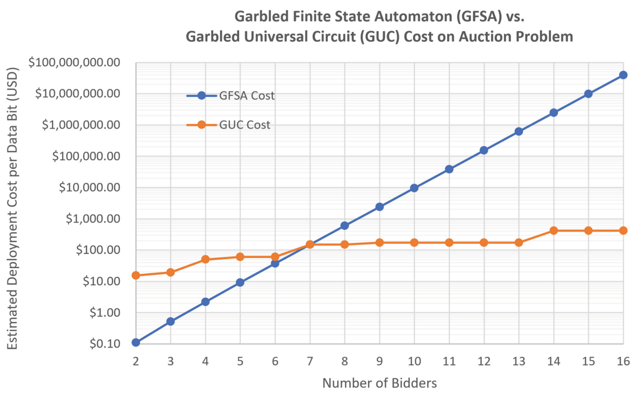


Fig. 4. ***Cost comparison for GFSA vs. configurable universal GCs for multi-party auctions.*** The break-even point occurs for $B = 7$ bidders, where the cost of both techniques is about 800 million gas per bit of input length. Prices here assumed optimistically that we are paying only 1 mETH (or in the ballpark of \$0.20) per million gas; however, after our tests, the average gas price grew substantially higher.

6 Conclusion

In this paper, we described a novel approach to performing secure computation (including functional privacy) on a blockchain. The general approach has two basic embodiments that we discuss, based on the garbling of finite-state automata (FSA) and Boolean circuits, respectively. We gave an overview of the basic structure of the approach, including its participant roles and high-level procedures, outlined a proof of its basic security properties, and discussed early implementations and test results.

We found that simple FSA-based applications can be executed privately at moderate dollar costs on the Ethereum blockchain. For more complex applications, a simple approach based on a construction we call configurable Garbled Universal Circuits (GUC) achieves complete functional privacy with costs that scale as $\Theta(wd \log w)$ in the width w and depth d of the application circuit, with reasonable constant factors. We carried out a detailed cost comparison for a multi-bidder auction application, for which GUC outperforms garbled FSA for $B > 7$ bidders, and remains arguably feasible to perform on-chain with full functional privacy for up to hundreds or even thousands of bidders.

We deployed and executed two GABLE demos on the Ethereum mainnet in late July and mid-September of 2020.[8] The purpose of these tests was to 1) ensure that there were no unforeseen difficulties with real-world deployment, and 2) validate our cost estimation methodology. Both purposes were realized, with no surprises. The first deployment [2,3] (July) was a very simple four-state machine, similar to the supply-chain example discussed in the full version [18]. The second deployment ([1], Sep.) was for a GFSA implementation of a two-party auction as in Fig. 4.

References

1. Auction contract. Ethereum address `0x98ccd7e190ac28a36d4f065a4f14dc5e0b67f5c7`
2. Simple executor contract. Ethereum address `0xc8a54a72f187ec444ed08968901284bbd6d2ec06`
3. Simple storage contract. Ethereum address `0x57f1c190982d0a9ecdf7c4703e134d9eaf347de0`
4. The Secret Network. https://scrt.network/. Accessed 25 Jun 2020
5. 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)
6. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 805–817. ACM Press (October 2016)

[8] Source code for these contracts is at https://github.com/sandialabs/GABLE.

7. Asharov, G., Orlandi, C.: Calling out cheaters: covert security with public verifiability. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 681–698. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_41
8. Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_8
9. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press (May 1990)
10. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACM Press (October 2012)
11. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 260–290. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_10
12. Benhamouda, F., Halevi, S., Halevi, T.: Supporting private data on hyperledger fabric with secure multiparty computation. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), pp. 357–363 (2018)
13. 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
14. Buterin, V.: Ethereum Whitepaper (2013). http://ethereum.org/en/whitepaper
15. Cecchetti, E., Zhang, F., Ji, Y., Kosba, A.E., Juels, A., Shi, E.: Solidus: confidential distributed ledger transactions via PVORM. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 701–717. ACM Press (October 2017)
16. Choudhuri, A.R., Goyal, V., Jain, A.: Founding secure computation on blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 351–380. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_13
17. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 719–728. ACM Press (October 2017)
18. Cordi, C., et al.: Auditable, available and resilient private computation on the blockchain via MPC. Cryptology ePrint Archive, Report 2022/398 (2022). https://eprint.iacr.org/2022/398
19. 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
20. Fischlin, M., Lehmann, A., Ristenpart, T., Shrimpton, T., Stam, M., Tessaro, S.: Random Oracles with(out) programmability. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 303–320. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_18
21. Frank, M.P., et al.: The GABLE report: garbled autonomous bots leveraging Ethereum. Technical report SAND2020-5413, Sandia National Laboratories (2020). https://www.osti.gov/biblio/1763537
22. Gentry, C., et al.: YOSO: you only speak once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
23. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. J. Cryptol. **3**(2), 99–111 (1991). https://doi.org/10.1007/BF00196791

24. Kolesnikov, V., Malozemoff, A.J.: Public verifiability in the covert model (almost) for free. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 210–235. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_9
25. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_7
26. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy, pp. 839–858. IEEE Computer Society Press (May 2016)
27. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: Blaze, M. (ed.) USENIX Security 2004, pp. 287–302. USENIX Association (August 2004)
28. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf. Accessed 25 Jun 2020
29. Szabo, N.: Secure property titles with owner authority (1998). https://nakamotoinstitute.org/secure-property-titles/. Accessed 25 Jun 2020
30. Thompson, C.: Generalized connection networks for parallel processor intercommunication. Technical report, Carnegie-Mellon University, Pittsburgh, PA (1977)
31. van Wirdum, A.: "Confidential assets" brings privacy to all blockchain assets: Blockstream. Bitcoin Magazine (April 2017). Accessed 25 Jun 2020
32. Wikipedia. Transactive Energy. https://en.wikipedia.org/wiki/Transactive_energy
33. Yang, A., Wei, L., Wu, J., Long, C.: Block-SMPC: a blockchain-based secure multiparty computation for privacy-protected data sharing. In: Proceedings of the 2020 the 2nd International Conference on Blockchain Technology, ICBCT 2020, pp. 46–51. Association for Computing Machinery, New York (2020)
34. Yao, A.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE (1986)
35. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science, SFCS 1982, Chicago, IL, USA, pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38
36. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8
37. Zyskind, G., Nathan, O., Pentland, A.: Decentralizing privacy: using blockchain to protect personal data. In: IEEE Symposium on Security and Privacy Workshops, pp. 180–184 (2015)

Union Buster: A Cross-Container Covert-Channel Exploiting Union Mounting

Novak Boskov[1](✉), Naor Radami[2](✉), Trishita Tiwari[3](✉), and Ari Trachtenberg[1,4](✉)

[1] Boston University, Boston, MA, USA
{boskov,trachten}@bu.edu
[2] Ben-Gurion University, Beersheba, Israel
radami@post.bgu.ac.il
[3] Cornell University, Ithaca, NY, USA
tt544@cornell.edu
[4] Red Hat, Inc., Raleigh, USA

Abstract. Software containers provide a light-weight counterpart to virtual machines, utilizing the native host operating system to efficiently manage virtualization. Though efficient, this sharing of resources opens a potentially exploitable communication channel between collocated containers. To this end, we present a novel class of container isolation attacks that exploit union mounting, a key infrastructural component that allows for file system resource sharing among containers. We show that these vulnerabilities enable an unprivileged attacker, running a container on a shared commercial platform, to attack vulnerable victim containers on the same platform. More precisely, we showcase two attacks: one, which gleans information from the vulnerable containers, and another which establishes a covert side channel for exfiltrating data from the victim. Our attack implementations leverage a page-cache attack (CVE-2019-5489), but the attack surface is intrinsic to the efficiency needs of container management, and they apply even to the most recently patched Linux kernels. Our results highlight the need to rethink the page cache design in the context of multi-tenant clouds, and we propose some partial mitigations in this direction.

Keywords: Container security · Cloud security · Covert-channels

1 Introduction

Software containers are widely used for light-weight virtualization in modern cloud, edge, fog, and Internet of Things (IoT) platforms [11,14,27,31]. The light-weight properties of these containers enable fast boot times, easy migration, and storage savings [22], making them a popular resource for a wide variety of computing contexts. For example, Linux container systems typically achieve storage savings by utilizing *union mount file systems* [39] to provide an abstraction of separate file systems despite significant sharing of common files.

As the number of container instances and shared files increase, the storage cost per container instance goes down [22]. This same storage saving properties can also result in faster boot times, faster file system mounts, and faster migration. However, these performance-enhancing properties often come at a price to the execution isolation, when compared to traditional virtualization techniques such as virtual machines.

In this work, we show that current container isolation approaches are insufficient in protecting collocated containers from one another. In particular, we introduce a novel attack that exploits an interplay between union mount file systems and the Linux page cache, and allows an unprivileged attacker to exploit infected containers to extract sensitive information from a victim. An attacker can then use this side channel to leak sensitive information about the victim containers or to function as a data mule. We note that this new attack surface is *fundamental* to the current design of efficient container architectures, and it identifies a core trade-off between the container isolation and efficiency. A fix would thus likely require a significant re-architecture or efficiency loss for modern container infrastructures.

Contributions

Our main contributions in this work are:

1. Identification and exploitation of a novel covert channel that is fundamental to the efficiency of popular software container engines.
2. Identification of a related, novel cross-container side-channel.
3. Demonstration of our exploitations in proof-of-concept attacks on the (i) Docker Engine and (ii) Azure Kubernetes Service within a commercial multi-tenant cloud.

The attacks described in this work have been responsibly disclosed to Microsoft Azure and Docker.

Roadmap

The rest of this work is organized as follows. In Sect. 2, we discuss the main components of the software container technology that we exploit in our attacks, and relevant related work. In Sect. 3, we introduce the basic elements of our attack, including our threat model. We then explicate a variety of end-to-end attacks in Sect. 4, together with experimental evidence of their feasibility. Finally, we discuss potential mitigations of our attacks in Sect. 5 and conclude with Sect. 6.

2 Background

We first present a relevant overview of how containers are implemented in Linux and describe the various isolation and security mechanisms used to prevent collocated containers from influencing one another. We then describe the functioning of the Linux page cache and detail how the kernel accounts for and evicts pages in the cache. We end this section with a discussion of CVE-2019-5489 [1], which we leverage in our proof-of-concept attack implementations.

2.1 Linux Containers

Containers allow a user to create a layer of isolation from a host machine, somewhat like Virtual Machines. Indeed, this isolation, and the resultant security, is one of the main selling points of container technologies [10]. However, unlike Virtual Machines, containers do not come with their own operating system kernel, and instead share the kernel of their host operating system.

Linux containers are built using core features provided by the kernel: *Linux Security Modules*, *Namespaces*, and *Control Groups*. Linux Security Modules (LSMs) provide additional security checks on top of standard access control, and include implementations like AppArmor [24] (Ubuntu and Debian distributions), and SELinux [34] (Red Hat Enterprise Linux). Namespaces are typically used to isolate process views from one another [7].

Control Groups (or `cgroups` [35], for short) help limit resource usage of processes (memory, CPU, I/O, *etc.*), and they can therefore help ensure fair utilization of container resources running on a common host [35]. Corbet [16] describes how Linux manages host memory across different `cgroups`, noting that pages can be shared across processes in separate `cgroups`. Indeed, this is an optimization that the container platform Docker uses to reduce a container's memory footprint. Seeing as similar containers may run from a common base image, Docker avoids copying all image files for each newly created container, and instead utilizes a *union mount file system* to share common files, with Copy on Write (CoW) semantics [19] to handle changes and versioning, as they occur.

Within the union mount file system, each container image consists of one or more layers, and each layer stores file system state changes with respect to lower layers. Containers only modify their topmost layers, and modification logic is typically implemented through the `overlay2` container engine driver. This `overlay2` driver (OverlayFS) implements layering in such a way that files in the inner file system layers are presented as the same `inode` (*i.e.,* containers observe the same internal data structure for common files). The design is particularly significant when considered in conjunction with the Linux page cache design that repurposes these `inodes` as indices [12]. This results in page cache sharing between collocated containers, and allows for cross-container attacks even in the presence of kernel isolation measures imposed by LSMs and namespaces.

2.2 Linux Page Cache

Because disk input/output (I/O) is relatively slow, most operating systems employ some sort of "page cache" to exploit temporal and spatial locality of page reads (from disk) and optimize page-fault handling. The operating system typically stores such frequently used pages by "caching" them in unused areas of main memory.

On Linux, page cache management is done by maintaining a doubly-linked list of pointers to those pages cached in memory. This list is divided into two parts 1) the `active_list`, which contains recently accessed pages, and 2) the `inactive_list`, which contains pages that are candidates for reclamation. It is important to note that these lists are *global*, in the sense that they are shared among all processes on the system. Initially, the kernel uses the function `lru_cache_add()` to place pages on the `inactive_list`, and then uses `mark_page_accessed()` to move a page to the `active_list` upon access. The kernel tries to maintain the `active_list` at about two thirds the size of the `inactive_list`, and uses the `refill_inactive()` to shuttle pages from the bottom of the `active_list` to the `inactive_list`, as needed, to maintain the two-thirds ratio [8].

Page reclamation is done from the `inactive_list`. The pages that are evicted from the page cache are written back to disk, and their corresponding entries are deleted from the `inactive_list`. Indeed, by maintaining such separate `active_list` and `inactive_list`, the Linux kernel implements a variant of a Least Recently Used (LRU) page reclamation policy, wherein the least recently used pages are generally cleared out first.

2.3 Related Work

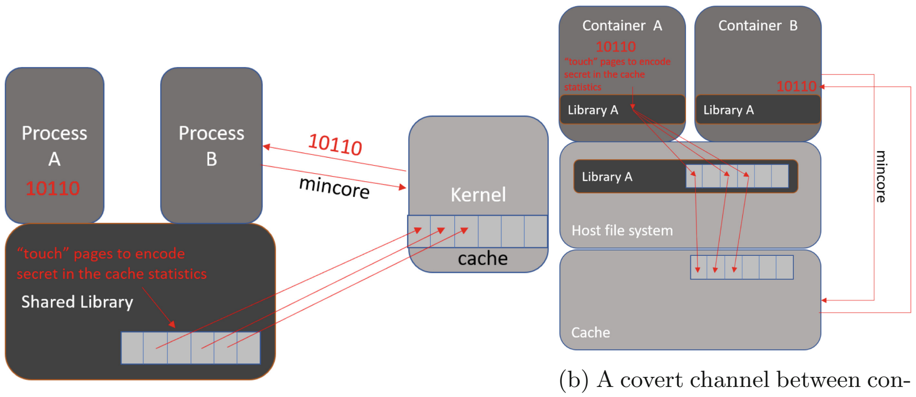
Although the idea of light-weight operating system-based virtualization can be traced back to the roots of `chroot` in BSD [13] and Solaris Containers introduced in 2004 [38], it was probably not before the cloud computing boom around 2010s s and the introduction of Docker in 2013 [20] that the concept reached wide-scale industrial adoption. Nowadays, containers are the *de facto* standard for light-weight virtualization in the cloud [14]. It is thus not surprising that they have also attracted the attention of cybersecurity researchers' and many vulnerabilities have been discovered in the process [14,23,32].

Gao et al. [23] considered various information leakages in multi-tenant clouds originating from operating-system-based resource sharing (as opposed to hypervisor-gaited sharing among traditional VM-based approaches [41]). Lin et al. [32] compiled an extensive set of privilege escalation attacks that are possible in modern cloud deployments. Despite an increasing body of literature on their security weaknesses [42], software containers remain the popular tool in the industry, suggesting that perhaps their performance benefits (*e.g.,* reliability and efficiency) outweigh their security risks in many use cases. For instance, software containers were recently proposed as a solution to module isolation in enterprise applications [28], energy-balancing fog computing architectures [33], and efficient Internet of Things (IoT) gateways [21].

On the other hand, several researchers [25,43] have recently demonstrated side-channel vulnerabilities in the Linux page cache. In this work, we continue the preliminary work in [40] and focus on attacks at the intersection of (i) the exploits rooted in the design of software containers, (ii) traditional operating-system optimization techniques, and (iii) modern multi-tenant cloud deployments.

3 Attack Elements

Our attacks utilize the Linux system call `mincore` to determine cache-based metadata about files and libraries within the same container, or between separate collocated containers on a major cloud container platform. This can be used for information leakage across these entities or for covert communication, as illustrated in Fig. 1. Despite a superficial resemblance to Yarom and Falkner's Flush+Reload [45], our attack relies solely on operating system-level abstractions (and not the structure of the underlying hardware caches). We next detail the main elements of our attack, starting with the threat model, continuing through the page cache attack surface, and ending with a deployment to which our attacks apply.



(a) Leaking sensitive data from Process A to B.

(b) A covert channel between containers.

Fig. 1. Two attack scenarios.

3.1 Threat Model

We assume that all relevant containers are created from the same base image and run a Linux kernel patched against CVE-2019-5489. As a result, we expect that the unmodified files in the file-system of both these images are shared and thus accessible through the common page cache.

For our covert-channel attacks, we further assume that both attacker and victim include a payload that allows a) the victim container to leak information, and b) the attacker to read the information sent (from a covert channel) and decode it. We note that neither of the payloads on the victim or attacker entities require special privileges – they can run with simple user-level privileges on the container. For our side-channel attacks, only the container that spies on the victim needs to have a payload that allows it to monitor the state of the page cache.

Though the mechanism for infecting a vulnerable container is explicitly out of scope for this paper, Table 1 describes various techniques by which this could be done in the wild, including tampering with public container repositories, using content injection vulnerabilities, and leveraging vulnerable applications or OS libraries [2,15,37].

Table 1. Mechanisms to infect the containers.

Threat	Vulnerability/Use Case	Asset	Control Recommendation
Malicious code (source)	Injecting malicious code in build environment	Build environment	Static tool analysis and checking with DB for open source libs. Sandbox for checking in runtime.
Malicious code (binaries)	Injecting malicious code in build environment	Build environment	Integrity checking tools. Comparison and signatures.
Vulnerable code in application	Vulnerable code in build environment from imported packages, and in dev code	Code	Static tool analysis and checking with DB for open source libs. Sandbox for checking in runtime.
Vulnerable libraries in the OS	Vulnerable code in build environment from imported packages, and in dev code	Code	Static tool analysis and checking with DB for open source libs. Sandbox for checking in runtime.
Content Injection	Injecting or replacing existing image in container repository	Container repository	Signing every image before storing in repository and write protection on existing data.
Repository compromise	Modification and tampering with the whole repository (e.g. modifying the image, signature or location)	Container repository	Protection and access control on the repository.
Content tampering	Tampering with Docker image content that is stored in the container repository	Container repository	Save the images content in an encrypted and signed fashion.
Deploy malicious container	Deploying malicious container to the same runtime environment	Production environment	Allow only signed images to be deployed. Allow deployment only from specific container repo (e.g. ip based).

3.2 Page Cache Attack Surface

The page cache plays a prominent role in the specific implementation of our attacks, and we next describe some of the ways it can be abused.

Monitoring. There are two typical mechanisms for an attacker to monitor the state of the page cache. One approach utilizes functions that Linux provides especially for this purpose, such as `mincore` [25,43], `preadv2`, and others [3].

For instance, an attacker could invoke `mincore` on every page within a shared victim file to see which pages of the victim are "in core" (*i.e.,* in RAM). The system call `preadv2` can also be used in a similar way with a `RWF_NOWAIT` flag to check the page cache [29]. The advantage of using dedicated functions is that their measurements are *non destructive*, meaning that the act of reading the cache state does not modify it (unlike previous attacks on other caches [45]).

Eviction. One of the easiest ways to remove a victim page from the page cache is by using functionality that Linux provides for this very purpose. The Linux Kernel proc-filesystem contains a file `/proc/sys/vm/drop_caches`, and writing 1 to this file clears out the entire page cache [4], although containers, as a security default, cannot write to the `/proc` directory. To enable a writable `/proc` directory, the container needs to be explicitly passed a flag on startup, which is not a realistic attack scenario. Aside from this, the kernel also provides `posix_fadvise` [9], which, when invoked on a file with the parameter `DONT_NEED`, requests a flush of the pages belonging to said file from the page cache. Another function, `madvise`, provides similar functionality [6]. However, these functions do not seem to work reliably within containers. As such, though the usage of these functions has indeed been possible for some categories of page cache attacks outside of containers [25,43], it does not seem feasible within our context.

Another method of evicting victim pages involves triggering the kernel's page frame reclaimer and coercing it to remove the victim's pages from the cache. This can be accomplished in a few different ways. One approach is to increase memory pressure on the system (*e.g.,* by spawning new processes or by making an existing process map more and more pages into memory); since the page cache fills up *all* unused memory on the system, memory pressure would cause page evictions. Indeed, this approach is possible because the page cache is globally shared with no per-process or per-user quotas enforced; instead, the page cache is "indexed" by inodes, which are unique per file system and deduplicated across containers. In the extreme case, it is possible for one process to take up almost all of the page cache by mapping many different pages into memory and continuously accessing them, thereby kicking out pages belonging to other processes. Variations of these techniques have been attempted in [5], but they suffer from two limitations: a) they require significant time to evict victim pages (around 8 s according to [25]), and b) they cause the system to become unstable because they require the attacker to utilize nearly 100% of the system memory [25].

In [26], Gruss et al. demonstrate an improved method of page eviction called *memory waylaying*, which takes around 2.86 s to evict a target page on Linux. Memory waylaying exploits the idea that the attacker does not need to explicitly map pages into memory in order to monopolize the page cache. Instead, the attacker can execute the following for every page in a large file:

1. **Map** the page using `mmap` with `read` and `execute` permissions (such pages have a higher priority in the page cache, and thus are more likely to stay there);
2. **Touch** this page in order to bring it into the page cache, since `mmap` is lazy and does not actually bring a page into memory until it's explicitly used; and,
3. **Unmap** the page.

The attacker does not need to use an existing file that is shared with the victim to accomplish this attack–any large file will work for this purpose. In this way, one can evict most pages from the page cache *without* utilizing too much system memory. This technique was further improved in [25] to achieve an eviction time of 0.149 s. In the improved approach, instead of mapping+unmapping pages from a large attacker-generated file, the attacker can re-use pages that are *already* present in the page cache by simply touching them. This has the effect of making the pages appear to be more frequently used, and, consequently, they are more likely to stay in the page cache.

CVE-2019-5489. The page-cache side-channel [25,43] is a hardware-agnostic side-channel that allows an unprivileged process (or service) to spy on other collocated processes. The main idea behind this attack involves the unprivileged process regularly polling select pages in the common page-cache, and translating page-cache residency (or non-residency) into information about the execution path of the victim process. This attack can allow the attacker to implement UI redressing, break ASLR, and even guess generated passwords for vulnerable implementations [25].

Registered as CVE-2019-5489 [1], this attack led to a patch to the `mincore` syscall in the Linux kernel, which allows for efficient polling of page-cache pages. The patch changed the semantics of the `mincore` system call to reveal only actively "mapped" pages rather than "cached" pages [17], thus thwarting an attacker from seeing if particular page is in the page cache.

However, the patch to CVE-2019-5489 ended up causing serious efficiency issues, and was thus reverted [18] with the following comment:

> For Netflix, losing accurate information from the `mincore` syscall would lengthen database cluster maintenance operations from days to months. We rely on cross-process mincore to migrate the contents of a page cache from machine to machine, and across reboots.

This then led the kernel developmental team to introduce a new patch, which added an additional permission-checking routine, `can_do_mincore`, that is invoked before `mincore` to check user permissions. If the user passes the permission checks, they may obtain the output for `mincore` as before. However, if the permission checks fail, the user cannot obtain the information that `mincore` would otherwise disclose. This permission check is illustrated in the following code.

```
 1 static inline bool can_do_mincore(struct vm_area_struct *vma)
 2 {
 3   if (vma_is_anonymous(vma))
 4     return true;
 5   if (!vma->vm_file)
 6     return false;
 7   /*
 8    * Reveal pagecache information only for non-anonymous mappings that
 9    * correspond to the files the calling process could (if tried) open
10    * for writing; otherwise we'd be including shared non-exclusive
11    * mappings, which opens a side channel.
12    */
13   return inode_owner_or_capable(file_inode(vma->vm_file)) ||
14     inode_permission(file_inode(vma->vm_file), MAY_WRITE) == 0;
15 }
```

The relevant portion of this routine is Line 13, which returns `true` if the caller of `mincore` is the owner of the file that backs the memory region on which `mincore` was called, or else if the caller has write permissions to the file. This basically prevents any user from calling `mincore` on a file that they do not exclusively own, or that to which the do not have write permissions (*e.g.,* shared-executable files that were exploited in [25]).

However, in the context of containers, this new patch is insufficient. The container management engine shares common files across different containers through the Copy on Write mechanism, meaning that each container gets the illusion of exclusive access even though the underlying file is shared. In other words, in a container environment, the condition that checks for ownership of the file will pass and return `true` even when the actual file is shared across multiple containers. For files that are rarely modified (*e.g.,* shared-execute only binaries - ironically the very type of files this patch was meant to protect), this opens the door for vulnerabilities.

3.3 Target Deployments

The attacks at the core of our work rely upon collocation of two containers (attacker and victim) on the same virtual machine. One way of achieving such a collocation is by seeding vulnerabilities within a library that is common to a variety of containers (*e.g.,* npm [15]), thereby making it likely that at least one of the infected containers be collocated with an attacker. Another mechanism for doing this is to repeatedly respawn an attacker container until it is collocated with a vulnerable container (as ascertained through the page-cache side-channel described in Sect. 3.2).

For purposes of exposition, we restrict our further deployment discussion to the Kubernetes framework [30], an extremely popular mechanism for running workloads on one or more hosts. Note that all containers that share a common host also share the same cache, thus making them potentially vulnerable to our attacks. Following container infection, an attacker can identify sensitive data within the process and then encode it into the page cache, thereby leaking it out of isolation. The attacker can use the covert channel from Sect. 4.2 as a data mule to transfer the sensitive data between containers until the data reaches a container with a web-server (or other outward-facing service, *cf.* [25]), which

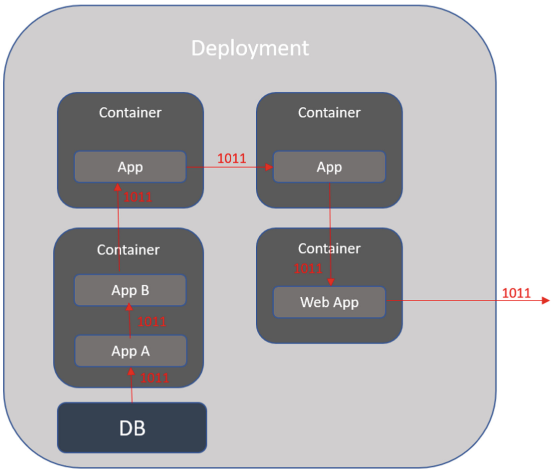


Fig. 2. Chained exploitation within Kubernetes

allows exfiltration of the sensitive data out of the deployment context as depicted in Fig. 2.

4 Attack Implementations

We next describe concrete end-to-end attacks based on the attack elements introduced in Sect. 3, starting with the simpler side-channel attacks and moving toward covert-channel attacks. For each attack, we describe the general attack followed by our own experimental proof of concept.

Our experiments are performed within two environments.

a) **Ubuntu and Docker:** An environment based on Ubuntu 20.04 and Docker 20.10.7.
b) **Azure Kubernetes Service:** An environment within the Azure Kubernetes service, based on Kubernetes version 1.22.6 and whose workload is defined in the file `azure_deployment.yaml` on out GitHub repository [36].

4.1 Side-Channel Attacks

In our side-channel attack, an attacker container infers the *high-level actions* of a victim container using the cache monitoring technique described in Sect. 3. Our victim and attacker containers are spawned from the same base image and thus share the same files in their inner layers (see Sect. 2.1).

MySQL Side-Channel. For the first experiment, we choose to monitor the pages of the MySQL server executable (*i.e.*, the file `/usr/sbin/mysqld`). The main goal of our attacker in this experiment is to distinguish between the MySQL server actions performed by the victim, and, to that end, the attacker seeks actions whose paths through the executable will result in different page cache traces. For the purpose of this experiment, the victim chooses between the following actions 1) login action using the correct root password, and 2) login attempt using an incorrect password. We consider the attack successful if the attacker container has a non-negligible advantage in discerning which of the two possible victim actions were taken.

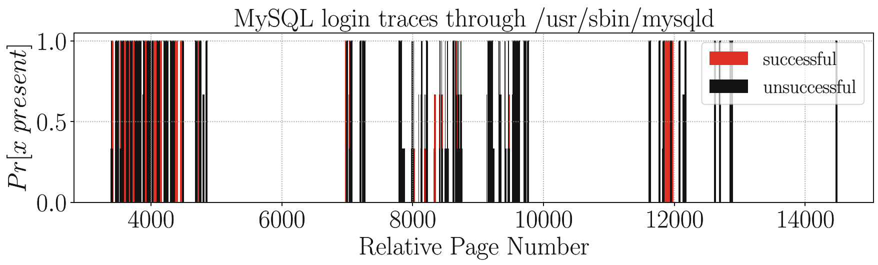


Fig. 3. Probability of page cache residency calculated over 10 experiment runs. The attacker can distinguish between the successful login into MySQL and an unsuccessful login attempt in the victim container.

In our experiments, the victim performs one of these two actions, and the attacker utilizes the page cache residency of its own view of `/usr/sbin/mysqld` to discern the action. The results of our experiments are depicted in Fig. 3, where we note that there exist pages that will certainly be loaded for one action and not in the other – for instance, the pages 11962–11984 are exclusively associated with a successful login. On the basis of our experimental evidence, we conclude that the attacker can reliably infer which of the two actions has been performed by the victim container, thus compromising container isolation. We further discuss the limitations of this attack in Sect. 5.1.

Bash Login Side-Channel. Another high-level event of possible interest to the attacker is the identification of when the victim's initiates of an interactive shell within its own container. An attacker can monitor when this happens by looking for the existence of files specific to a shell session in the page cache. We demonstrated this capability experimentally on two collocated Ubuntu 20.04 Docker containers. The attacker container identifies the start of the victim's interactive shell by applying `mincore` to monitor the cache activity of `/bin/bash` and `/etc/bash.bashrc`, which are pulled into the page cache by the victim process.

4.2 Covert-Channel Attacks

Encoding and decoding covert messages through the page cache registry is essential to a successful cover-channel attack. The standard approach for encoding involves mapping message bits to file pages. For example, a file with 8 pages with all its pages uncached, could encode the vector "00010100" by reading (and thereby loading into cache) pages 4 and 6 of the file. A second container would use `mincore` to read the file's page cache status and decode the vector.

Our proof-of-concept exploits are based on some simple example applications that we have designed (their source code is available at [36]):

1. `spy_on` – An application that outputs the current status of a file's cache registry.
2. `read_page` – An application that reads pages of a given file.
3. `secret_text_infected` – A malicious code example that encodes the victim's password into the page cache.
4. `eviction` – An application that clears the page cache.
5. `reproduce_AKS.sh` –A script that recreates the scenario using the application spy_on and read_page in Azure Kubernetes service.
6. `reproduce_VM.sh` – A script that recreates the scenario using the application spy_on and read_page in the environment a) described earlier in this section.

Sensitive Data Leakage from a Process Context. Our first exploit identifies sensitive information within a process context and leaks it out of the process context. The malicious code inspects the data flowing through the process (*e.g.,* user input), and if the data is be identified to be sensitive (*e.g.,* passwords), our malicious code encodes it bit-wise into the page cache by reading corresponding pages of a file.

Experiment. In order to experiment with exploiting the vulnerability, we have written a proof-of-concept application named `secret_text_infected`, which takes as input some text and the name of a shared file or library with which to encode the secret using the page cache. The code scans the input text for a potential password, and encodes any such password via the cache registry of the file. For this experiment we used an `nginx` container from docker-hub, within which we included our compiled code. Starting from the attacker-controlled container, we executed `spy_on` on the file `/usr/sbin/nginx-debug` in order to make sure that the pages of the file are not in the cache. The output of the execution should be `[00..00]`, or else the pages of this file are already somehow in the cache. In the latter case, we ran a program `eviction` in order to cause the cache to clear out the pages of the file `/usr/sbin/nginx-debug`.

We verified that the file's pages are not in the cache by re-executing `spy_on` on the file. From the victim's container, we ran `secret_text_infected` with the input text "d!" and the file `/usr/sbin/nginx-debug`. The program identified the text as a potential password and encoded the text into the cache registry of the file by reading specific file pages. Finally, the attacker re-ran `spy_on` on the file `/usr/sbin/nginx-debug` and observed `[00..11110000110000]` as the result, revealing the encoded secret. This demonstrated our ability to leak a secret from a process context.

Covert Channel Between Containers. The second exploit we were able to demonstrate is a covert channel used to mule and transfer data between containers. The malicious code will use a file shared between the containers to transfer data through the page cache.

Experiment. The setup included two containers running an `nginx` container image from docker-hub, to which we copied our compiled binaries. We first executed `spy_on` in *Container A* on the file `/usr/sbin/nginx-debug` in order to make sure that the pages of the file were not in the cache. The output of the execution should be `[00..00]`, or else the pages of this file are already in the cache and can be evicted with the program `eviction`. We verified that the file's pages were not in the cache by re-executing `spy_on` on the file. Next we executed `spy_on` in *Container B* on the file `/usr/sbin/nginx-debug` to make sure that the pages of the file are not in the cache. The output of the execution was again `[00..00]`.

Once we made sure that the file `/usr/sbin/nginx-debug` has no pages in the cache, we executed `read_page` on *Container A* with the filename `/usr/sbin/nginx-debug` and the page number `0` as parameters. This triggered the last page in the file to be loaded into the cache. We then ran `spy_on` in both *Container A* and *B* one more time. The result on both containers was `[00..01]`, demonstrating that we can use the page cache registry to create a covert channel between collocated containers.

5 Discussion

We next discuss some limitations, mitigations, and future directions of our work.

5.1 Limitations

Side-Channel. The side-channel attack we described in Sect. 4.1 requires two attacker capabilities 1) being able to collocate a malicious container with the victim, and 2) being able to spawn the container from the same base image as the victim. If the former is not fulfilled, then the attacker will not share the page cache and the union mount file system with its victim, which precludes attacker's ability to run the attack. On the other hand, if the attacker cannot spawn the container from the same base image, then there may be no shared files that the attacker can weaponize. The silver lining is that spawning containers from different base images may not be in the best interest of a deployment that aims at maximizing its performance (as we discuss in Sect. 2.1).

Covert-Channel. The covert-channel attack we described in Sect. 4.2 relies on (i) being able to infect collocated containers with malicious code, (ii) being able to flush the pages of a file from the cache, and (iii) being able to remotely exploit a container.

Infecting a collocated container may seem challenging, but it is possible if an attacker performs a broad attack such as infecting a library or a container base image, as happened recently with the `npm` attack as described in [15]. Flushing the page cache is necessary to create an effective covert channel as described in [25] and Sect. 3.2. However, eviction is not stealthy and might be identified by monitoring the containers [44,46]. In order to leak data out of a victim's infrastructure, its deployment will need to include a container with outbound network access, such as a web server (see [25]).

5.2 Mitigations

The attack surfaces described in this work are in many ways fundamental to the vision of the light-weight container environment.

For one, containers are meant to run or layer off common base images, making the sharing of common files essential to their light-weight nature and reasonable performance. Without such file sharing, all files across all container layers would have to be duplicated for each container, significantly diminishing the benefits of such an infrastructure over the relatively heavier-weight virtual machine approach. The page-cache querying functionality upon which our attack implementation is based is also not amenable to a simple and efficient correction, as described in Sect. 3.2. Indeed, unless the efficiencies of a global page-cache are obviated, one may also use a timing side-channel to infer page cache residence of different pages (*i.e.,* pages that are cached should be accessible more quickly than non-cached pages). However, such an attack will be much harder to mount, since the timing measurement is destructive, *i.e.,* the act of measuring the load time of a specific page brings the page into the page-cache, thus necessitating eviction after every measurement. Eviction already presents as a bottleneck of this attack, thereby rendering a timing based attack much less feasible than the attack we present.

At its core, our attack shares the style of a Flush+Reload [45] attack, for which there typically exist two broad mitigation approaches: defense against 1) the Reload aspect, or 2) the Flushing aspect of the attack. Defending against reloading would involve eliminating shared pages across processes (and therefore containers). However, this would also obviate the performance benefits of sharing pages, making it an impractical solution.

An alternate mitigation would involve complicating the client's ability to flush pages across different processes. This would prevent an attacker from flushing the victim's pages out of the page cache, a step that is critical to executing multiple iterations of our attacks. This approach would likely require modification of the global page replacement policy that Linux employs (described in Sect. 2). One way to effect this change would be to employ per-process page cache working sets, and prevent any process from flushing out pages of any other process's working set. By doing so, each process would be constrained to its own working set such that it cannot affect the others.

Indeed, this is what was done for the Windows operating system in response to CVE-2019-5489 [1] that arose out of [25]. Since Windows already employs a similar mechanism, we posit that the performance overhead of such a mitigation would not be significant, and would be outweighed by the security benefits. However, we leave a concrete analysis of this to future work.

5.3 Future Directions

During the implementation of our covert channel described in Sect. 4.2 on Microsoft Azure, it appeared that memory exhaustion in the attacker's container may crash the container, causing the container within the pod to repeatedly reboot, meaning that the Kubernetes pod never recovers. Assuming that the attacker has collocation capabilities similar to those in our covert-channel threat model, this pattern could allow the attacker to mount a Denial of Service (DoS) attack on the victim, on demand, by simply performing a memory exhaustion from its own container's context. As a future direction, we will investigate the root causes and circumstances for such behavior on Microsoft Azure, and possibly other platforms that offer Kubernetes services.

6 Conclusion

This work has considered the ability of software containers to isolate their work from other containers running on the same host. We have demonstrated that some resource sharing responsible for efficient container implementation may also be inherently vulnerable to cross-container communication.

Specifically, we have identified a class of side-channel attacks against the union file system (UFS) that manages file access for containers on a host, and demonstrated its use in two scenarios between a vulnerable container and an unprivileged foreign container. In the first scenario, the foreign container can learn private information about the execution of the vulnerable container, and in the second scenario, the foreign container colludes through a covert channel with code injected into the vulnerable container, thereby potentially bypassing firewalls or intrusion detection systems. In both cases, channel information flows through metadata about the page channel undergirding the UFS.

Mitigation of this class of attacks may be extremely difficult, as the attacks flows from an efficiency that is core to container execution. Nevertheless, we have posed some potential avenues for hardening systems going forward.

Acknowledgments. The authors would like to thank David Starobinski and Aryeh Kontorovich for early involvement and support of this work. The authors would also like to thank Red Hat and the Boston University Red Hat Collaboratory for their support.

References

1. CVE-2019-5489. https://access.redhat.com/security/cve/cve-2019-5489. Accessed 21 Oct 2019
2. CVE-2021-44228. https://access.redhat.com/security/cve/cve-2021-44228. Accessed 28 Mar 2022
3. Defending against page-cache attacks. https://lwn.net/Articles/776801/. Accessed 21 Oct 2019
4. Drop_caches. https://linux-mm.org/Drop_Caches. Accessed 21 Oct 2019
5. Experiments and fun with the linux disk cache. https://www.linuxatemyram.com/play.html. Accessed 21 Oct 2019
6. Madvise. http://man7.org/linux/man-pages/man2/madvise.2.html. Accessed 21 Oct 2019
7. Namespaces(7). http://man7.org/linux/man-pages/man7/namespaces.7.html. Accessed 21 Oct 2019
8. Page frame reclamation. https://www.kernel.org/doc/gorman/html/understand/understand013.html. Accessed 21 Oct 2019
9. posix_fadvise. https://linux.die.net/man/2/posix_fadvise. Accessed 21 Oct 2019
10. What is a container? - docker. https://www.docker.com/resources/what-container/. Accessed 28 Mar 2022
11. Bernstein, D.: Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
12. Bovet, D., Cesati, M.: Understanding the Linux Kernel. Oreilly Associates Inc. (2005)
13. BSD Developers. Chroot (1982). https://docs.freebsd.org/44doc/papers/jail/jail-9.html. Accessed 16 Feb 2022
14. Casalicchio, E., Iannucci, S.: The state-of-the-art in container technologies: application, orchestration and security. Concurrency Comput. Pract. Expe. **32**(17), e5668 (2020)
15. Cimpanu, C.: Malware found in npm package with millions of weekly downloads, 23 October 2021
16. Corbet. Documentation/cgroups/memory.txt (2011). https://www.lwn.net/Articles/432224/
17. Linux Kernel Developers: "Change mincore() to count "mapped" pages rather than "cached" pages" (2019). https://gitlab.raptorengineering.com/meklort/talos-obmc-linux/-/commit/574823bfab82d9d8fa47f422778043fbb4b4f50e
18. Linux Kernel Developers: Revert "Change mincore() to count "mapped" pages rather than "cached" pages" (2019). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=30bac164aca750892b93eef350439a0562a68647
19. Docker. About storage drivers (2022). https://docs.docker.com/storage/storagedriver. Accessed 16 Feb 2022
20. Docker Developers. Docker project (2013). https://www.docker.com/. Accessed 16 Feb 2022
21. Dolui, K., Kiraly, C.: Towards multi-container deployment on IoT gateways. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–7. IEEE (2018)
22. Funari, L., Petrucci, L., Detti, A.: Storage-saving scheduling policies for clusters running containers. IEEE Trans. Cloud Comput., 1 (2021). https://doi.org/10.1109/TCC.2021.3104662

23. Gao, X., Gu, Z., Kayaalp, M., Pendarakis, D., Wang, H.: ContainerLeaks: emerging security threats of information leakages in container clouds. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 237–248. IEEE (2017)
24. Gruenbacher, A., Arnold, S.: AppArmor technical documentation (2007)
25. Gruss, D., et al.: Page cache attacks. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, pp. 167–180. Association for Computing Machinery, New York (2019)
26. Gruss, D.: Another flip in the wall of rowhammer defenses. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 245–261. IEEE (2018)
27. Hoque, S., De Brito, M.S., Willner, A., Keil, O., Magedanz, T.: Towards container orchestration in fog computing infrastructures. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 294–299 (2017)
28. Kehrer, S., Riebandt, F., Blochinger, W.: Container-based module isolation for cloud services. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 177–17709. IEEE (2019)
29. Kerrisk, M.: readv(2) - Linux manual page. https://man7.org/linux/man-pages/man2/preadv2.2.html. Accessed 29 Mar 2022
30. Kubernetes Develoeprs. Kubernetes. https://kubernetes.io/. Accessed 17 Feb 2022
31. Lin, L., Liao, X., Jin, H., Li, P.: Computation offloading toward edge computing. Proc. IEEE **107**(8), 1584–1607 (2019)
32. Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., Zhou, Q.: A measurement study on Linux container security: attacks and countermeasures. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp. 418–429 (2018)
33. Luo, J., et al.: Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. Futur. Gener. Comput. Syst. **97**, 50–60 (2019)
34. Mayer, F., Caplan, D., MacMillan, K.: SELinux by example: using security enhanced Linux. Pearson Education (2006)
35. Menage, P.: Cgroups. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt. Accessed 21 Oct 2019
36. Novak Boskov Naor Radami. Union buster (2022). https://gitlab.com/radaminaor/union-buster
37. U.S. Department of Homeland Security. Apache Log4J vulnerability guidance (December 2021). https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance. Accessed 28 Mar 2022
38. Oracle. Solaris containers (2005). https://docs.oracle.com/cd/E36784_01/html/E36848/zones.intro-1.html. Accessed 16 Feb 2022
39. Quigley, D., Sipek, J., Wright, C.P., Zadok, E.: UnionFS: user-and community-oriented development of a unification filesystem. In: Proceedings of the 2006 Linux Symposium, vol. 2, pp. 349–362 (2006)
40. Radami, N., Boskov, N., Tiwari, T., Trachtenberg, A.: Stash your cache - cross-container Linux page cache covert channel (2021). Poster
41. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 199–212. Association for Computing Machinery, New York (2009)
42. Souppaya, M., Morello, J., Scarfone, K.: Application container security guide. Technical report, National Institute of Standards and Technology (2017)

43. Tiwari, T., Trachtenberg, A.: Cashing in on the file-system cache. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, pp. 2303–2305. ACM, New York (2018)
44. Wang, Y., et al.: ContainerGuard: a real-time attack detection system in container-based big data platform. IEEE Trans. Industr. Inf. **18**(5), 3327–3336 (2022)
45. Yarom, Y., Falkner, K.: FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In: 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, August 2014, pp. 719–732. USENIX Association (2014)
46. Zou, Z., Xie, Y., Huang, K., Xu, G., Feng, D., Long, D.: A docker container anomaly monitoring system based on optimized isolation forest. IEEE Trans. Cloud Comput. **10**, 134–145 (2019)

Mutual Accountability Layer: Accountable Anonymity Within Accountable Trust

Vanesa Daza[1,2], Abida Haque[3](✉), Alessandra Scafuro[3], Alexandros Zacharakis[1], and Arantxa Zapico[1]

[1] Pompeu Fabra University, Barcelona, Spain
{vanesa.daza,alexandros.zacharakis,arantxa.zapico}@upf.edu
[2] Cybercat, Quebec, USA
[3] North Carolina State University, Raleigh, USA
{ahaque3,ascafur}@ncsu.edu

Abstract. Anonymous cryptographic primitives reduce the traces left by users when they interact over a digital platform. But they also prevent a platform owner from holding users accountable for malicious behaviour. Revocable anonymity offers a compromise by allowing only the manager of the digital platform to de-anonymize a user's activities when necessary. However, a misbehaving manager can abuse their de-anonymization power by de-anonymizing activities without the user's awareness.

Although previous works mitigate this issue by distributing the de-anonymization power across several entities, there is no comprehensive and formal treatment where both accountability and non-frameability (i.e., the inability to falsely accuse a party of misbehavior) for *both* the user and the manager are *explicitly* defined and provably achieved.

In this paper we formally define *mutual accountability*: a user can be held accountable for her otherwise anonymous digital actions and a manager is held accountable for *every* de-anonymization attempt. Also, no honest party can be framed regardless of what malicious parties do.

In contrast with previous work, we do not distribute the de-anonymization power across entities, instead, we decouple the power of de-anonymization from the power of monitoring de-anonymization attempts. This allows for greater flexibility, particularly in the choice of the monitoring entities.

We show that our framework can be instantiated generically from threshold encryption schemes and succinct non-interactive zero-knowledge. We also show that the highly-efficient threshold group signature scheme by Camenisch et al. (SCN'20) can be modified and extended to instantiate our framework.

1 Introduction

We target *accountable* anonymity: an authorized user of a digital platform can generate a value anonymously[1], but when deemed necessary, a value can be

[1] This work focuses on the *application* layer and not on the network layer. We assume that all users communicate over anonymous channels.

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 318–336, 2022.
https://doi.org/10.1007/978-3-031-07689-3_24

de-anonymized and linked to the identity of its creator. The right balance between the two desirable properties is an important and difficult task to achieve. Recently, there are regulatory attempts on the matter. The Council of the European Union recently published a document on the matter [1] that states: *"Protecting the privacy and security of communications through encryption and at the same time upholding the possibility for competent authorities in the area of security and criminal justice to lawfully access relevant data for legitimate, clearly defined purposes [...] are extremely important."*

However, accountability is naturally in tension with perfect anonymity because it is achieved via a master trapdoor that enables a designated party to de-anonymize *any* message exchanged through the platform. Anonymity then holds conditionally on how the designated party *uses the trapdoor*.

Shouldn't the designated party be accountable for their de-anonymization activities? Shouldn't users at least be aware when de-anonymization activities take place?

Accountability: What is Missing from Previous Work. Earlier works recognized that the manager can abuse a master trapdoor [2–7]. The countermeasures proposed in the literature rely on distributing the role of the manager across n parties. That way, any de-anonymization can be performed only if a minimum number of parties agree (e.g., [8,9]). Other solutions [6,7] instead introduce a new party whose task is to allow the manager to de-anonymize the message. However, all existing definitions either lack generality or formality of the security goals for manager accountability. Consequently, the definitions and constructions can be applied only to specific scenarios. In particular,

- Weak Manager-Accountability Guarantees. Most work (e.g., [2,9]) splits the manager into n parties. This approach lessens the manager's ability to abuse de-anonymization since multiple parties must agree to conduct the process. However, these works do not introduce any formal accountability guarantee against the managers, and often omit discussion on how the n parties are selected so that the underlying trust assumption can be satisfied.
- Lack of Comprehensive Definitions. Works such as [6,7] introduce a third entity (called "admitters" in [6]) that enables the manager to de-anonymize. This introduces an added layer of protection for the users, but the formal security definition provided leaves out any protection for the manager. Specifically, it does not consider the non-frameability of the manager, making an implicit assumption on the good behavior of the admitters and users.
- Lack of Generality. Even in the more general framework where two separate entities control the de-anonymization power [6], the de-anonymization activity has a pre-defined granularity and is, therefore, suitable only in specific applications. For instance, in [6] the admitters give the manager a message-dependent trapdoor to de-anonymize *all messages* that have the same value (e.g., all messages that have the same date). This is useful for the application proposed in [6], where one wishes to de-anonymize all messages signed at a specific time with a single trapdoor, but not in general.

– Lack of Flexibility. In earlier work, the manager accountability property is achieved by using parties who are also involved in the functioning of the platform (e.g., [8]), hence they might not be *independent of* the platform. Furthermore, there is an underlying assumption that these parties are fixed for the entire lifetime of the system. There is little discussion on how such parties are chosen and why they would be trustworthy and active throughout the lifetime of the system.

1.1 Our Contribution

In this work, we provide a general framework, the *mutual accountability layer*, to capture accountability guarantees for *both* the users and the managers. We believe that our definitions can enable the designing of mutually accountable systems for a wider set of applications than the ones considered so far in literature. We summarize our contributions and then elaborate on each in the following sections.

1. **A Formal Definition for Mutual Accountability.** We introduce the Mutual Accountability Layer (MUTAL), a cryptographic primitive that captures the properties of anonymity for the users, accountability, and non-frameability, for both the users and the manager. Our new definitions sharpen the ones provided in previous works, providing flexibility and verifiability properties. To guarantee accountability to all parties, we introduce **guardians**, a set of (potentially malicious) monitoring parties that oversees every de-anonymization process, but it does not learn anything about the de-anonymized users. Security is guaranteed if the number of malicious guardians is below a threshold.
2. **Instantiations.** We provide a general template to construct a Mutual Accountability Layer that is based on a two-layer encryption: one for the guardians and one for the manager. Only if both collaborate, the identity of a user can be reconstructed. Besides our general construction, we show how to change the group signature scheme of [9] so that it can be used to instantiate a MUTAL scheme. The second approach is less general and uses bilinear maps.
3. **Evolving Monitoring Parties.** We use results on proactive secret sharing [10] and its more recent implementations [11] to allow the guardians (the monitoring parties) to change over time.

1.2 Formal Definition for Mutual Accountability (MUTAL)

The *Mutual Accountability Layer* (MUTAL) holds both the users and the manager accountable. This layer can be added to any content that members of a digital platform generate and thus is not tied to any specific application. Unlike previous work, we do not distribute the manager's role among n parties. We model two entities with separate roles[2]. The platform manager is the entity exclusively entitled to de-anonymize users, while the set of *guardians* are solely entitled to give permission to de-anonymize. The manager cannot de-anonymize unless he generates a *publicly verifiable request of de-anonymization* for the guardians, declaring what value he wants to de-anonymize. This prevents the manager(s) from surreptitiously de-anonymizing. The guardians must agree to collaborate, but they do not learn the identity of the misbehaving user.

Threat Model and Security Guarantees. In our model, every party could act maliciously and collude. Specifically, the users, the manager, and **up to** t guardians can be fully malicious and can collude (for certain properties such as non-frameability, we even assume that all guardians are malicious). Within this threat model, we target the following security properties. **User anonymity:** Even if a malicious manager colludes with up to t malicious guardians, a message cannot be de-anonymized without a publicly verifiable request made by the manager. Moreover, even if all guardians are corrupted and collude, they still cannot de-anonymize any message when the manager is honest. **User non-frameability:** an honest user cannot be falsely accused of being the creator of a value v that she did not generate. This should be true even if *all* the parties are malicious and collude (except the party who enrolls the users, though this can be avoided whenever the real identity can be proven cryptographically, we discuss this below). **User accountability:** a manager can de-anonymize any value if enough guardians cooperate. **Manager accountability:** it should be infeasible for a manager to de-anonymize a value v without leaving a publicly verifiable trace in the system. This property is guaranteed if at most t guardians are malicious. **Manager non-frameability:** even if users and guardians are malicious and collude, they should not be able to fabricate a de-anonymization request for which the manager will be held accountable.

Flexibility. Decoupling the manager role from the monitoring role is crucial for allowing great flexibility in the implementation of the system. First, in our definition, guardians are only involved in handling de-anonymization requests and take no role in the functioning of the system. They are "platform independent" and hence one could even use the same set of guardians for multiple platforms managed by distinct managers. This is in contrast with previous proposals [12] where the parties performing the guardians' activity were also responsible for the functioning of the platform. Second, our definition identifies the set of guardians with a single public key and makes no assumption on the actual identity of the guardians. Specifically, the set of guardians can change over time – so long the same public key is maintained.

[2] We note also that we separate the role of a key issuer who lets users join the group. Such a key issuer is assumed to be honest in the scope of our work, as we focus on anonymity of users and the accountability of the manager. Further research could be done for malicious key issuers and the security issues they present.

1.3 MUTAL: Instantiations

A General Instantiation of MUTAL We provide a general instantiation of MUTAL based on a threshold public-key encryption scheme (TES)[3] and a simulation-extractable non-interactive zero-knowledge proof system. Assume that a set of n guardians has been chosen (we describe selection mechanisms in Sect. 1.4) and assume that a threshold of them is honest (up to t could be arbitrarily malicious). First, the guardians will engage in a (non-interactive) protocol to compute a public key for the threshold encryption scheme $\mathsf{pk}^{\mathcal{GU}}_{\mathsf{TE}}$. Next, assume that the manager of the digital platform has published her public key $\mathsf{pk}^{\mathcal{GM}}$ for a CPA-secure encryption scheme. At high-level, a MUTAL is instantiated as follows. A user U_i becomes a member of the platform by enrolling with the key issuer using her "real identity", along with a new, freshly picked key vk_i that the user will use to be identified as a member of the platform. Here the meaning of "real identity" depends on the application. For generality, we assume that there exists a procedure $\mathsf{Valid}(\mathrm{ID})$ that is applied to the real identity provided by the user. Once the ID is validated, the key issuer provides a signature σ_i on the pair $(\mathrm{ID}, \mathsf{vk}_i)$. The tuple $\mathsf{cert} = (\mathrm{ID}, \mathsf{vk}_i, \sigma_i)$ is then communicated to the manager of the platform. When the user generates a value v for the platform, she will send v along with an encryption of the identity $c_1 = \mathsf{Enc}_{\mathsf{pk}^{\mathcal{GM}}}(\mathsf{cert}_i)$ using the public key of the manager $\mathsf{pk}^{\mathcal{GM}}$ and a zero-knowledge proof of knowledge of the secret associated to vk_i and valid signatures σ_i computed by the key issuer. The size of the proof is *independent* of the number of authorized users enrolled in the digital service. To ensure *mutual* accountability, the ciphertext c_1 is wrapped inside another layer of encryption c_2, where c_2 was computed using the threshold encryption scheme under the public key of the guardians.

Thus, the final message posted by the user is $(\mathsf{m}, c_2, \mathsf{proof})$ where proof is a zero-knowledge proof that everything was computed correctly. We instantiate the TES with ElGamal Threshold Encryption Scheme [13,14], where the size of the public key is independent of the number of shares, n. Hence, the extra layer c_2 is succinct and independent of the number of guardians. If more than one set of guardians is available, the user can select the set of guardians that she trusts the most[4] and can add this to the tuple. By looking at the tuple $(\mathsf{m}, c_2, \mathsf{proof})$, the manager alone cannot learn c_1 (and thus decrypt the identity vk_i), without having the guardians remove the layer of threshold decryption. This is true even if the manager colludes with up to t guardians. To de-anonymize a message, the manager must provide a publicly verifiable request for de-anonymization that at least $t+1$ guardians accept. We note that even if all guardians are fully malicious and decrypt every single instance, they still cannot de-anonymize any user without the secret key of the manager. While this approach is natural and is outlined in previous work, they did not provide formal definitions or

[3] A threshold public-key encryption scheme is an encryption scheme where the secret key is split among n parties, and a cipher text can be decrypted only if at least t shares of the secret keys are used.

[4] For simplicity in this paper, we consider only one set of guardians.

proofs for accountability and non-frameability. We are the first to provide formal guarantees. The scheme is described in Sect. 5.1.

An Efficient Instantiation from Threshold Group Signature. Camenisch et al. [9] provide a practical t-out-of-n group signature scheme based on bilinear maps. Recall, in a group signature, a member of the group can sign anonymously within the group, but a group manager can de-anonymize any signature. In the scheme of Camenisch et al. [9], the manager is split into n parties, hence a signature can be de-anonymized if any subset of t managers agrees. This scheme does not directly fit our setting, where we want only the manager to de-anonymize and the guardians should only allow this action. To fit our setting, we change their t-out-of-n scheme so that any subset of t guardians will only be able to remove one layer from the group signature. The other layer can be removed solely by the manager, and no other party will learn the decryption. More discussion is provided in Sect. 5.2.

1.4 Monitoring Committee: Selection and Evolution

The suitability of a selection procedure for guardians depends on the application. We outline some possibilities below.

(a) Selection among the users (only trust in your peers). When no external party is trusted, the guardians can be elected among the users enrolled in the platform using cryptographic sortition techniques. This can be implemented using a Verifiable Random Function (VRF) [15]. When a user registers in the system, she will choose a public key for a VRF. Then in each "epoch" e, each user checks if she is elected as a *guardian* for the next epoch, by evaluating the VRF on input e, and check if the output is below a threshold ρ. This technique is used in Algorand [16] to select the committees that run the underlying consensus protocol. In our setting, we do not need a blockchain; we only need that the users of the digital platform have access to the public VRF keys of all users.

(b) Selection of external parties through voting mechanisms or by platform designers. When external parties that can be trusted exist, guardians could be selected through some voting mechanism among the platform designers. For instance, guardians can be chosen among nonprofit organizations that monitor citizen's rights in the US (such as the ACLU), etc. We stress that in our proof, we do not need all the guardians to be honest. We tolerate up to t completely malicious guardians.

(c) Selection through a public permissionless blockchain. Any blockchain that satisfies chain quality[5] can be used to select a committee of n guardians with the guarantee that at most t parties are malicious (with high probability), where the parameters n, t are tied to the chain quality parameter [11,17,18]. The idea is the following: people who wish to be part of the guardians try to add a block to the blockchain containing a transaction with a public key that they want to

[5] Chain quality α_l means that in any sequence of l consecutive block at least α fraction of them are added by honest parties.

use if they are selected. When enough blocks containing such transactions are stored on the blockchain, the public keys that appear in the first N blocks are automatically selected to be part of the guardians.

On Guardians' Incentives. The incentive for external parties to participate in MUTAL comes from the application. For example, if guardians are chosen among nonprofit organizations, their incentive for following the protocol follows from their social responsibility and reputation.

Evolving Committee. For more robustness of the system, the set of guardians changes periodically at epochs. To change the guardians, we propose to use a proactive secret-sharing mechanism for re-sharing the secret key sk among the new set of guardians using fresh shares which are independent than those of the old guardians. Proactive secret sharing techniques allow a secret to be *handed-off* between two sets of parties [10,19]. The procedure where we specifically use the dynamic proactive secret sharing (DPSS) scheme of Goyal et al. [11], is described in Sect. 5.3.

In [11] the hand off works as follows: To hand off the shares of the secret, the old and new guardians first perform an initial computation that allows them to hold two independent sharings of the same random value r. The old committee can use the sharing of s and the sharing of r to reconstruct the value $s - r$ and publish it. Since r is random, the value $s - r$ leaks no information. Next, each member of the new committee adds $s - r$ to their own share of r. As a result, each member holds shares of $s - r + r = s$, the original secret. The new set of shares is independent of the old set of shares. This completes the hand-off of the secret from the old committee to the new committee. It is assumed that the old committee erases the old sharing after this phase is complete. Else, an adversary could slowly eventually corrupt the old committee afterwards and learn the secret. To end up with different sharings of r, each member C_i of the new committee picks a random value r_i and creates two different sharings of it. Then C_i shares one of the shares with the old committee and the other with the new committee. Each party will then obtain a share for each r_i, it will sum their local shares and hold a sharing of $r = r_1 + \ldots + r_n$, which is guaranteed to be random if one party provided a random r_i. To ensure that no party misbehaves, a polynomial commitment scheme is used to guarantee that all shares are well-formed. For details see Sect. 5.3.

2 Related Work

Several works [4,8,12] have explored the concept of accountable anonymity, but lack formal definitions of accountability and thus provable guarantees and are suitable only to the communication layer. In such works, trusted mixers maintain the communication channel and are responsible for anonymization and de-anonymization.

More recently, Corrigan-Gibbs and Ford [20] targeted a closed group of people that self-manages the communications of its members and guarantees anonymity and some form of accountability. This work is specific for settings where the digital platform itself is decentralized, and it is not clear how one might extend it to other settings (e.g., where there is a platform manager).

Von Ahn et al. [2] note the threat of abuse of the de-anonymization power and proposes an anonymous and accountable system where the master secret key is not known by a single manager but is distributed (using some threshold schemes [21,22]) to a set of parties. This idea reduces the threat of abuse since the de-anonymization power is not concentrated in one entity. However, in [2], they provide only informal guarantees and do not discuss traceability or transparency of the de-anonymization process. Indeed, a later work by Danezis and Sassaman [23] highlights that it could be arbitrary which messages get to remain anonymous and which ones will be censured. More importantly, it is unclear under what circumstances the parties are *provably* accountable for their de-anonymization activities.

More recently, Camenisch et al. [9] proposed dynamic group signatures. These are anonymous signatures that can be de-anonymized by a set of designated parties non-interactively (providing some form of public traceability). This approach also lacks the generality provided by our framework. Also, this line of work simply distributes the platform manager among several parties. We want manager accountability to be independent of the platform manager and to be enforced by a crowd.

Frankle et al. [3] discuss accountability within the context of electronic surveillance of platforms such as Facebook. Here the goal is to track secret law enforcement requests to digital platforms. This work is tailored to this specific setting where all parties (i.e., Facebook, FBI, judges) are assumed to act in good faith. In particular in this setting users have no anonymity to begin with, with respect to group manager (i.e., Facebook). Another context in which accountability is needed is in anonymous decentralized transactions. Here, users wish to remain anonymous when making payments, but may need to be de-anonymized if they do something illegal (e.g., money laundering or illegal transactions). Spreading the trust and adding accountability for transactions has been explored over various works [24–27] but misses the more general setting of group signatures.

Libert and Joye [6], building on Sakai et al. [7], presented a group signature scheme with message-dependent opening. A dedicated committee, called the admitters, jointly decides if a message should be de-anonymized. If so, they jointly generate a *per-message* trapdoor that allows the manager to de-anonymize all instances that contain message m. As we mentioned earlier, there are significant differences with our approach. First, in [6] there is no focus on the traceability guarantees of the de-anonymization procedure. Thus, users can still be de-anonymized unknowingly. Second, the de-anonymization is message-dependent instead of instance-dependent. Third, manager non-frameability is not considered, suggesting that admitters and manager are the same authority, working towards the same goal, thus, admitters would not frame the manager. Like MUTAL, the signature has two layers of encryption, which must be removed by different entities. Message-dependent opening relies on *identity-based encryption* (IBE) scheme, specifically a fully collusion-resistant partially structure-preserving IBE, which is a variant of Waters' IBE scheme [28]. In IBE systems, a trusted party owns a master public and private key and private keys are signatures on the corresponding message [29]. A user asks for a private key from the key issuer, who can derive it from the master private key. Usually, the signature is on the user's identity [29], but here, the *messages* are the public key.

To sign, a user generates a two-level signature on their ID and the message they wish to sign m. Later, for de-anonymization, the guardians first remove the *message* layer and the authority removes the *identity* layer. The guardians generate a token t_m that depends on the message m. The fact that the admitters can generate a message dependent token inherently relies on using IBE. Upon receiving t_m, the authority can remove the other layer of encryption and then use t_m to decrypt the identity. The authority can then reuse the token t_m to de-anonymize *any* signatures on the message m. Deviating from our work, the presence of a trusted key issuer is crucial to their system. [6] only achieve full traceability, while we have *non-frameability*. In full traceability, the adversary is *passive*, meaning she only *receives* keys. Meanwhile, non-frameability allows the adversary to *make* her own keys. This means it is important that the key issuer not be able to learn the private group keys of each user. Libert and Joye just assume that the key issuance is done honestly and that the private key is *erased* after all the members join. The adversary only gets secret keys off the admitter/guardians and opener/authority.

3 Preliminaries

Notation. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We use $y \leftarrow \mathsf{F}(x)$ to say y is the output of a randomized algorithm F on input x and write $y \leftarrow \mathsf{F}(x; r)$ to explicitly refer to the randomness r used. We use $y := \mathsf{F}(x)$ if F is a deterministic algorithm. PPT stands for probabilistic polynomial time. A function $\mathsf{negl}(n)$ is *negligible* if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $\mathsf{negl}(n) \leq \frac{1}{p(n)}$. We denote the security parameter by λ.

We use $(z, (y_i)_{U_i \in S}) \leftarrow \mathsf{F}\langle U_i(x_i)\rangle_{U_i \in S}(w)$ to denote a protocol between parties in a set S. Here, each party holds a secret input x_i and receives secret output y_i, z is the public output and w is the public input.

Cryptographic Primitives. To instantiate our generic construction, we use standard cryptographic primitives such as a one-way function f, a secure signature scheme S, and a public key encryption scheme E. Their syntax and definitions can be found in any reference textbook (e.g., [30–32]). We also use non-interactive zero knowledge arguments of knowledge (NIZK) that satisfy the stronger notions of simulation extractability and succinctness (SNARK) [33, Def. 2.10]. Finally, we use a threshold encryption scheme TE which satisfies the property of simulatable decryption.

For the threshold dynamic group signature (DGS) based construction in 5.2, following [9], we use Pointcheval-Sanders signature scheme [34], various sigma protocols made non-interactive via the Fiat-Shamir transform [35], and a signature of knowledge [36].

4 Formal Definition of Mutual Accountability Layer

In this section, we present MUTAL. We identify the parties in the protocol below. In Definition 1 we describe the syntax of MUTAL. Finally, we introduce the security properties.

- **Users**: The parties that generate values on the platform. A user U can (1) join the platform using a valid identity ($\mathsf{JoinUser}^{\mathsf{Valid}}$) and (2) generate authorized value m anonymously ($\mathsf{MemberAuth}$).
- **Key issuer:** This party, denoted by $\mathcal{KI}$, checks the identity of users and registers them (i.e., helps to execute $\mathsf{JoinUser}^{\mathsf{Valid}}$).
- **Manager:** This party, denoted by $\mathcal{GM}$, can request de-anonymization for a message ($\mathsf{ReqDeanon}$). If the request is granted, it learns the identity of the message creator (Deanon). The manager's requests are publicly verifiable.
- **Guardians:** The set of parties that grants access to a de-anonymization. These parties, denoted by $\{\mathsf{C}_1, \dots, \mathsf{C}_n\}$, collectively protect the users against a potentially misbehaving group manager. They perform a one-time joint computation to compute a public key ($\mathsf{KeyGenGu}$). They then monitor the de-anonymization requests generated by $\mathcal{GM}$. Once a request associated with m is validated, the guardians perform a joint computation to generate a value that will allow the group manager to trace the identity of the user who created m ($\mathsf{GrantDeanon}$). The outputs provided by the guardians are publicly verifiable.

Definition 1 (Mutual Accountability Layer Syntax). *A Mutually Accountable Layer* MUTAL *consists of the following PPT procedures:*

1. $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$. *On input the security parameter λ, outputs parameters* pp *for the scheme. We assume* pp *implicitly contains the information about the message space $\mathcal{M}$, key space, etc.*
2. $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$. *On input the parameters* pp, *outputs a key pair* $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}})$ *for $\mathcal{KI}$. It also initializes a state* $\mathsf{st}^{\mathcal{KI}}$ *used to maintain information of the members that join a group.*
3. $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$. *On input* pp, *outputs a key pair for the group manager* $(\mathsf{sk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GM}})$ *and an initial state* $\mathsf{st}^{\mathcal{GM}}$ *used to manage the group.*
4. $\left(\mathsf{pk}^{\mathcal{GU}}, (\mathsf{sk}_l^{\mathcal{GU}})_{\mathsf{C}_l \in \mathcal{GU}}\right) \leftarrow \mathsf{KeyGenGu}\,\langle \mathsf{C}_l(\cdot)\rangle_{\mathsf{C}_l \in \mathcal{GU}}\,(\mathsf{pp})$. *On common input* pp, *the set of $\mathcal{GU} := \mathsf{C}_1, \dots, \mathsf{C}_n$ perform an interactive protocol. As output, each guardian gets their own secret key* $\mathsf{sk}_l^{\mathcal{GU}}$ *and all guardians get a public key* $\mathsf{pk}^{\mathcal{GU}}$. *We denote with* PK *the set public keys of the authorities, i.e.,* $\mathsf{PK} = \left\{\mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{GM}}\right\}$.
5. $(\mathsf{cert}_i, (\mathsf{sk}_i, \mathsf{st}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{GM}})) \leftarrow \mathsf{JoinUser}^{\mathsf{Valid}}\,\langle U_i\ (\mathrm{ID}_i), \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}), \mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})\rangle\,(\mathsf{pp}, \mathsf{PK})$. *An interactive protocol run between a user, $\mathcal{KI}$, and $\mathcal{GM}$. User participates with a public identity ID_i that can be validated according to a predicate* Valid. *$\mathcal{KI}$ and $\mathcal{GM}$ participate with their secret keys and their states. At the end of the protocol, the user gets a secret member*

key sk_i *for a member identity* ID_i*, the public output is a certificate* cert_i *that is added to key issuer and group manager's states.*

6. $\pi \leftarrow \mathsf{MemberAuth}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_i, \mathsf{sk}_i)$. U_i *executes* $\mathsf{MemberAuth}$ *to create an authorization for a message* $\mathsf{m} \in \mathcal{M}$*. On input* pp*,* PK*,* m*, secret key* sk_i*, and the associated certificate* cert_i*, it outputs a proof of membership* π *that proves his eligibility to produce* m*.*
7. $b \leftarrow \mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi)$*. On input* pp*,* PK*,* m*, and* π *outputs a bit* b *indicating whether the message is authorized.*
8. $\mathsf{req} \leftarrow \mathsf{ReqDeanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$*. On input* pp*,* PK*,* m*,* π*, and the secret key of the group manager* $\mathsf{sk}^{\mathcal{GM}}$ *produces a request* req *to de-anonymize the member who posted* m*.*
9. $b \leftarrow \mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req})$*. On input* pp*,* PK*,* m*,* π*, and a request* req*, it outputs a bit* b *indicating whether* $\mathcal{GM}$ *produced the request.*
10. $\perp/\mathsf{access} \leftarrow \mathsf{GrantDeanon}\langle \mathsf{C}_\ell(\mathsf{sk}_\ell^{\mathcal{GU}})\rangle_{\mathsf{C}_l \in \mathcal{GU}}$ $(\mathsf{pp},\mathsf{PK},\mathsf{req},\mathsf{m},\pi)$*. This is an interactive protocol between the guardians* $\mathcal{GU}$*. Guardian* C_l *has as secret input its secret key* $\mathsf{sk}_l^{\mathcal{GU}}$*, and all parties have common input* pp*,* PK*,* m*,* π*, and* req*. The result is a common output of either* access *or* $\perp$*.*
11. $\perp/(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) \leftarrow \mathsf{Deanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$*. On input* pp*,* PK*,* m*,* π*,* access*, and the secret key of the group manager* $\mathsf{sk}^{\mathcal{GM}}$*, outputs* $\mathsf{cert}_{\mathrm{ID}}$ *and a publicly verifiable proof* $\mathsf{proof}_{\mathrm{ID}}$ *that* $\mathsf{cert}_{\mathrm{ID}}$ *is the one accountable for* m*. Otherwise, it outputs* $\perp$*.*
12. $b \leftarrow \mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$ *On input* pp*,* PK*,* (m, π)*,* access*, and a pair* $(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$*; outputs a bit denoting whether the user assigned with* $\mathsf{cert}_{\mathrm{ID}}$ *is accountable for the pair* (m, π)*.*

Security Properties. Below, we discuss the security properties of MUTAL. We refer the reader to the full version for formal definitions [37, Section 4.1].

Unforgeability captures the property that anyone (even group manager and guardians) who is *not enrolled* in the system (i.e., has not executed protocol JoinUser) cannot produce valid membership authorizations. We capture this in a game where an adversary $\mathcal{A}$ controls the group manager and guardians and can access oracles to create new users (but not to *control* them) and to see honestly generated membership authorizations of her choice. The adversary wins the game if she can produce a pair (m, π) that verifies without controlling any user and without querying for an honestly generated authorization m.

User accountability guarantees that a pair (m, π) must trace back to *some* user when de-anonymized. $\mathcal{A}$'s goal is to create a pair (m, π) that does not correctly de-anonymize. $\mathcal{A}$ has control over malicious users and the guardians and can also ask for honest users' membership proofs, requests, and de-anonymizations.

Manager Non-Frameability guarantees that no one can accuse $\mathcal{GM}$ of creating a de-anonymization request, even if all guardians and users are malicious. The adversary can ask for requests/de-anonymizations and aims to craft a valid de-anonymization request.

Anonymity guarantees that no one can learn who the creator of a pair (m, π) is unless both the $\mathcal{GM}$ and the guardians collaborate. We consider two cases for anonymity: (1) $\mathcal{GM}$ and a minority of the guardians are corrupted, (2) all guardians are corrupted. In both cases, the adversary needs to distinguish membership authorizations from two honest signers of her choice.

Remark 1. Anonymity also implicitly covers a property of *manager accountability*: the manager cannot open messages on its own. It must ask for permission and must present a proof if it wants to blame a user for a message.

5 Instantiations

5.1 General Instantiation

The high-level description of the scheme was provided in Sect. 1. We present the protocol of our instantiation Π-MUTAL of MUTAL in Figs. 1, 2 and 3.

Zero-Knowledge Relation. The relation $\mathcal{R}$ for a statement $(\mathsf{m}, c_2, \mathsf{PK})$ is:

$$\mathcal{R} = \left\{ \begin{array}{c} ((\mathsf{m}, c_2, \mathsf{PK}), (r_2, c_1, r_1, \mathsf{cert}_i, \mathsf{sk}_i)) \text{ s.t. } \mathsf{PK} = ((\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}), (\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{vk}_\mathsf{S}^{\mathcal{GM}}), (\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}})) \\ \wedge \mathsf{cert}_i = (\mathsf{vk}_i, \mathrm{ID}_i, \sigma_i) \wedge \mathsf{S.Vrfy}_{\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i, \sigma_i) = 1 \\ \wedge c_1 = \mathsf{E.Enc}_{\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}}(\mathsf{cert}_i; r_1) \wedge \mathsf{vk}_i = f(\mathsf{sk}_i) \wedge c_2 = \mathsf{TE.Enc}_{\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}}}(\mathsf{m} \| c_1; r_2) \end{array} \right\}$$

We also use a nizk proof for the relation $\mathcal{VD} = \{(\mathsf{pk}, \mathsf{m}, c), (\mathsf{sk}) \text{ s.t. } \mathsf{m} = \mathsf{E.Dec}_\mathsf{sk}(c)\}$ for correct decryption.

Efficiency. When using ElGamal and Feldman secret sharing scheme for the threshold scheme, the guardians key generation takes $\mathcal{O}_\lambda(n)$ operations and communication per party -where n is the number of guardians- and granting deanonimization takes $\mathcal{O}_\lambda(1)$ per party. De-anonymizing and judging takes $\mathcal{O}_\lambda(t)$ time, where t is the threshold of the encryption scheme. Using [33] for the snark, a membership authorization takes $(\mathcal{O}_\lambda(|C| \log |C|)$ time, where C is the circuit realizing $\mathcal{R}$; its size is $\mathcal{O}_\lambda(1)$. All other operations take $\mathcal{O}_\lambda(1)$ time.

Security. We sketch the security proofs below. For formal proofs, we refer the reader to the full version [37, Section 4.1 and Appendix B].

Unforgeability holds because the adversary (1) cannot construct a pair (m, π) that verifies without a valid witness (due to simulation extractability of the snark), (2) cannot invert one of the vk in the pool of all honest certificates to find a witness (due to f being one-way) and (3) cannot forge a signature σ on a public key vk to join without the key issuer (due unforgeability of the signature scheme). User non-frameability holds as the adversary (1) cannot win unforgeability, (2) cannot produce a valid proof $\mathsf{proof}_{\mathrm{ID}}$ for the relation $\mathcal{VD}$ without a witness and (3) cannot provide two different $\mathsf{cert}, \mathsf{cert}_{\mathrm{ID}}$ that are part of a valid witness for $\mathcal{R}$ and $\mathcal{VD}$, respectively (due to soundness of the threshold scheme, namely the fact that decryptions are unique).

User accountability holds for the same reasons as unforgeability. The adversary cannot forge a signature on the key of $\mathcal{KI}$ or create a proof of membership

```
$\underline{\mathsf{SetupParams}}(1^\lambda)$
• $\mathsf{pp_E} \leftarrow \mathsf{E.Setup}(1^\lambda)$, $\mathsf{pp_S} \leftarrow \mathsf{S.Setup}(1^\lambda)$; $\mathsf{pp_{TE}} \leftarrow \mathsf{TE.Setup}(1^\lambda)$
• $(\mathsf{crs}_\mathcal{R}, \tau_\mathcal{R}) \leftarrow \mathsf{snark}_\mathcal{R}.\mathsf{Setup}(1^\lambda)$
• $(\mathsf{crs}_\mathcal{VD}, \tau_\mathcal{VD}) \leftarrow \mathsf{NIZK}_\mathcal{VD}.\mathsf{Setup}(1^\lambda)$
• Output $\mathsf{pp} := (\mathsf{pp_E}, \mathsf{pp_S}, \mathsf{pp_{TE}}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_\mathcal{VD})$
$\underline{\mathsf{KeyGenIssuer}}(\mathsf{pp})$
• $(\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{sk}_\mathsf{S}^{\mathcal{KI}}) \leftarrow \mathsf{S.KeyGen}(\mathsf{pp_S})$.
• Initialize an empty list $\mathsf{Members}$
• Output $(\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{sk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{Members})$
$\underline{\mathsf{KeyGenManager}}(\mathsf{pp})$
• $(\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{E}^{\mathcal{GM}}) \leftarrow \mathsf{E.KeyGen}(\mathsf{pp_E})$;
  $(\mathsf{vk}_\mathsf{S}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{S}^{\mathcal{GM}}) \leftarrow \mathsf{S.KeyGen}(\mathsf{pp_S})$
• Initialize an empty list $\mathsf{Members}$
• Set $\mathsf{pk}^{\mathcal{GM}} := (\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{vk}_\mathsf{S}^{\mathcal{GM}})$; $\mathsf{sk}^{\mathcal{GM}} := (\mathsf{sk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{S}^{\mathcal{GM}})$; $\mathsf{st}^{\mathcal{GM}} = \mathsf{Members}$
• Output $(\mathsf{sk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})$
$\underline{\mathsf{KeyGenGu}}\ \langle \mathsf{C}_\ell(\cdot) \rangle_{\mathsf{C}_l \in \mathcal{GU}}\ (\mathsf{pp})$
• Parties execute $\big(\mathsf{pk_{TE}}, (\mathsf{sk}^l_\mathsf{TE})_{\mathsf{C}_\ell \in \mathcal{GU}}\big) \leftarrow \mathsf{TE.KeyGen}\big\langle \mathsf{C}_\ell(\cdot) \big\rangle_{\mathsf{C}_\ell \in \mathcal{GU}}\ (\mathsf{pp_{TE}})$
• $\mathsf{pk}^{\mathcal{GU}} \leftarrow \mathsf{pk_{TE}}$ and $\mathsf{sk}_l^{\mathcal{GU}} \leftarrow \mathsf{sk}^l_\mathsf{TE}$
$\underline{\mathsf{JoinUser}}(\langle U_i(\mathrm{ID}_i), \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}), \mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \rangle(\mathsf{pp}, \mathsf{PK})$
• $U_i$: Samples $\mathsf{sk}_i \leftarrow \mathcal{SK}$ and sets $\mathsf{vk}_i := f(\mathsf{sk}_i)$
• $U_i \rightarrow \mathcal{KI}$: $\mathrm{ID}_i, \mathsf{vk}_i$
• $\mathcal{KI}$: if $\mathsf{Valid}(\mathrm{ID}_i) = 1 \wedge (\mathrm{ID}_i, \cdot, \cdot) \notin \mathsf{Members}$ : $\sigma_i \leftarrow \mathsf{S.Sign}_{\mathsf{sk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i)$. Else:
  Output $\perp$ and halt.
• $\mathcal{KI} \rightarrow U_i$: $\sigma_i$
• $U_i$: If $\mathsf{S.Vrfy}_{\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i, \sigma_i) = 1$: $\mathsf{cert}_i := (\mathrm{ID}_i, \mathsf{vk}_i, \sigma_i)$. Else: Output $\perp$ and
  halt.
• $\mathcal{KI}$: $\mathsf{Members} := \mathsf{Members} \cup \mathsf{cert}_i$ (Send to $\mathcal{GM}$)
• $U_i$: Output $(\mathsf{cert}_i, \mathsf{sk}_i)$
• $\mathcal{KI}$: Output $\mathsf{cert}_i$
```

Fig. 1. Protocol Π-MUTAL: Parameter setup and Key Generation Protocols and Algorithms.

```
$\underline{\mathsf{MemberAuth}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_i, \mathsf{sk}_i)$
• $c_1 \leftarrow \mathsf{E.Enc}_{\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}}(\mathsf{cert}_i; r_1)$;  $c_2 \leftarrow \mathsf{TE.Enc}_{\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}}}(\mathsf{m} || c_1; r_2)$
• $\mathsf{proof} \leftarrow \mathsf{snark}_\mathcal{R}.\mathsf{Prove}(\mathsf{crs}, (\mathsf{m}, c_2, \mathsf{PK}), (r_2, c_1, r_1, \mathsf{cert}_i, \mathsf{sk}_i))$
• Output $\pi := (c_2, \mathsf{proof})$
$\underline{\mathsf{AuthVrfy}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi)$
• Parse $\pi = (c_2, \mathsf{proof})$
• Output $b \leftarrow \mathsf{snark}_\mathcal{R}.\mathsf{Vrfy}(\mathsf{crs}, (\mathsf{m}, c_2, \mathsf{PK}), \mathsf{proof})$
```

Fig. 2. Protocol Π-MUTAL: Anonymous Membership Proof Generation and Verification Algorithms

```
$\underline{\mathsf{ReqDeanon}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$
• Output $\mathsf{req} \leftarrow \mathsf{S.Sign}_{\mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m}\|\pi)$
$\underline{\mathsf{JudgeReq}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req})$
• Output $b \leftarrow \mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m}\|\pi, \mathsf{req})$
$\underline{\mathsf{GrantDeanon}} \left\langle \mathsf{C}_1(\mathsf{sk}_1^{\mathcal{GU}}), \ldots, \mathsf{C}_n(\mathsf{sk}_n^{\mathcal{GU}}) \right\rangle (\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$
Upon receiving request $(\mathsf{req}, \mathsf{m}, \pi)$ each party $\mathsf{C}_l$ in $\mathcal{GU}$ proceeds as follow:
• If $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 0 \vee \mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 0$ it aborts.
• Participate in protocol $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m}\|c_1) \leftarrow \mathsf{TE.ProveValidDec} \left\langle \mathsf{C}_\ell(\mathsf{sk}_{\mathsf{TE}}^{\ell}) \right\rangle_{\ell=1}^{n}(c_2)$
  and output $\mathsf{access} := (\mathsf{proof}_{\mathsf{m}}, \mathsf{m}\|c_1)$
$\underline{\mathsf{Deanon}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$.
• Parse $\mathsf{access}$ as $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m}\|c_1)$. If $\mathsf{TE.ValidDec}(\mathsf{m}\|c_1, c_2, \mathsf{proof}_m) = 0$, output $\perp$
• $\mathsf{cert}_{\mathrm{ID}} \leftarrow \mathsf{E.Dec}_{\mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}}}(c_1)$. Parse $\mathsf{cert}_{\mathrm{ID}} = (\mathrm{ID}, \mathsf{vk}, \sigma)$. If $\mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}}(\mathrm{ID}\|\mathsf{vk}, \sigma) = 0$,
  output $\perp$
• $\mathsf{proof}_{\mathrm{ID}} \leftarrow \mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Prove}\ (\mathsf{crs}_{\mathcal{VD}}, (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}), \mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}})$
• Output $(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$
$\underline{\mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})}$
• Parse $\mathsf{access}$ as $(\mathsf{proof}_m, \mathsf{m}\|c_1)$ and $\mathsf{cert}_{\mathrm{ID}} = (\mathrm{ID}, \mathsf{vk}, \sigma)$
• If $\mathsf{TE.ValidDec}(\mathsf{m}\|c_1, c_2, \mathsf{proof}_m) = 0$ or
  $\mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Vrfy}(\mathsf{crs}_{\mathcal{VD}}, (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}), \mathsf{proof}_{\mathrm{ID}}) = 0$ output $\perp$
• Output $b \leftarrow \mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}}(\mathrm{ID}\|\mathsf{vk}, \sigma)$
```

Fig. 3. Protocol Π-MUTAL: De-anonymization Algorithms.

without holding any valid certificate. Manager non-frameability holds directly for the unforgeability of the Signature Scheme. Finally, in both cases of anonymity an adversary must distinguish between membership permissions generated by different users. For both proofs, we replace real memberships by simulated ones. Then, we can extract a witness and do a reduction to indistinguishability of encryption either in the plain, or the threshold setting.

5.2 Instantiation Based on *t*-out-of-*n* Group Signatures (Camenisch et al. [9])

We show how the construction of Camenisch et al. [9] can be used in the MUTAL framework. We present a high level overview but due to lack of space we refer the reader to the full version [37, Section 6] for a more detailed exposition. The protocol appears in Appendix A of the full version. Algorithms ReqDeanon and JudgeReq remain the same as in the general instantiation (Fig. 3).

Camenisch et al. build a t-out-of-n group signature scheme based on the Pointcheval Sanders (PS) signature scheme [34] and zero-knowledge signatures of knowledge (SoK) [36]. A SoK allows a party to prove that they know a witness to an NP-statement to sign a message, and PS signatures' reliance on bilinear pairings make it easy to create said SoK.

To conveniently describe the steps of the group signature scheme we introduce some notation. In a nutshell, PS signatures rely on bilinear pairings, described by

$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The secret key is $x, y_0, y_1 \leftarrow \mathbb{Z}_q$ and $h \leftarrow \mathbb{G}_2$. The public key is $X := h^x, Y_0 := h^{y_0}, Y_1 := h^{y_1}$. A PS signature is a tuple (a, σ_1, σ_2) where $a \in \mathbb{Z}_p$, σ_1 is a group element, and σ_2 is the exponentiation of the group element to a linear combination of a and the message m.

To join the group, a user U generates its secret key sk and asks the key issuer to blindly sign sk. The key issuer generates a PS signature on sk. When U wants to sign a message and prove that it belongs to the group, it will generate a SoK of a valid message/PS signature pair under the key issuer's verification key. This signature is $(a, \sigma_1', \sigma_2')$. A *function* of the user's secret key is secret shared with the committee and later allows for de-anonymization in the following way: when this value is reconstructed, it allows to check whether the SoK corresponds to the secret value associated with a user. Thus, the managers can check against all user and identify the signer.

Specifically, the user secret-shares the value Y_0^{sk} with the committee where $Y_0 = h^{y_0}$ is part of the PS signature public key as described earlier. The de-anonymization procedure requires the committee to compute the values $e(\sigma_1', Y_0^{\mathsf{sk}_i})$ and perform the test of the verification equation against a PS signature $(a, \sigma_1', \sigma_2')$. The crucial property that is satisfied is that this equation holds if and only if sk_i was used to produce the SoK (except with negligible probability). Verifiable secret sharing techniques and sigma protocols are used to perform the above actions verifiably, both from the side of the users as well as the openers.

To modify this system for MUTAL, we need to show how to separate the guardians from a manager. Basically, in the joining procedure, U will sample sk as before and then create $\mathsf{sk}_1, \mathsf{sk}_2$ such that $\mathsf{sk} = \mathsf{sk}_1 + \mathsf{sk}_2$. As before, it will share $Y_0^{\mathsf{sk}_1}$ to the guardians, who naturally take the role of the openers. sk_2 will be shared only with the manager. Because of this, any $t + 1$ of the guardians can recover sk_1 but the manager *must* collaborate to recover Y_0^{sk} and to de-anonymize. Analogously, the manager and fewer than t of the guardians cannot de-anonymize users.

As for efficiency, the most attractive feature is signature size and computation time. Because very efficient sigma protocols are used, these costs are minimal: [9] reports 232 bytes per signature on a Cocks-Pinch curve and 8 (resp. 12) group operations to sign (resp. verify). To achieve that though, de-anonymization becomes much slower. Since a signature is checked against each user it is linear in the number of users. However, this can be acceptable in most application scenarios since normally, producing membership proofs happens often while de-anonymization is scenario that should happen rarely.

5.3 Evolving Committees

The security properties introduced in this work require the participation of the guardians. However, in real life such a committee of guardians may only be available short-term and may also become corrupted over time. For this reason, we allow the committee to change via the dynamic proactive secret sharing scheme (DPSS) by Goyal et al. [11]. DPSS allows a set of n parties, who hold n shares of a secret, to hand-off the secret to another set of n parties. The n shares of the secret in our setting are those corresponding to the secret key associated to $\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}$. The public and secret key of the guardians remain the same, but the shares of the secret key are updated. Therefore, users always sign their messages with the same public key and the changing of committee does not affect them.

The DPSS protocol in [11] consists of *setup*, *hand-off*, and *reconstruction* phases. In our instantiations (Sects. 5.1, 5.2) the three phases for DPSS are as follows: (1) In the *setup*, an initial committee produces ElGamal distributed keys. We denote the shared secret s and the public encryption key g^s; (2) in the *hand-off* the old committee transfers the secret to the new committee, and we present this phase in Fig. 4; (3) rather than reconstruction, in our setting, the new committee work together to grant de-anonymizations. The current set of guardians removes the outer layer of decryption using the shared secret key s in a standard execution of verifiable decryption of ElGamal.

When instantiating our framework with the signature scheme by Camenisch et al. we have that each user generates a secret key and distributes its shares with the guardians (and manager). Namely, the guardians store one secret share for each user. Still, the protocol of Goyal et al. can easily be adapted. Parties in $\mathcal{GU}^{(i+1)}$ create two independent secrets $(r_i, \tilde{r}_i)$ for each user and batch them and the shares $([r_i]_t, [\tilde{r}_i]_t)$ through a random linear combination that is also used by the parties of $\mathcal{GU}^{(i)}$ with $\{[s_i]_t\}$ to reconstruct $\{[s_i - r_i]_t\}$.

Security Overview. The system tolerates adaptive corruptions as soon as the adversary is not able to corrupt $t+1$ parties in one committee, due to the fact that the secret is continuously updated. Note that after the hand-off phase, guardians from the old committee have the instruction to delete their shares and thus having the adversary corrupting $t+1$ guardians from any previous committee can happen with only a small probability. An adversary capable of corrupting at most t parties from any committee, learns nothing about the secret (*secrecy*), nor can prevent parties from reconstructing the secret (*robustness*).

Hand-off $\left\langle \mathsf{C}_\ell^{(i)}(\ell, s_{i,\ell}), \mathsf{C}_\ell^{(i+1)}(\ell) \right\rangle_{\mathsf{C}_\ell^{(i)} \in \mathcal{GU}^{(i)}, \mathsf{C}_\ell^{(i+1)} \in \mathcal{GU}^{(i+1)}}$ (pk).

- $\mathsf{C}_\ell^{(i)}$ holds a secret value-commitment pair $s_{i,\ell} \in \mathbb{Z}_p$, $g^{s_{i,\ell}} \in \mathbb{G}$. $[g^s]_t$ is public
- Parties of $\mathcal{GU}^{(i+1)}$ verifiably create two independent secret shares of a random value r, $[r]_t, [\tilde{r}]_t$, as explained in [11, Sec. 4]. They publish $[g^r]_t, [g^{\tilde{r}}]_t$,
- Parties of $\mathcal{GU}^{(i)}$ reconstruct the secret $[s-r]_t = [s]_t - [r]_t$ and announce it.
- $\mathsf{C}_\ell^{(i+1)}$, holding $\tilde{r}_\ell$, computes $s_{i+1,\ell} = \tilde{r}_\ell + s - r$. Since $\tilde{r} = r$, the values $s_{i+1,\ell}$ constitute a (different) share of s, that is $[\tilde{s}]_t = (s_{i+1,1}, \ldots, s_{i+1,n})$ for $s = \tilde{s}$.
- All parties update the public values $[g^{\tilde{s}}]_t = [g^{\tilde{r}}]_t \cdot g^{s-r}$.
- Parties of $\mathcal{GU}^{(i)}$ erase their local state, namely, the shares of $[s]_t$.

Fig. 4. Hand-off phase.

To achieve security in [11], the polynomial commitment scheme of Kate et al. [38] can be used to ensure that the values shared by each party are consistent, namely, that party ℓ receives the share $p(\ell)$ for a committed polynomial p of degree t. We refer the reader to [11, Sect. 4, 5] for the details.

6 Conclusion

Our paper has the first formal definitions balancing accountability and anonymity in digital platforms. Our main contribution was to show these definitions and we then used prior work to show how such a scheme that satisfies these definitions can be implemented.

References

1. Security through encryption and security despite encryption - council resolution on encryption (2020). https://data.consilium.europa.eu/doc/document/ST-13084-2020-REV-1/en/pdf. Accessed 20 Sept 2021
2. von Ahn, L., Bortz, A., Hopper, N.J., O'Neill, K.: Selectively traceable anonymity. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 208–222. Springer, Heidelberg (2006). https://doi.org/10.1007/11957454_12
3. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.: Practical accountability of secret processes. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018, pp. 657–674, Baltimore, MD, USA, 15–17 August 2018. USENIX Association (2018)
4. Köpsell, S., Wendolsky, R., Federrath, H.: Revocable anonymity. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 206–220. Springer, Heidelberg (2006). https://doi.org/10.1007/11766155_15
5. Libert, B., Mouhartem, F., Nguyen, K.: A lattice-based group signature scheme with message-dependent opening. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 137–155. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_8
6. Libert, B., Joye, M.: Group signatures with message-dependent opening in the standard model. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 286–306. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_15
7. Sakai, Y., Emura, K., Hanaoka, G., Kawai, Y., Matsuda, T., Omote, K.: Group signatures with message-dependent opening. In: Abdalla, M., Lange, T. (eds.) Pairing 2012. LNCS, vol. 7708, pp. 270–294. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36334-4_18
8. Claessens, J., Diaz, C., Goemans, C., Dumortier, J., Preneel, B., Vandewalle, J.: Revocable anonymous access to the internet? Internet Research (2003)
9. Camenisch, J., Drijvers, M., Lehmann, A., Neven, G., Towa, P.: Short threshold dynamic group signatures. In: Galdi, C., Kolesnikov, V. (eds.) SCN 2020. LNCS, vol. 12238, pp. 401–423. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57990-6_20
10. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_27

11. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504 (2020). https://eprint.iacr.org/2020/504
12. Diaz, C., Preneel, B.: Accountable anonymous communication. In: Petkovic, M., Jonker, W. (eds.) Security, Privacy, and Trust in Modern Data Management, pp. 239–253. Springer, Cham (2007). https://doi.org/10.1007/978-3-540-69861-6_16
13. Boneh, D., et al.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 565–596. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_19
14. Boneh, D., Gennaro, R., Goldfeder, S., Kim, S.: A lattice-based universal thresholdizer for cryptographic systems. Cryptology ePrint Archive, Report 2017/251 (2017). http://eprint.iacr.org/2017/251
15. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th FOCS, pp. 120–130, New York, NY, USA, 17–19 October 1999. IEEE Computer Society Press (1999)
16. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454 (2017). http://eprint.iacr.org/2017/454
17. Pass, R., Shi, E.: Hybrid consensus: efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
18. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 260–290. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_10
19. Schultz, D.A., Liskov, B., Liskov, M.: Mobile proactive secret sharing. In: Bazzi, R.A., Patt-Shamir, B. (eds.) 27th ACM PODC, p. 458, Toronto, Ontario, Canada, 18–21 August 2008. ACM (2008)
20. Corrigan-Gibbs, H., Ford, B.: Dissent: accountable anonymous group messaging. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V., (eds.) ACM CCS 2010, pp. 340–350, Chicago, Illinois, USA, 4–8 October 2010. ACM Press (2010)
21. 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
22. Shamir, A.: How to share a secret. Commun. Associat. Comput. Mach. **22**(11), 612–613 (1979)
23. Danezis, G., Sassaman, L.: How to bypass two anonymity revocation schemes. In: Borisov, N., Goldberg, I. (eds.) PETS 2008. LNCS, vol. 5134, pp. 187–201. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70630-4_12
24. Gupta, S., Lauppe, P., Ravishankar, S.: A blockchain-backed central bank cryptocurrency. Dept. of Computer ScienceYale University (2017)
25. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_5
26. Wüst, K., Kostiainen, K., Čapkun, V., Čapkun, S.: PRCash: fast, private and regulated transactions for digital currencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 158–178. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_11

27. Puzis, R., Barshap, G., Zilberman, P., Leiba, O.: Controllable privacy preserving blockchain. In: Dolev, S., Hendler, D., Lodha, S., Yung, M. (eds.) CSCML 2019. LNCS, vol. 11527, pp. 178–197. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20951-3_16
28. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_7
29. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
30. Goldreich, O.: Foundations of Cryptography: Basic Tools, vol. 1. Cambridge University Press, Cambridge (2001)
31. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
32. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. 2nd edn., CRC Press, Boca Raton (2014)
33. Groth, J., Maller, M.: Snarky signatures: minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 581–612. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_20
34. Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 111–126. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_7
35. 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
36. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_5
37. Daza, V., Haque, A., Scafuro, A., Zacharakis, A., Zapico, A.: Mutual accountability layer: accountable anonymity within accountable trust. Cryptology ePrint Archive, Report 2021/596. https://eprint.iacr.org/2021/596
38. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11

Faster Post-Quantum TLS Handshakes Without Intermediate CA Certificates

Panos Kampanakis[1](✉) and Michael Kallitsis[2]

[1] AWS Cryptography, Seattle, USA
kpanos@amazon.com
[2] Merit Network Inc., Ann Arbor, USA
mgkallit@merit.edu

Abstract. Traditionally, the most data-heavy part of a (D)TLS handshake has been authentication which includes a handshake signature and digital certificates. Although most common (D)TLS usecases are not significantly affected, some constrained ones such as low bandwidth environments or delay sensitive applications can see drastic performance degradation due to big certificates or certificate chains. That has led the security community to seek options to alleviate the issue. Post-quantum signatures and keys, on the other hand, have been proven to noticeably slow down handshakes even for common Internet (D)TLS or QUIC applications due to the significantly higher amounts of post-quantum authentication data they include. In this work, we quantify the size issue of post-quantum certificates in (D)TLS and QUIC and make the case for speeding up (D)TLS and QUIC handshakes by omitting the intermediate certificate authority certificates in the handshake. We present how that can be achieved along with the usecases that will mostly benefit from such a mechanism. We offer quantitative analyses to show that this approach is relatively straightforward, backwards compatible and with little overhead introduced for caching the certificates. We also discuss caching mechanisms based on different optimization goals.

Keywords: Post-quantum TLS · PQ authentication · Post-quantum certificate chains · ICA suppression

1 Introduction

Digital communications have completely penetrated everyday life as enablers of numerous critical services including telemedicine, online banking, massive e-commerce, machine-to-machine automation, mobile and cloud computing. To ensure that it is secure, information is exchanged over secure tunnels which guarantee confidentiality and authenticity. Secure tunnel protocols (e.g. (D)TLS, QUIC, SSH) use cryptography to encrypt the data and Public Key Infrastructure (PKI) certificates to authenticate the communicating peers.

A PKI infrastructure consists of various parts. A Certificate Authority (CA) issues an entity's X.509 certificate [13] which assures the entity's identity and the public key (PK) tied to that identity. The identity is included in the Subject field of the certificate, while the entity's public key is stored in the Subject Public Key Information along with the algorithm used by the issuer to create the signature. A certificate contains a specific validity period and extensions included by CAs to enable additional functionality. The certificate is signed by the CA's private key using the specified signature algorithm and the signature is added to the certificate's Signature field. The two most popular digital signature algorithms used in certificates today are the Elliptic Curve Digital Signature (ECDSA) and Rivest-Shamir-Adleman (RSA).

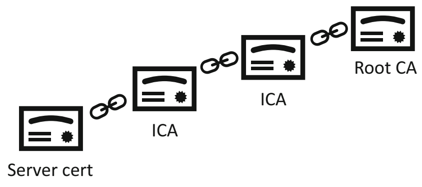


Fig. 1. Certificate chain

At the top of the PKI, there are trusted CAs that self-sign their own certificates known as Root CA certificates. Normally a Root CA issues certificates for Intermediate CAs (ICAs). An ICA can further issue certificates for other ICAs that in turn sign leaf/entity certificates in the PKI. This process results in the creation of certificate chains of trust (Fig. 1) that usually consist of two to four certificates but can be arbitrarily long.

Widely used protocols like (D)TLS [7,30,32,34], IKEv2/IPsec and QUIC [14,48] leverage X.509 certificates and certificate chains rooted to a pre-trusted Root CA to authenticate a peer's identity and public key (PK). After exchanging ephemeral keys, the server (and optionally the client) sends its certificate chain to its peer along with a signature of the connection transcript which proves that it owns the identity private key corresponding to its identity certificate. The server's certificate is authenticated by verifying the certificate chain to a pre-trusted Root CA certificate. Given that certificate chains can be of arbitrary size, sometimes they could affect connection performance which we aim to alleviate in this work.

The **key contributions of our work** are summarized as follows:

(i) We quantify the heavy post-quantum authentication data issue in (D)TLS.
(ii) We quantify the amplification protection issue post-quantum algorithms introduce in QUIC.
(iii) We analyze public server datasets in order to quantify the number of ICAs used in (D)TLS connections today. We identify the ICAs and the types of certificates sent in the most popular servers' certificate chains. We analyze the numbers of distinct ICAs as the Top number of servers increases. We quantify the size of the ICA cache that someone would need to store ICAs in. Using these analyses we show that caching ICAs would speed up the connection handshakes without overloading most software applications used today.
(iv) We qualitative evaluate options of trimming down the authentication data in a handshake and propose ICA suppression, the most straightforward, backwards-compatible one.

(v) We propose ICA caching mechanisms which would allow entities to request ICA suppression.

(vi) We discuss considerations and security implications of ICA suppression and how they could be overcome.

The rest of the paper is organized as follows: Sect. 2 describes the issue with heavy authentication data in secure connection handshakes and presents usecases where it is more prevalent. Section 3 summarizes related work. Section 4 presents statistics of commonly used server datasets in order to quantify the number of ICAs an entity would need to store in order to implement ICA suppression. Section 5 describes options to alleviate the heavy authentication-data issue. Section 6 discusses the ICA caching mechanisms which could be used to alleviate (D)TLS handshakes. Section 7 goes over considerations and concerns with ICA suppression and potential protection mechanisms. Section 8 concludes this work and discusses future research topics.

2 The Heavy Authentication Data Issue

Although not the case for common Internet uses today, some important wireless sensor network protocols in the Internet-of-Things (IoT) ecosystem such as constrained mesh networks like Wi-SUN and IEEE 802.14.5 [23] usually operate in low-speed mediums. They often depend on EAP-TLS [43] for authentication. Using big certificate chains in such usecases could further exacerbate the low-speed issue and lead to authentications that fail or take minutes or hours [15,21,39].

What is more, recent developments in post-quantum cryptography are expected to lead to increases in the size of certificates which could slow down secure connection establishment even in typical Internet connections or applications. The cryptographic community has been researching quantum-secure algorithms for some time in order to address the quantum computer threat, and the US National Institute of Standards and Technology (NIST) has an ongoing, public project to standardize quantum-resistant public key algorithms. At the time of this writing, NIST's evaluation process is in Round 3 with 15 post-quantum (PQ) algorithm candidates remaining. Additionally, a few Internet Engineering Task Force (IETF) RFC drafts are already introducing these algorithms in IETF standards [8–11,22,28,45,49]. When it comes to PQ signatures, the public key and signature sizes of NIST PQ signature candidates start from a few and can go up to tens of Kilobytes (KB) for the heavier schemes. The integration of such PQ signatures in X.509 certificates will naturally increase the size of these certificates significantly [17]. Certificate chains of higher sizes could exceed any certificate chain that our applications see today. That could mean many more packets which increases the loss probability in constrained conditions [29]. It also could lead to more round-trips due to TCP Congestion Control [41,42] and, thus, connection establishment slowdowns.

Specifically (D)TLS includes multiple signatures in its handshake which could fluctuate based on usecase. All connections include a signature in the

`Certifica-teVerify` message and public keys with signatures in each certificate in the chain. According to Shodan [40], ∼77% of TLS connections include certificate chains with one or two ICA certificates, which usually do not exceed 4KB. X.509 leaf certificates used in the Web (HTTPS), on the other hand, usually include two or more additional Signed Certificate Timestamps (SCTs) [20] which incorporate one signature each. Recently, browsers have increased their minimum SCT requirement. For example, as of April 2021, Apple's Certificate Transparency policy [2] requires three SCTs if the certificate lifetime is longer than 180 days. Similarly, Chrome has been requiring at least two SCTs or more depending on certificate lifetime. If SCTs are not included in the certificate, they can optionally be included in the handshake in a TLS Extension. Furthermore, when Online Certificate Status Protocol (OCSP) [35] stapling is used in TLS, one more OCSP signature may be included in the handshake to verify certificates are not revoked. Thus, it is clear that (D)TLS can include $(x + 2) + s + o$ signatures and $x+1$ public keys, where x is the number of ICAs in the certificate chain, s is the number of SCTs and $o = 1$ only if OCSP stapling is used.

To quantify the minimum authentication data size of PQ certificate authentication in (D)TLS, we calculated them for the leanest PQ signature candidates in NIST's Round 3. Lattice-based Dilithium and Falcon offer the smallest public key and signature sizes. Rainbow is the third PQ signature finalist and SPHINCS$^+$ is the most well-analyzed and trusted algorithm in terms of cryptographic primitives. We analyzed authentication data sizes for all the parameters of Dilithium and Falcon, and the leanest ones for Rainbow and SPHINCS$^+$. We assumed 500KB of X.509 attributes in each certificate. In terms of certificate formats, we assumed binary DER encoding [12] which is used to transfer certificates on the wire.

Table 1 includes our results. We can observe that only Falcon is consistently in the 4-8 KB range for 1–4 ICAs which is in the range of the sizes we see today. Dilithium offers less flexibility and remains below ∼14.5 KB, the most commonly used TCP `initcwnd` used today (10MSS), only for its Dilithium-2 parameter set for two or more ICAs in the chain. When SCTs and/or OCSP staples are present Dilithium starts from ∼15 KB. Note that even below ∼14 KB, the more data included in a handshake the higher the loss probability and the connection slowdown in lossy environments [29]. Based on [51], we would like to minimize this data to 9-10 KB (in green), not just below ∼14.5 KB. All other post-quantum signature algorithms are shown to exceed 15-20 KB which admittedly is a higher price to pay in order to establish a connection that otherwise exchanges much less data.

QUIC [14,48], on the other hand, is a protocol which was built with speed and performance in mind. It runs over UDP and uses TLS 1.3 for its secure tunnel negotiation. QUIC includes an amplification protection mechanism according to which initial QUIC client data has to be padded to at least 1200B and the server response should not exceed three times the initial client request size. If it does, the server has to wait for a response from the client (one round-trip) before sending more data. Amplification protection is used only for addresses which have not yet been validated.

Table 1. Approximate size of auth data (`CertificateVerify` + `Certificates`) in TLS handshake (KB)

	1 ICA	2 ICAs	3 ICAs	4 ICAs	PQ Signature
TLS (no SCTs, no OCSP staples)	8.77	11.94	15.12	18.29	Dilithium-2
	11.91	16.22	20.53	24.84	Dilithium-3
	3.76	5.31	6.86	8.40	Falcon-512
	6.64	9.32	12.00	14.68	Falcon-1024
	89.12	133.64	178.16	222.69	Rainbow-I(cz)
	20.44	26.73	33.02	39.31	SPHINCS$^+$-128s
Web TLS (SCTs, no OCSP staples)	12.40	15.57	18.75	21.92	Dilithium-2
	16.85	21.16	25.47	29.78	Dilithium-3
	4.76	6.31	7.85	9.40	Falcon-512
	8.56	11.24	13.92	16.60	Falcon-1024
	89.21	133.74	178.26	222.79	Rainbow-I(cz)
	32.22	38.51	44.80	51.10	SPHINCS$^+$-128s
Web TLS (Web (SCTs, OCSP staples)	14.82	17.99	21.17	24.34	Dilithium-2
	20.14	24.45	28.76	33.07	Dilithium-3
	5.43	6.97	8.52	10.07	Falcon-512
	9.84	12.52	15.20	17.88	Falcon-1024
	89.28	133.80	178.33	222.85	Rainbow-I(cz)
	40.08	46.37	52.66	58.95	SPHINCS$^+$-128s

In order to quantify the impact of PQ algorithms on QUIC's amplification protection, we analyzed the size of QUIC messages using PQ algorithm candidates in NIST's Round 3. For key exchange, we analyzed hybrid key exchange that uses X25519 [19] with a PQ KEM candidate NIST Round 3 finalist like Kyber, Saber or NTRU. We evaluated various parameters for these PQ KEMs. Regarding signatures, we chose the leanest lattice-based signature parameters (i.e., Dilithium-2, Falcon-512). We assumed no SCTs or OCSP stapling were used, 500B of X.509 attributes and the certificates were DER encoded.

Table 2 summarizes the QUIC message sizes. We observe that all of the PQ signature options will exceed (in red) 3× the initial request packet and will trigger QUIC's amplification protection which would lead to a slowdown due to the extra round-trip.

Table 2. Approximate Initial Client and Server Data in QUIC using Hybrid Key Exchange and PQ signatures

Initial Client Data (B)	Server Data (KB)			PQ Algorithms
	1 ICA	2 ICAs	3 ICAs	
1050	13.32	16.49	19.66	(X25519 + Kyber-512)+ Dilithium-2
1050	5.68	7.22	8.77	(X25519 + Kyber-512) + Falcon-512
949	5.61	7.16	8.70	(X25519 + ntruhps2048509) + Falcon-512
922	5.65	7.19	8.74	(X25519 + LightSaber) + Falcon-512
1434	6.00	7.54	9.09	(X25519 + Kyber-768) + Falcon-512
1181	5.84	7.39	8.94	(X25519 + ntruhps2048677) + Falcon-512
1242	6.00	7.54	9.09	(X25519 + Saber) + Falcon-512
1818	6.48	8.02	9.57	(X25519 + Kyber-1024) + Falcon-512
1480	6.14	7.69	9.23	(X25519 + ntruhps4096821) + Falcon-512
1562	6.38	7.93	9.48	(X25519 + FireSaber) + Falcon-512

From the data in Tables 1 and 2 it is clear that big certificate chains could pose challenges to secure tunnel establishment. EAP, constrained condition applications, and post-quantum (D)TLS and QUIC connections will be negatively affected. Our goal is to alleviate these challenges in order to speed up the secure tunnel handshakes.

3 Related Work

Various works have focused on the issues certificate chains introduce to certain usecases. IETF RFC drafts draft-ietf-emu-eaptlscert [39] and ietf-emu-eap-tls13 [21] discuss the problem of big TLS certificate chains for EAP authentication. [15] presents the heavy authentication data issue with (D)TLS connections in Wi-SUN and IEEE 802.14.5 mesh networks. Compact TLS (cTLS) IETF draft draft-ietf-tls-ctls [31] is a compact version of TLS 1.3 designed for constrained conditions. cTLS proposes using pre-determined certificate dictionaries which peers can use to convey their certificate chains without actually sending the certificates. In the same context, Mozilla introduced an ICA Pre-load list from the multi-browser Common CA Database (CCADB) [25] in its Desktop Firefox browser in 2020.

On the size of post-quantum authentication data, early work by Bindel et al. emulated large hybrid PQ certificates and studied their impact on TLS libraries and browsers [3]. [16] showed that post-quantum Stateful Hash-Based Signatures in certificates will not break (D)TLS, QUIC and IKEv2. Sikeridis et al. [42] also studied the impact of PQ signatures on TLS 1.3 and proved that lattice-based PQ candidates offer the most efficient options whereas all other NIST Round 2 schemes could introduce round-trips due to the TCP `initcwnd`.

Additionally, [41] tested hybrid (i.e. classical ECDH and PQ KEMs) key exchange and signatures in TLS 1.3 and SSH. It showed that some lattice-based PQ algorithms schemes do not detrimentally slow down TLS handshakes. [51], on the other hand, showed that 9–10 KB certificate chains will lead to double digit TLS handshake slowdowns and some clients or middleboxes cannot handle chains larger than 10KB.

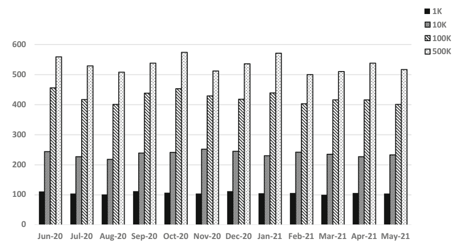
In [6], Crockett et al. presented the challenges of implementing NIST's PQ key exchange and authentication algorithms in TLS and SSH, with a focus on hybrid schemes. What's more, Paquin et al. showed in [29] that the more data included in a PQ handshake, the higher the loss probability ($1-(1-p)^n$, where n is the number of authentication packets and p is the individual packet loss probability) and the connection slowdown in unstable network environments. Finally, Schwabe et al. proposed KEMTLS [38] which uses PQ KEMs in the leaf certificate. One of the advantages it offers is that the certificate chain ends up being a few KB smaller than it would have been when using lattice-based PQ signatures.

4 ICA Statistics

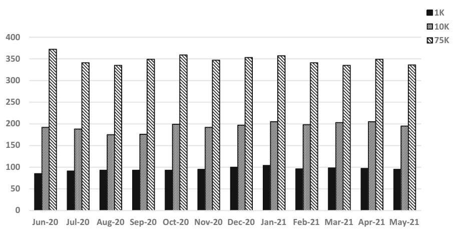
From our analysis so far, it is clear that certificate chains could pose challenges to secure tunnel establishment for some of today's usecases and most future post-quantum ones. A straightforward, potential solution which we will investigate in Sect. 5 is caching ICA certificates and omitting them from the handshake. Before looking into solutions, we wanted to study how many ICA certificates exist today and how many someone would need to cache in order to speed up secure tunnel establishment.

Initially, we used CCADB's ICA list to get the number of non-revoked active ICA certificates used in the Web. We also used the Alexa and Cisco Umbrella Top1M datasets [1,47] for the top visited sites. Alexa is a well-known ordered list of the most popular sites on the Internet. Since Alexa Top1M stopped being free, Cisco published their own dataset, Cisco Umbrella Top1M. The Umbrella dataset is different than Alexa's as it is built with a different methodology. In order to retrieve the certificates of the top visited sites in our two datasets, we used data from Censys.io [4]. Censys.io is a popular analytics engine which scans the Internet daily and inventories information about connections to public open servers.

We wanted to investigate the status of the certificate chains returned from servers over time. Thus, we analyzed the certificates returned from the Alexa and Umbrella sites at the beginning of each month for a period of 12 months (June 2020-May 2021). The results are shown in Table 3. We can see that the total non-revoked Web ICA certificates (based on CCADB [26]) were 1306 which roughly matches with ICA count for the Web in [50]. For the domains within the Top1M Alexa sites supporting HTTPS, the distinct ICAs were 500–560. For the domains that support HTTPS within the Top1M Umbrella sites, the ICAs were 335–375. Caching these ICA certificates is manageable for most usecases.



(a) Alexa



(b) Umbrella

Fig. 2. # of ICAs for TopX servers (12 month)

Our results also show that a few of the top sites on the Internet do not include ICAs in their certificate chain; they either send only a server certificate or a server certificate and Root CA certificates. We also see that ∼99% of the sites include up to 3 ICA certificates and ∼96% (∼92% for Umbrella) include 1–2 ICA certificates. We also see a small amount of self-siged (SS) certificates. Self-signed certificates could be server or Root CA certificates. For the purpose of alleviating as much handshake data as possible for the (D)TLS handshake, Root CA certificates should not be sent[1]. We also observed a limited number of certificates which were not server certificates but also had the *BasicConstraints* X.509 extension set to *cA:False*. These were misconfigurations. Readers should note that Alexa's Top1M sites ended up being ∼500K and Umbrella's ∼75K servers. The reason is that we investigated servers that were listening on well-known TLS port 443.

Subsequently, we wanted to quantify the ICA certificates in our datasets for each month and how the count increased based on the top servers visited. Figure 2 shows the ICA certificate count per month using data from Censys.io for the Top 1K, 10K, 100K, 500K servers for Alexa and 1K, 10K, 75K for Umbrella. Note that Umbrella's dataset included much less TCP port 443 active servers which is why the count is lower. We can see how the ICA count increases with the number of top servers. Intuitively that makes sense as the more servers one examines (using scanning or by directly querying databases such as Censys.io) the more ICAs are discovered and thus the less ICAs remain unseen. As we can see in the two figures, the ICA size growth slows down as the servers increase; the ICA counter increases faster between 1K and 10K servers. Most Internet peers tend to get certs from a limited set of CAs (e.g., Let's Encrypt CA) and not evenly use all ICAs available. Thus, the rate of increase of the ICA count is slower than the server count.

[1] Because certificate validation requires that root keys be distributed independently, the self-signed certificate that specifies the root certificate authority MAY be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case [7,30].

Table 3. CA data from Alexa, Umbrella Top 1M (12 month) and Mozilla's CCADB (June 2021).

Data Set	0 ICAs	1 ICA	2 ICAs	3 ICAs	>3 ICAs	# distinct servers	SS certs	non-CA certs	Distinct ICAs
Alexa 06-2020	632	388071	82325	12418	579	484025	45738	3545	559
Alexa 07-2020	1339	350178	74475	11661	685	438338	39543	6104	529
Alexa 08-2020	629	318805	66524	11369	442	397769	38741	3252	508
Alexa 09-2020	582	323014	67606	11680	380	403262	37071	3057	538
Alexa 10-2020	816	480133	104064	16734	537	602284	58195	4258	574
Alexa 11-2020	498	339441	62940	12496	442	415817	41138	3177	512
Alexa 12-2020	1136	353411	73531	12878	580	441536	40064	6764	536
Alexa 01-2021	1350	414144	90538	15209	719	521960	46379	7177	571
Alexa 02-2021	1090	362439	75886	12994	693	453102	39501	6314	500
Alexa 03-2021	1006	364351	77777	12973	627	456734	41062	6024	510
Alexa 04-2021	809	469614	98482	17240	556	586701	50654	3714	538
Alexa 05-2021	607	440865	92548	13469	502	547991	45668	3166	517
Umbrella 06-2020	52	56981	13216	5171	47	75467	11795	465	372
Umbrella 07-2020	43	54480	11887	4860	54	71324	11040	425	341
Umbrella 08-2020	52	52741	11353	4877	44	69067	10780	424	335
Umbrella 09-2020	42	53744	11603	5042	44	70475	11070	411	349
Umbrella 10-2020	63	64214	14228	6113	71	84689	13261	570	359
Umbrella 11-2020	44	57246	11475	5554	53	74372	11316	458	347
Umbrella 12-2020	54	57844	11905	5482	63	75348	11117	478	353
Umbrella 01-2021	53	62994	13129	5829	47	82052	12395	538	357
Umbrella 02-2021	48	59890	11432	5560	49	76979	10624	504	341
Umbrella 03-2021	58	56084	10878	5263	34	72317	99994	31	335
Umbrella 04-2021	62	67689	14042	7044	65	88902	12748	569	349
Umbrella 05-2021	49	66686	13554	7020	61	87370	12122	542	336
Mozilla CCADB [26]									1306

Table 4. Intersection of domains and ordinal dissimilarity (using the Kendall rank correlation coefficient) between the Alexa and Umbrella lists over time.

Date	**2020-06-02**	**2020-07-07**	**2020-08-04**	**2020-09-01**	**2020-10-06**	**2020-11-03**
Intersection	53909	49755	49414	49430	60569	51491
Kendall τ	0.258	0.226	0.221	0.247	0.241	0.256
Date	**2020-12-01**	**2021-01-01**	**2021-02-01**	**2021-03-01**	**2021-04-01**	**2021-05-01**
Intersection	53367	58347	54202	51822	62446	61465
Kendall τ	0.256	0.238	0.234	0.228	0.249	0.247

We also wanted to compare our datasets. Table 4 shows the intersection of domains between the Alexa and Umbrella lists. Specifically, the table tabulates the number of common domains supporting HTTPS/TLS found in Censys.io for the twelve days we examined. We observe that the two top lists are different; this is expected since they are compiled using different methodologies (see also [37,46]). The table also illustrates the ordinal similarity between the two lists using the Kendall rank correlation coefficient (also employed in [37] for similar comparisons). Although the two top lists are dissimilar, the number of ICAs required to be cached does not differ much.

5 Speed Up Mechanisms

So far we have seen that data-heavy authentication could negatively affect secure tunnel establishment. Thus, limiting the authentication data in these connections could alleviate the issue. There are multiple ways to achieve that. The most straightforward one is to suppress the ICA certificates in the handshake after the peer has cached them and signalled that it does not need them. [18] proposes using a TLS 1.3 flag extension to request the peer to suppress its ICA ceritificates. A flag in the `ClientHello` and the `CertificateRequest` would suffice for the client or server to request ICA suppression. Suppressing ICAs would drop the authentication data in Table 1 to acceptable levels (9–11 KB) even for Web TLS connections. It would also drop the server response sizes in Table 2 to $\sim$5.5 and 3.2KB for Dilithium-2 and Falcon-512 respectively. Extrapolating from [41,42, 51], that would speed up the handshake by $\sim$60ms when TCP `initcwnd`=10MSS or $\sim$10% when `initcwnd`=30MSS.

Another option for TLS is to increase the TCP `initcwnd`. [41] showed that by generously increasing it, handshakes can be sped up by eliminating extra round-trips. Readers should note that this option does not prevent handshake slowdowns in environments with increased loss probability [29]. But even in non-lossy environments, [51] showed that 9–10 KB certificate chains could lead to double digit TLS handshake slowdowns even with 30MSS TCP `initcwnd`. Additionally, although some Content Deliver Network (CDN) providers optimize their infrastructures and increase the TCP `initcwnd`, generally increasing `initcwnd` without thorough experimentation could negatively affect constrained usecases, slow links, cellular networks, bursty traffic patterns, and highly multiplexed links in developing regions [5, § Appendix A].

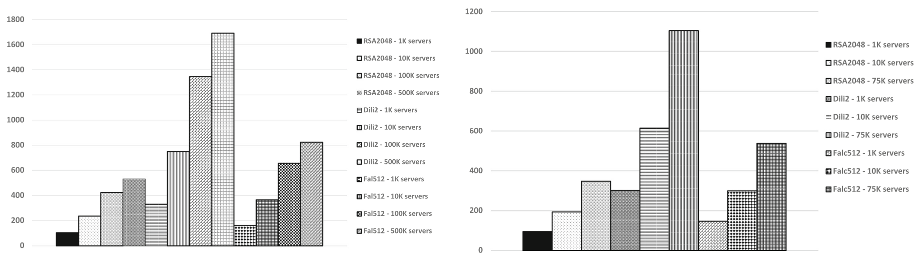
Alternatively, we could omit all certificates and use a fingerprint in the TLS handshake to indicate which peer certificate we have cached as proposed in [36]. Although standardized a few years already, this mechanism has not seen wide industry adoption. It also introduces security caveats and operational concerns like allowing TLS session correlation [36, § 7] and actively and frequently managing and updating a large certificate cache of certificates with relatively short lifetimes.

Section 2 discussed how QUIC amplification protection will introduce extra round-trips when using post-quantum signature algorithms. One option to prevent these round-trips is to include QUIC `PADDING` frames in the request in order for the response to fit within the 3$\times$ size of the request. We could pad with enough data to be safe in all cases, but then the client wastes resources unnecessarily without even knowing if a round-trip would be warranted based on the server's supported algorithms and certificates. Additionally, even if we prevented the round-trips, excessive authentication data will still be sent (in one round-trip) which still introduces increased loss probability in unstable or congested networks [29,51].

[42, §VII-B] suggests using different signature mechanisms at the Root CA, the OCSP staple and the SCTs (for the Web) as a way to alleviate the data issue. The Root CA and the OCSP staple and SCT public keys are not sent in the handshake, thus using a signature algorithm that has a small signature would slim down the data sent. Example algorithms would be Stateful HBS signatures [11, 22] or multivariate candidates in NIST's PQ Project like Rainbow. Although this method would trim down the data, big signatures or public keys would still be included in the handshake, so the issue is not eliminated. And we should not underestimate that this option would require peers to support multiple signature algorithms. Introducing new algorithms has traditionally not been a smooth process for the industry.

cTLS [31] proposes using pre-established dictionaries to omit sending certificates in the handshakes. This method would work nicely for peers that can be provisioned with the right certificate dictionaries. Different usecases like the Web would pose challenges with establishing these dictionaries, keeping them up-to-date and making sure the peers have the same version.

From our analysis, it is obvious that the most straightforward option to convey to the (D)TLS peer to omit its ICA certificates is using a TLS 1.3 flag [18, 27]. This mechanism is backwards compatible; if the peer does not support the flag the connection still completes. It also would work well when we communicate with finite peers whose ICA certificates is trivial to cache. In usecases where there are multiple or infinite peers, we need an all-inclusive ICA list or an ICA caching mechanism which we discuss in Sect. 6.



(a) Alexa (b) Umbrella

Fig. 3. 12 month average ICA cache size (KB) for TopX servers

6 ICA Caching

After discussing the best options to alleviate the authentication-data issue, we wanted to quantify the size of the ICA cache that would suffice for clients visiting the Top1M servers in our datasets. We analyzed the ICA certificate cache size as the top server count in our datasets increases. Again, we assumed 500B of X.509 attributes in each binary DER encoded [12] certificate. We also assumed that the same algorithm is used for all signatures in the TLS handshake and that ICAs certificates do not contain SCTs.

Figure 3 shows the ICA certificate sizes for RSA-2048, Dilithium-2 and Falcon-512 based on the average 12-month ICA certificate count from our Alexa and

Umbrella datasets. We can see how the cache size for caching all ICAs increases as we include more servers from 1K to 500K for Alexa and from 1K to 75K for Umbrella. Intuitively that makes sense as more servers mean more CA vendors issuing their certificates. Figures 3a shows that the ICA cache size for Alexa servers will not exceed 550KB on average for RSA-2048 ICAs, 1.7MB for Dilithium-2 and 850KB for Falcon-512 ICAs. Figure 3b shows the equivalent sizes for Umbrella are 400KB, 1.1MB and 600KB. Although someone could use more efficient caching mechanisms than caching all ICAs, we believe this analysis shows that caching does not introduce detrimentally high resource requirements (<1-2MB) even for big post-quantum ICA certificates. For comparison, the Web cache size including all RSA and ECDSA ICAs chaining to Root CAs trusted in the Mozilla Root Program is ∼1MB [50] which is manageable for most applications.

Although we could probably cache all ICAs for some usecases, it would make more sense to limit our cache size especially for cases where an entity can communicate with infinite peers or where an up-to-date full ICA list is not available. A crude option could be to use a static partial list of common ICAs and only request suppression from peers we have seen before that use a chain with ICAs in that list. More efficient options are, of course, possible. Caching is a well-studied topic used in various Internet and memory usecases. Most mechanisms cache data and usually have a way to update the cache when there is a cache miss (missing entry in the cache).

For the purposes of ICA caching we can follow a similar approach. We initially have our ICA List which consists of the ICA certificates cached while connecting with peers. These certificates can be omitted from subsequent connections. The ICA List entries are referenced by a secondary list (Peer List) which binds peers with the ICAs cached. The Peer List is an ordered list for faster lookups. It includes the ICAs in the peer's certificate chain, a counter of the times communicated with that peer, a timestamp for the last communication and a timeout value. These attributes will be used by the caching mechanism in order to update the cache. Figure 4 shows the ICA cache architecture.

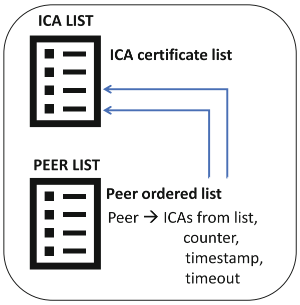


Fig. 4. ICA Cache

The Peer List timeout is used to clean up the cache at regular maintenance intervals. It can be set according to a default timeout value, or it can be updated based on the frequency of a cache miss. Busy caches dealing with multiple peers are normally more frequently updated. Deciding on the best timeout value is a trade-off decision; the lower the timeout the more operational burden on the cache, the higher the timeout the more peer entry evictions will need to take place with a cache miss. Additionally, timeouts can be set based in certificate expiration. When adding a peer entry, we could set the timeout to the peer certificate expiration. When the peer certificate expires, it is expected that we would need a new connection to get its new certificate and ICAs. An ICA revocation would not affect security, as the ICA cache does not preclude normal certificate revocation checks when validating the peer identity.

Algorithm 1 shows a simple cleanup mechanism based on one criterion. That could be the timeout in a peer entry in the Peer List which removes the peer from the list. Generally, if there are ICAs in the ICA List which are not referenced by any peer entries, these are deleted as proper cache hygiene.

```
for PeerEntry in Peer List do
   if (PeerEntry eviction criteria met) then
      Remove PeerEntry from Peer List
   end if
end for
for ICAEntry in ICA List do
   if (ICAEntry is not referenced by any peer in the Peer List) then
      Remove ICAEntry from ICA List
   end if
end for
```

Algorithm 1: ICA Cache Cleanup

Algorithm 2 shows the process of updating the ICA and Peer Lists when connecting to a peer. Before connecting to a new peer, we look up the Peer List and if the peer already exists we update the counter and timestamp of the entry and connect by asking for ICA suppression. If the connection fails due to a certificate chain authentication error we remove that peer from the Peer List so that subsequent connections will not depend on the cached ICAs.

In case the peer does not exist in the Peer List, we connect without ICA suppression, we update the ICA cache with the peer's ICAs and add the peer in the Peer List. When updating the cache with new ICAs we can use Algorithm 1 if the cache is full. We could also remove peer entries and their corresponding ICAs from the two lists in order to make room by using different criteria like peer age (timestamps), counter, or randomly.

```
if (NewPeer in Peer List) then
   Update NewPeer entry counter, timestamp in Peer List
   Connect to NewPeer asking for ICA suppression
   if (Connection failed) then
      Remove NewPeer from Peer List
   end if
else
   Connect to NewPeer without ICA suppression and get NewPeer ICAs.
   while (not enough room in cache for NewPeer and its ICAs) do
      Make room for ICAs (Algorithm 1)
   end while
   Add NewPeer in Peer List
   Add NewPeer ICAs in ICA List
end if
```

Algorithm 2: New Connection ICA cache Update

Note that the caching mechanisms discussed do not require fault-tolerance, persistence or replication. They are built on straightforward concepts. In the event of a failure or application crash the cache can be rebuilt without having detrimental results other than potentially slower (due to heavy authentication-data) initial handshakes. Additionally, readers should note that the proposed ICA cache does not act as a CA store. These ICA certificates are cached, but not trusted by default in any way. Certificate chain validation does not change.

7 Considerations

Signalling to the peer that it ought to not send its ICA certificates is a straightforward option to trim data from the handshakes, but it comes with security considerations. When the client includes the TLS flag in its `ClientHello`, the flag is cleartext and passive observers could tell that the client is requesting ICA suppression. As the feature would not be immediately ubiquitous in all clients, passive observers could use the signal to fingerprint clients and know which ones the traffic is coming from. If the TLS flag is included in the server `CertificateRequest` which is an encrypted message in TLS 1.3 the concern does not apply.

Client fingerprinting is a security concern especially in a world where user privacy has become top of mind for users and vendors. We should note that the information leaked by the TLS flag is about the client having communicated with certain peers and about having added support for ICA suppression. Although these can be considered some loss of privacy, there is no additional leakage about the identity of the client or the server. To alleviate fingerprinting concerns, the client can encrypt its `ClientHello` using draft-ietf-tls-esni [33]. Although harder, a passive observer could still infer that the client asked for suppression if the server supports it by observing the amount of encrypted data sent from the server in the response. Wide-adoption of the suppression mechanism in TLS clients and servers would minimize the effect of any fingerprinting. Even then, an ICA system cache could create side-channels which could leak information (e.g., communicating destinations) even to off-path adversaries. Thus, the cache should be protected.

Based on the usecase, ICA suppression would be trivial when the amount of peers we are communicating with is finite or when the peers belong to a small set of PKI domains. For example, a device that only communicates with a controller and a few direct peers could trivially cache all ICAs. Usecases where the peers are infinite or the PKI domains numerous will need ICA caching mechanisms like the ones described in Sect. 6 when an up-to-date full ICA list is not available. Desktop Firefox preloading all Web ICAs [24] is another example where resourceful entities can store all ICAs that their peers could be using. Depending on how elaborate the caching mechanism is and how frequently cache misses take place, caching can be an operational concern. Designing proper caching is important for these cases especially since failures could have adverse effects on performance they are trying to improve.

Including one TLS flag for the client and one for the server to signal ICA suppression means that they can only ask for all or none of the ICA certificates from their peer. Sometimes ICA cache size is limited. In these cases we may want to control the ICA size. We may not want to cache ICAs that sign server certificates; we may want to start from second-level ICAs and above. We could achieve that by introducing one more TLS flags. Note that sometimes first or second-level ICAs are not clear-cut as an ICA may have signed leaf certificate and subordinate ICA certificates. To address that ambiguity in the caching algorithm, we would only depend on the peer certificate chain to decide the level of the ICA. We need detailed testing to decide if the complexity of another TLS flags is worth the cache size savings and the TLS handshake speedup.

The proposed caching mechanisms in Sect. 6 are based on straightforward practical approaches. Other options may exist. For example, a new ICA Chain List could include lists of pointers to ICA entries and a timeout or timestamp. The Peer List entries would then point to ICA Chain List entries. ICA List entries would be removed only when they are not referenced by any ICA Chain entry. Whole ICA Chain List entries would be removed when they are not referenced by Peer List entries or when their timeout expires. In the event of full cache and a cache miss, ICA Chain entries, the corresponding peers and orphan ICAs would be removed to make room for new entries. Eviction mechanisms could include timeouts, age (timestamps) or counters in the ICA Chain entries.

As discussed in Sect. 6, the ICA cache does not operate like a CA Trust Store. Cached certificate authorities are not pre-trusted; they are only cached to avoid being sent on the wire. Someone may argue that the handshakes could be sped up by altering the chain validation to use the ICA cache entries as trusted certificates. There is a precedent with Trust Stores sometimes including ICAs and Root CAs. Ryan Sleevi explained in [44] the complication of overlapping certificate chains and how implementations suffer in correctly dealing with them. We consider pre-trusting any of the ICA cache certificates as more involved. We prefer solutions that align well with operations today and do not introduce new risks.

8 Conclusion and Future Work

In conclusion, in this work we saw that authentication data-heavy (D)TLS handshakes are slower. We quantified the issue and discussed usecases that are mostly affected by it. We evaluated potential alleviation mechanisms and argued that ICA suppression is the best option. We quantified the ICAs in use on the Internet today and we proved that ICA suppression can be made possible by caching ICAs. We also qualitatively evaluated ICA suppression signaling and caching mechanisms to make this possible. As future work, we are planning to investigate the expected ICA cache sizes for different usecases, applications and traffic profiles different that the Web. Finally, reviving [18] in IETF would be the next step to make authentication data-heavy (D)TLS connections lighter.

References

1. Amazon: Alexa top 1 Million, August 2021. https://www.alexa.com/topsites/
2. Apple: Apple's Certificate Transparency policy, March 2021. https://support.apple.com/en-us/HT205280
3. Bindel, N., Herath, U., McKague, M., Stebila, D.: Transitioning to a quantum-resistant public key infrastructure. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 384–405. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_22
4. Censys: censys.io data, August 2021. https://censys.io/data
5. Chu, J., Dukkipati, N., Cheng, Y., Mathis, M.: Increasing TCP's initial window. RFC 6928, April 2013. https://doi.org/10.17487/RFC6928. https://rfc-editor.org/rfc/rfc6928.txt
6. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. In: NIST 2nd Post-Quantum Cryptography Standardization Conference 2019, August 2019
7. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. https://doi.org/10.17487/rfc5246. https://rfc-editor.org/rfc/rfc5246.txt
8. Fluhrer, S., Dang, Q.: Additional parameter sets for LMS hash-based signatures. Internet-Draft draft-fluhrer-lms-more-parm-sets-05, Internet Engineering Task Force, June 2021. https://datatracker.ietf.org/doc/html/draft-fluhrer-lms-more-parm-sets-05. work in Progress
9. Hoffman, P.E.: The transition from classical to post-quantum cryptography. Internet-Draft draft-hoffman-c2pq-07, Internet Engineering Task Force, May 2020. https://datatracker.ietf.org/doc/html/draft-hoffman-c2pq-07. work in Progress
10. Housley, R.: Use of the HSS/LMS Hash-based signature algorithm in the cryptographic message syntax (CMS). RFC 8708, February 2020. https://doi.org/10.17487/RFC8708. https://rfc-editor.org/rfc/rfc8708.txt
11. Huelsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended merkle signature scheme. RFC 8391, May 2018. https://doi.org/10.17487/RFC8391. https://rfc-editor.org/rfc/rfc8391.txt
12. International Telecommunications Union (ITU-T): ASN.1 encoding rules: specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). https://www.itu.int/rec/T-REC-X.690-202102-I/en
13. International Telecommunications Union (ITU-T): X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. https://www.itu.int/rec/T-REC-X.509/en
14. Iyengar, J., Thomson, M.: QUIC: A UDP-based multiplexed and secure transport. RFC 9000, May 2021. https://doi.org/10.17487/RFC9000. https://rfc-editor.org/rfc/rfc9000.txt
15. Kampanakis, P., Chandra, R.: Mechanism to speed up secure communication handshakes in constrained conditions. Technical Disclosure Commons, December 2020. https://www.tdcommons.org/dpubs_series/3916/
16. Kampanakis, P., Panburana, P., Daw, E., Van Geest, D.: The viability of post-quantum X.509 Certificates. IACR Cryptology ePrint Archive 2018, 63 (2018)
17. Kampanakis, P., Sikeridis, D.: Two PQ signature use-cases: non-issues, challenges and potential solutions. Cryptology ePrint Archive, Report 2019/1276 (2019). https://ia.cr/2019/1276

18. Kampanakis, P., Stebila, D., Friedl, M., Hansen, T., Sikeridis, D.: Post-quantum public key algorithms for the Secure Shell (SSH) protocol. Internet-Draft draft-kampanakis-curdle-pq-ssh-00, Internet Engineering Task Force, October 2020. https://datatracker.ietf.org/doc/html/draft-kampanakis-curdle-pq-ssh-00. work in Progress
19. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748, January 2016. https://doi.org/10.17487/RFC7748. https://rfc-editor.org/rfc/rfc7748.txt
20. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962, June 2013. https://doi.org/10.17487/RFC6962. https://rfc-editor.org/rfc/rfc6962.txt
21. Mattsson, J.P., Sethi, M.: Using EAP-TLS with TLS 1.3 (EAP-TLS 1.3). Internet-Draft draft-ietf-emu-eap-tls13-18, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-emu-eap-tls13-18. work in Progress
22. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali hash-based signatures. RFC 8554, April 2019. https://doi.org/10.17487/RFC8554. https://rfc-editor.org/rfc/rfc8554.txt
23. Montenegro, G., Schumacher, C., Kushalnagar, N.: IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals. RFC 4919, August 2007. https://doi.org/10.17487/rfc4919. https://rfc-editor.org/rfc/rfc4919.txt
24. Mozilla: Preloading intermediate CA certificates into Firefox, November 2020. https://blog.mozilla.org/security/2020/11/13/preloading-intermediate-ca-certificates-into-firefox/s
25. Mozilla: Common CA Database (CCADB), July 2021. https://www.ccadb.org/resources
26. Mozilla: Common CA Database (CCADB), February 2022. https://ccadb-public.secure.force.com/mozilla/MozillaIntermediateCertsCSVReport
27. Nir, Y.: A flags extension for TLS 1.3. Internet-Draft draft-ietf-tls-tlsflags-06, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-tls-tlsflags-06. work in Progress
28. Ounsworth, M., Pala, M.: Composite signatures for use in internet PKI. Internet-Draft draft-ounsworth-pq-composite-sigs-05, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ounsworth-pq-composite-sigs-05. work in Progress
29. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 72–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_5
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. https://doi.org/10.17487/RFC8446. https://rfc-editor.org/rfc/rfc8446.txt
31. Rescorla, E., Barnes, R., Tschofenig, H.: Compact TLS 1.3. Internet-Draft draft-ietf-tls-ctls-03, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-03. work in Progress
32. Rescorla, E., Modadugu, N.: Datagram transport layer security version 1.2. RFC 6347, January 2012. https://doi.org/10.17487/rfc6347. https://rfc-editor.org/rfc/rfc6347.txt
33. Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-12, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12. work in Progress

34. Rescorla, E., Tschofenig, H., Modadugu, N.: The datagram transport layer security (DTLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-dtls13-43, Internet Engineering Task Force, April 2021. https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43. work in Progress
35. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, D.C.: X.509 internet public key infrastructure online certificate status protocol - OCSP. RFC 6960, June 2013. https://doi.org/10.17487/RFC6960. https://rfc-editor.org/rfc/rfc6960.txt
36. Santesson, S., Tschofenig, H.: Transport layer security (TLS) cached information extension. RFC 7924, July 2016. https://doi.org/10.17487/RFC7924. https://rfc-editor.org/rfc/rfc7924.txt
37. Scheitle, Q., et al.: A long way to the top: significance, structure, and stability of internet top lists. In: Proceedings of the Internet Measurement Conference 2018, pp. 478–493. IMC 2018, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3278532.3278574. https://doi.org/10.1145/3278532.3278574
38. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1461–1480. CCS 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3423350. https://doi.org/10.1145/3372297.3423350
39. Sethi, M., Mattsson, J.P., Turner, S.: Handling large certificates and long certificate chains in TLS-based EAP methods. Internet-Draft draft-ietf-emu-eaptlscert-08, Internet Engineering Task Force, November 2020. https://datatracker.ietf.org/doc/html/draft-ietf-emu-eaptlscert-08. work in Progress
40. SHODAN: HTTPS (443) Overview (2019). https://www.shodan.io/report/mNs9fa3I
41. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In: Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies, pp. 149–156. CoNEXT 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3386367.3431305. https://doi.org/10.1145/3386367.3431305
42. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-quantum authentication in TLS 1.3: a performance study. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, 23–26 February 2020. The Internet Society (2020). https://www.ndss-symposium.org/ndss-paper/post-quantum-authentication-in-tls-1-3-a-performance-study/
43. Simon, D., Hurst, R., Aboba, D.B.D.: The EAP-TLS authentication protocol. RFC 5216, March 2008. https://doi.org/10.17487/RFC5216. https://rfc-editor.org/rfc/rfc5216.txt
44. Sleevi, R.: Path building vs path verifying: the chain of pain, June 2020. https://medium.com/@sleevi_/path-building-vs-path-verifying-the-chain-of-pain-9fbab861d7d6
45. Stebila, D., Fluhrer, S., Gueron, S.: Hybrid key exchange in TLS 1.3. internet-Draft draft-ietf-tls-hybrid-design-03, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-03. work in Progress
46. Lists Study, T.: Scheitle, quirin and jelten, jonas, July 2021. https://toplists.github.io/index.html

47. Systems, C.: Cisco Umbrella 1 Million, August 2021. https://umbrella.cisco.com/blog/cisco-umbrella-1-million
48. Thomson, M., Turner, S.: Using TLS to Secure QUIC. RFC 9001, May 2021. https://doi.org/10.17487/RFC9001. https://rfc-editor.org/rfc/rfc9001.txt
49. Tjhai, C., et al.: Multiple Key Exchanges in IKEv2. Internet-Draft draft-ietf-ipsecme-ikev2-multiple-ke-03, Internet Engineering Task Force, July 2021. https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-multiple-ke-03. work in Progress
50. Valsorda, F.: filippo.io/intermediates, February 2022. https://github.com/FiloSottile/intermediates
51. Westerbaan, B.: Sizing up post-quantum signatures, November 2021. https://blog.cloudflare.com/sizing-up-post-quantum-signatures/

Enhancing Cybersecurity of Satellites at Sub-THz Bands

Rajnish Kumar(✉) and Shlomi Arnon

Ben-Gurion University of the Negev, Beersheba, Israel
rajnish@post.bgu.ac.il, shlomi@bgu.ac.il

Abstract. In this work, we present a novel idea to enhance the cybersecurity of a low earth orbit (LEO) satellite link assuming that the communication between the satellite (Alice) to the ground station receiver (Bob) is done at sub-THz frequency bands. Such short wavelength creates highly directional antenna beam with very small spot on the ground and thus making it difficult for eavesdropping. However, the satellite transmitter power can be further tuned close to the noise floor so as to make the eavesdropping even more practically difficult. In order to augment the security of the link further, we communicate only within a very specific and designated angle of arrivals of the satellite signal with transmitter power tuned close to the noise floor. The lowering of transmitter power will lead to a lower signal-to-noise ratio (SNR) at the ground station receiver and will thus cause a very high probability of eavesdropper (Eve) outages. In order to improve the SNR of Bob's receiver, we derive an optimization algorithm considering the impact of thermal noises from the sky, the ground and the receiver circuit. The algorithm adapts the inter-element spacing between the phased array elements to shape the gain pattern so as to minimize the required transmitter power for a given SNR at the receiver of Bob.

Keywords: Satellite communication · THz frequency · Cybersecurity · Phased array

1 Introduction

The evolution of technology to 6G will make LEO satellites an integral part of the communication network. To meet the increasing consumer demands with massive traffic volume, these LEO satellites will be designed at sub-THz frequencies above 100 GHz to exploit the high bandwidth available [1]. With the launch of mega-constellations, the satellites are expected to play a larger role in the communication infrastructure so the bad actors from international level to low-level criminals are trying to take advantage from the growing satellite network [2,3]. With the advancement in technology, the easy and low cost availability of communication equipment like signal sources, low-noise amplifiers, mixers, filters, modulators and demodulators integrated circuits, phased array beamformers, and other commercial off-the-shelf electronics and open source software makes the satellite links more vulnerable to the cyberattacks [4]. There are growing concerns regarding the cybersecurity of satellites as the hackers may be able to disrupt many essential services like GPS, vehicles, drones, offshore oil and gas operations and electricity grid [5,6]. Hackers could deorbit and destabilize the satellite and make them collide with each other causing havoc to the infrastructure. With the recent progress in the applications of machine learning in communication systems, the security may be more compromised [7,8]. So, the 6G satellite networks will have a high priority for privacy and security [9,10]. Some of the risks in the 6G network will include the denial-of-service attacks, data pollution attacks, and control-flow hijacking besides kinetic and electronic attacks [11]. Such cyberattacks can lead to the corruption and interception of data, and breach of security secrets and other sensitive and valuable information.

The communication at sub-THz frequencies will make the link less vulnerable to eavesdropping due to very narrow beam divergence at such high frequency thus creating a smaller spot at ground compared to mm-wave links [12,13]. Thus, THz bands can support secure link with high capacity and ultra-broad bandwidth communication [14]. The communication links can be made secure with the use of various techniques including quantum cryptography [15,16], encryption algorithms [17], and spread spectrum modulations [18]. However, the next generation of communication system relying on quantum computing will make the use of traditional encryption key not difficult to hack [19]. Under such scenario, a more preferred way to deal with cybersecurity in post-quantum era would be by taking advantage of the fundamental physical aspect of the atmospheric channel. The satellite (Alice) communicating with low transmitter power could be one of the possible ways as it will enable covert communication link with the ground station receiver (Bob). The higher attenuation associated with the atmospheric absorption at THz will make sure that the signal with slant path will be severely attenuated in the atmosphere so that the link between Alice and Bob will be highly directional. At such high absorption loss over atmospheric channel, the transmitted power can be tuned very close to the noise floor such that Eve will not able to notice any transmission between Alice and Bob. The one time encryption key between Alice and Bob will then be shared with minimum transmitter power required while maintaining a low bit-error rate (BER) at the receiver of Bob.

In this work, we derive an algorithm to optimally shape the radiation pattern of the phased array receiver at the ground station so as to minimize the required transmitted satellite power for a given number of array elements. The LEO satellite moves in its orbit with a high velocity and so the link dynamics changes rapidly. This could be potentially used to design adaptive receivers that will optimize the performance of the link. The gain pattern of antenna can be optimally shaped by adapting the inter-element spacing between the array emitters. The choice of inter-element spacing will affect the sidelobes, grating lobes and the beamwidth of array antenna [20]. The derived mathematical algorithm will take into consideration the effects of thermal noise from the sky, the

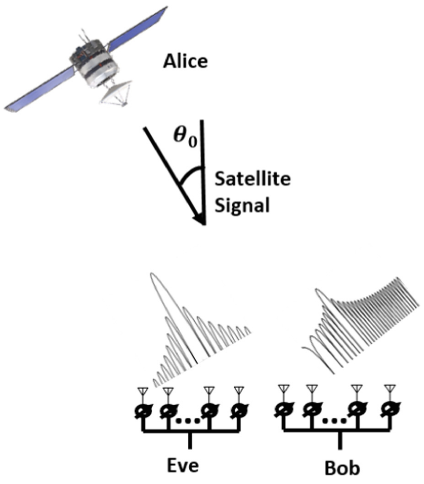


Fig. 1. Scenario under consideration

ground and the receiver circuit connected to the array terminal. Thus, by optimally shaping the radiation pattern of receiver array, the encryption key can be shared with minimum transmitter power covertly between Alice and Bob without being detected by Eve. As compared to a phased array design with uniform spacing of $\lambda/2$ between the array emitters, we show a significant improvement of about 3 dB in the transmitted power at which the Alice can send the encryption key to the receiver of Bob. In Fig. 1, we show the transmitted signal from Alice arriving at the receiver of Bob with an angle of arrival (AoA) θ_0 while Eve is placed nearby. Bob receives the signal with non-uniformly placed phased array with optimized spacing between the elements and Eve receives the signal with uniformly placed array of antenna elements while both use same number of antenna elements. The optimization of transmitter power as the satellite AoA changes at the ground station provides a window of opportunity for transmitting the encryption key safely to Bob thus making the link less vulnerable to eavesdropping.

2 Losses and Noise over Channel

The propagation of satellite signal through the earth's atmosphere will cause signal energy to be absorbed by the gaseous molecules and leading to molecular absorption loss. The absorption loss can be found by slicing the atmosphere into several homogeneous layers of a certain thickness h_i and finding the specific attenuation for each of them defined with a pressure, temperature and water vapor density. Thus, we can express the molecular absorption loss at the AoA θ_0 as [21]

$$L_A(\theta_0) = \prod_{i=1}^{M} e^{k_i(f)d_i(\theta_0)} \tag{1}$$

where, f is the frequency, $k_i(f)$ is the absorption coefficient through each layer, θ_0 is the angle measured from zenith, and $d_i(\theta_0)$ is the slant path length through each gaseous layer expressed as [22]

$$d_i(\theta_0) = -r_i \cos\theta_0 + \sqrt{(r_i^2 \cos^2\theta_0 + 2r_i h_i + h_i^2)} \tag{2}$$

where, r_i is the radius taken from the bottom of i^{th} layer to the center of earth, the earth's radius r_1 is taken as 6371 km km.

The free space path loss for the link as a function of AoA is given by

$$L_{FS}(\theta_0) = \left[\frac{4\pi}{\lambda}(-R_0\cos\theta_0 + \sqrt{(R_0 + h_s)^2 - R_0^2 \sin^2\theta_0})\right]^2 \tag{3}$$

where, λ is the wavelength, h_s is the satellite altitude and R_0 is the radius of the earth.

Noise is the unwanted random signal that interfere with the desired signal in the frequency band. There are various external and internal sources of noise at THz that will degrade the system performance by increasing antenna noise temperature. These noise sources can be natural or man-made. Given the antenna gain pattern $G(\theta, \phi)$ and the brightness temperature $T_b(\theta, \phi)$ of noise source, the antenna noise temperature can be found as given in [23]. The atmospheric media with gaseous constituents will radiate thermal noise into the antenna due to its temperature. The downwelling brightness temperature of gaseous atmosphere of the earth at the ground station antenna is expressed as [24]

$$\begin{aligned} T_{down}(\theta_0) = T_B(f, 2.73) \times 10^{-\sum_{j=k}^{1}\gamma_i d_i(\theta_0)/10} &+ \\ \sum_{n=k}^{1} T_B(f, T_j)(1 - 10^{-\gamma_n d_n(\theta_0)/10})&10^{-\sum_{j=k-1}^{1}\gamma_i d_i(\theta_0)/10} \end{aligned} \tag{4}$$

where, 2.73 K is the cosmic background temperature, T_j is the physical temperature of each layer, and T_B is the brightness temperature of each layer.

The earth at a certain physical temperature will also act as noise source and the upwelling brightness temperature at the antenna will be given by [24]

$$T_{up}(\theta_0) = \epsilon T_B(f, T_{earth}) + \rho T_{down}(180^0 - \theta_0) \tag{5}$$

where, $T_{earth} = 290$ K is the physical temperature of earth's surface (K), ϵ is emissivity, and ρ is effective reflection coefficient.

The antenna will also receive noise from the connected receiver system to its terminal that includes the feeder, low-noise amplifier (LNA), mixer, and Intermediate-frequency (IF) amplifier. The noise temperature at the antenna terminals due to the receiver circuit is [25]

$$T_{RX} = T_F(L_F - 1) + \frac{T_{LNA}}{G_F} + \frac{T_{MX}}{G_F G_{LNA}} + \frac{T_{IF}}{G_F G_{LNA} G_{MX}} \tag{6}$$

where, T_F, and G_F are the noise temperature and gain of antenna feeder respectively and T_{IF}, T_{MX}, and T_{LNA} are the noise temperature of IF amplifier, mixer and LNA respectively. G_{MX}, and G_{LNA} are the respective gains of mixer and LNA.

Thus, at any given angle θ_0 the total noise temperature of the phased array receiver antenna will be given by

$$T_A(\theta_0) = T_{AS}(\theta_0) + T_{AG}(\theta_0) + T_{RX} \tag{7}$$

where, $T_{AS}(\theta_0)$, and $T_{AG}(\theta_0)$ are the noise temperature of antenna due to the downwelling and the upwelling brightness temperature respectively.

3 Minimizing Transmitter Power

The higher losses associated with the THz channel requires higher gain antennas at the transmitter as well at the receiver. The movement of LEO satellite in their orbit requires fast tracking at the ground station which can be achieved by the phased array antennas. The phased array can be electronically steered to direct its main beam to the desired angle of arrival for receiving the satellite signal. In this section, we will minimize the transmitted signal power by optimizing the gain pattern of the antenna for a given number of elements of the phased array. The inter-element spacing between the antenna elements can be adapted so as to shape the antenna gain pattern optimally. By varying the inter-element spacing, we can control the sidelobes, grating lobes, and beam width. We consider that the elements of the phased array are non-uniformly placed such that the array factor will be given by [26]

$$\begin{aligned} AF(\theta) = a_1 + a_2 \exp[jk'd_1(\cos\theta - \cos\theta_0)] + a_3 \exp[jk'(d_1+d_2)(\cos\theta - \cos\theta_0)] \\ +... + a_N \exp[jk'(d_1+d_2+...+d_{N-1})(\cos\theta - \cos\theta_0)] \end{aligned} \tag{8}$$

where, $d_1, d_2, ..., d_{N-1}$ are the spacing between the array elements, $a_1, a_2, ..., a_{N-1}$ are the amplitude excitation of the elements, θ_0 is beam steering angle, θ is the angle taken from zenith, and $k' = \frac{2\pi}{\lambda}$ is the wavenumber.

Assuming that the phased array antenna is lossless, the gain pattern and the maximum gain of the phased array can be calculated as shown in [26]. In order to find the total realized gain of phased array, the maximum gain of the array antenna G_R will multiplied by g, where g is the gain of the array elements.

Now, we can write the expression of transmitter power for a signal arriving with AoA, θ_0 at the output terminals of the phased array receiver as [27]

$$P_T(\theta_0) = \frac{kBL_{FS}(\theta_0)L_A(\theta_0)T_A(\theta_0)SNR(\theta_0)}{gG_TG_R(\theta_0)} \tag{9}$$

As the LEO satellite moves, the atmospheric loss and the free space path loss changes with θ_0 along with the noise temperature of the antenna. This provides an opportunity to shape the gain pattern of phased array optimally at the ground station so as to minimize the requirement on transmitted power P_T. For a given SNR at the receiver, we would minimize the transmitter power $P_T(\theta_0)$ as the

AoA at the receiver is changing due to satellite movement by varying d_n, which will require that

$$\begin{bmatrix} \frac{\partial P_T(\theta_0)}{\partial d_1} & \frac{\partial P_T(\theta_0)}{\partial d_2} & \cdots & \frac{\partial P_T(\theta_0)}{\partial d_{N-1}} \end{bmatrix}^T = 0 \tag{10}$$

Uisng 9 and 10, we manipulate the equations to write

$$\begin{bmatrix} T_A(\theta_0)\frac{\partial G_R(\theta_0)}{\partial d_1} - G_R(\theta_0)\frac{\partial T_A(\theta_0)}{\partial d_1} \\ \vdots \\ T_A(\theta_0)\frac{\partial G_R(\theta_0)}{\partial d_{N-1}} - G_R(\theta_0)\frac{\partial T_A(\theta_0)}{\partial d_{N-1}} \end{bmatrix} = 0 \tag{11}$$

The solution of above equations would yield the optimum spacing of the array elements that will minimize the transmitter power required to receive the signal while maintaining a constant SNR at the receiver.

4 Numerical Results

We now perform numerical simulation for the parameters of satellite link as mentioned in Table 1 [27]. The model of atmosphere considered is taken as a uniform layer of 922 homogeneous layers as shown in [24] and the mean annual global atmosphere is considered for calculating the losses over the earth-space link as given in [28]. We assume that both the Eve and Bob are close to each other so that main beam of their antennas make the same elevation angle with the satellite. The close placement of Even and Bob receiver is the worst case scenario from the link security point of view because as Eve moves farther away from Bob, the signal attenuation will be larger due to the slant path and highly directional beam of antenna and so Eve will have even lower SNR. Bob uses the optimized phased array with non-uniformly placed elements while Eve use a uniform phased array with spacing of $\lambda/2$ which is the usual practice. In order to provide them with similar scenario, both use the same number of elements of phased array. For maintaining a bit-error rate (BER) at the receiver of Bob, we employ binary PSK with soft decision convolution coding with a code rate of 2/3. For an SNR of 5 dB at the receiver of Bob, the BER would be 1.63×10^{-9}. We maintain this SNR_{Bob} at 5 dB as the signal AoA changes by tuning the satellite transmitter power P_T. We minimize the transmitter power by employing the non-uniform phased array for the receiver of Bob. As Eve uses a uniform phased array with uniform spacing of $\lambda/2$, the SNR_{Eve} at Eve will be lower for the same transmitter power P_T and hence the higher BER. We plot the minimized transmitted power P_T as the AoA changes in Fig. 2. The plot of BER of Eve is shown in the Fig. 3 while the BER of Bob remains at 1.63×10^{-9}.

In order to increase the cybersecurity of the satellite link, we designate only a specific narrow range of AoA to communicate with Bob at the ground station. Assuming a BER threshold of 10^{-6}, we can observe from Fig. 3 that Alice can transmit the encryption key to Bob when the AoA is between $0^0 - 10^0$ and

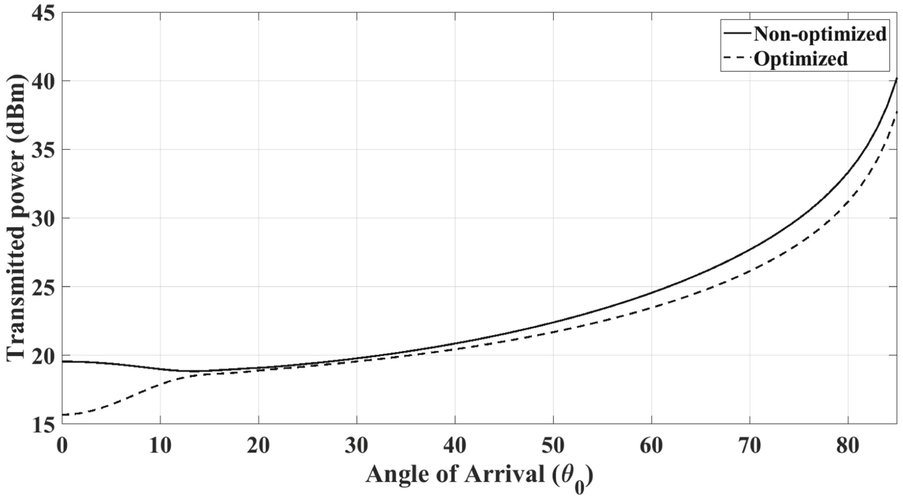


Fig. 2. Minimized transmitter power using non-uniform phased array at the receiver of Bob to maintain a constant BER with $SNR_{Bob} = 5$ dB

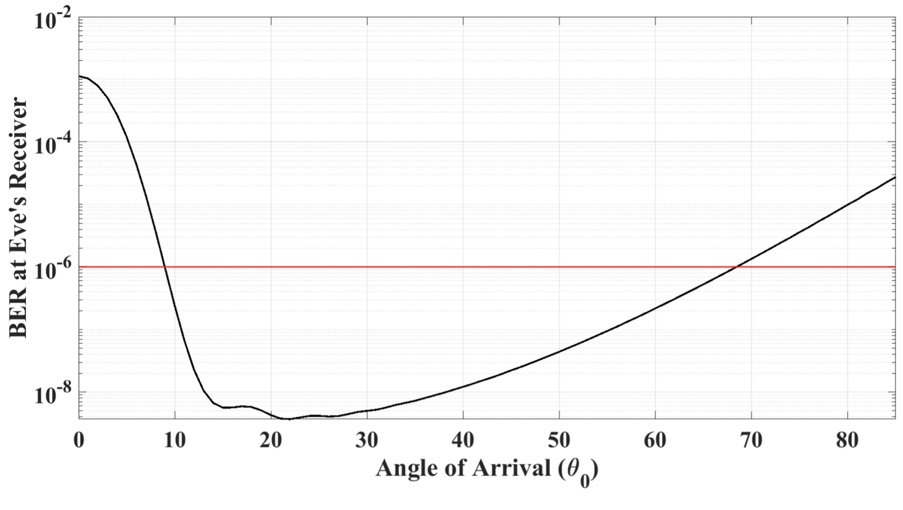


Fig. 3. BER at the receiver of Eve as the transmitter power of satellite is minimized as the AoA changes, the BER at the receiver of Bob is 1.63×10^{-9}

$70^0 - 85^0$. For these range of AoA, the BER of Eve will be very high and SNR will be be very low so that Bob will hardly be able to notice the transmission between Alice and Bob, that is a very much desirable scenario for the secure link. We employ these range of AoA i.e. $0^0 - 10^0$ and $70^0 - 85^0$ to communicate the encryption key to the Bob while making it practically difficult for Eve to receive with significant fidelity as the probability of error would be very high.

Table 1. Simulation parameters

Definition	Symbol	Value
Satellite antenna gain	G_T	70 dBi
Satellite altitude	h_s	1000 km
Atmospheric height	H	100 km
Ground surface temperature	T_G	290 K
Emissivity	ϵ	0.95
Effective reflection coefficient	ρ_g	0.05
Feeder loss	L_F	0.1 dB
Feeder temperature	T_F	300 K
LNA noise figure		3 dB
LNA gain	G_{LNA}	15 dB
Mixer temperature	T_{IF}	150 K
Mixer gain	G_{MX}	−10 dB
IF amplifier noise figure		5 dB
IF amplifier gain	G_{IF}	25 dB
Number of array elements	N	50
Array weights	$a_1, \ldots, a_N$	1
System bandwidth	B	1 GHz

5 Conclusion

In this work, we have proposed a novel method to enhance the cybersecurity of the satellite link with the ground station for the next generation of communication system. The method is based on shaping the gain pattern of phased array receiver of Bob at the ground station so that transmitted signal power will be minimized while the SNR of Eve worsens to degrade its BER performance. With this minimum transmitter power, we can send the encryption key to Bob only when there is a window of opportunity in terms of AoA such that BER of Eve is very high while the BER of Bob remains low in order to maintain the fidelity of the link with Alice. The AoA for which the sharing of encryption key between Alice and Bob would be done covertly is in the range from $0^0 - 10^0$ and $70^0 - 85^0$.

References

1. Tataria, H., Shafi, M., Molisch, A.F., Dohler, M., Sjöland, H., Tufvesson, F.: 6G wireless systems: vision, requirements, challenges, insights, and opportunities. In: Proceedings of the IEEE, pp. 1–34 (2021)
2. Akyildiz, I.F., Kak, A., Nie, S.: 6G and beyond: the future of wireless communications systems. IEEE Access **8**, 133995–134030 (2020)
3. Shlomi, A., Kupferman, J.: Effects of weather on drone to IoT QKD. In: Cyber Security Cryptography and Machine Learning, CSCML (2019)

4. Manulis, M., Bridges, C.P., Harrison, R., Sekar, V., Davis, A.: Cyber security in New Space. Int. J. Inf. Secur. **20**(3), 287–311 (2020). https://doi.org/10.1007/s10207-020-00503-w
5. Saha, S.S., Rahman, S., Ahmed, M.U., Aditya, S.K.: Ensuring cybersecure telemetry and telecommand in small satellites: recent trends and empirical propositions. IEEE Aerosp. Electron. Syst. Mag. **34**(8), 34–49 (2019)
6. Libicki, M.C.: Correlations between cyberspace attacks and kinetic attacks. In: 2020 12th International Conference on Cyber Conflict (CyCon), vol. 1300, pp. 199–213 (2020)
7. Liu, Y., Yu, F.R., Li, X., Ji, H., Leung, V.C.: Blockchain and machine learning for communications and networking systems. IEEE Commun. Surv. Tutor. **22**(2), 1392–1431 (2020)
8. Xue, M., Yuan, C., Heyi, W., Zhang, Y., Liu, W.: Machine learning security: threats, countermeasures, and evaluations. IEEE Access **8**, 74720–74742 (2020)
9. Dang, S., Amin, O., Shihada, B., Alouini, M.-S.: What should 6G be? Nat. Electron. **3**, 20–29 (2020)
10. Chen, H., Tu, K., Li, J., Tang, S., Li, T., Qing, Z.: 6G wireless communications: security technologies and research challenges. In: 2020 International Conference on Urban Engineering and Management Science (ICUEMS), pp. 592–595 (2020)
11. Hao, J., Mithun, M., Jie, Z., Jaime, L.: Channel modeling and characteristics for 6G wireless communications. IEEE Netw. **35**(1), 296–303 (2021)
12. Federici, J., Moeller, L.: Review of terahertz and subterahertz wireless communications. J. Appl. Phys. **107**(111101), 1–23 (2010)
13. Ma, J.: Security and eavesdropping in terahertz wireless links. Nature **563**, 89–93 (2018)
14. Tekbıyık, T., Ekti, A.R., Kurt, G.K., Gorcinad, A.: Terahertz band communication systems: challenges, novelties and standardization efforts. Phys. Commun. **35**(100700), 1–18 (2019)
15. Arnon, S.: Quantum technology for optical wireless communication in data-center security and hacking. In: Dingel, B.B., Tsukamoto, K., Mikroulis, S. (eds.) Broadband Access Communication Technologies XIII, vol. 10945, pp. 97–101. International Society for Optics and Photonics, SPIE (2019)
16. Gabay, M., Arnon, S.: Quantum key distribution by a free-space MIMO system. J. Lightwave Technol. **24**(8), 3114–3120 (2006)
17. Caparra, G., Curran, J.T.: On the achievable equivalent security of GNSs ranging code encryption. In: 2018 IEEE/ION Position, Location and Navigation Symposium (PLANS), Monterey, CA (2018)
18. Burg, A., Chattopadhyay, A., Lam, K.Y.: Wireless communication and security issues for cyber-physical systems and the internet-of-things. In: Proceedings of the IEEE (2018)
19. Porambage, P., Gür, G., Osorio, D.P.M., Liyanage, M., Gurtov, A., Ylianttila, M.: The roadmap to 6G security and privacy. IEEE Open J. Commun. Soc. **2**, 1094–1122 (2021)
20. ASlimani, A., Bennani, S.D., El Alami, A.: Effect of inter-elements distance and phase shift excitation on radiation performance of linear, planar and circular arrays antennas. In: 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), pp. 1–7 (2017)
21. Petrov, V., Komarov, M., Moltchanov, D., Jornet, J.M., Koucheryavy, Y.: Interference and SINR in millimeter wave and terahertz communication systems with blocking and directional antennas. IEEE Trans. Wireless Commun. **16**(3), 1791–1808 (2017)

22. Kokkoniemi, J., Jornet, J.M., Petrov, V., Koucheryavy, Y., Juntti, M.: Channel modeling and performance analysis of airplane-satellite terahertz band communications. IEEE Trans. Veh. Technol. **70**(3), 2047–2061 (2021)
23. Balanis, C.A.: Antenna Theory: Analysis and Design, 4th edn. Wiley (2016)
24. ITU. ITU Recommendation: Attenuation by atmospheric gases and related effects. P. 676–12, August 2019
25. Sturdivant, R.L., Chong, E.K.P.: Systems engineering of a terabit elliptic orbit satellite and phased array ground station for IoT connectivity and consumer internet access. IEEE Access **4**, 9941–9957 (2016)
26. Mailloux, R.J.: Phased Array Antenna Handbook, 3rd edn. Artech House (2017)
27. Kumar, R., Arnon, S.: SNR optimization for LEO satellite at sub-THz frequencies. IEEE Trans. Antennas Propag. 1 (2022)
28. ITU. ITU Recommendation: Reference standard atmospheres, pp. 835–6, July 2017

Polynomial Approximation of Inverse sqrt Function for FHE

Samanvaya Panda(✉)

International Institute of Information Technology, Hyderabad, Hyderabad, India
samanvaya.panda@research.iiit.ac.in

Abstract. Inverse sqrt and sqrt function have numerous applications in linear algebra and machine learning such as vector normalisation, eigenvalue computation, dimensionality reduction, clustering, etc. This paper presents a method to approximate and securely perform the inverse sqrt function using CKKS homomorphic encryption scheme. Since the CKKS homomorphic scheme allows only computation of polynomial functions, we propose a method to approximate the inverse sqrt function polynomially. In the end, we provide an implementation of our method for the inverse sqrt function.

Keywords: Polynomial approximation · Inverse sqrt · Homomorphic encryption · CKKS

1 Introduction

Privacy-preserving computation has been used to mitigate personal and sensitive information leakage problems. There are many ways to compute functions on data securely, keeping the data private. One such technique is homomorphic encryption. Homomorphic encryption allows us to evaluate any function on encrypted data without knowing the actual data. In recent times, there have been numerous advancements in homomorphic encryption schemes. The CKKS homomorphic encryption scheme proposed recently by Cheon et al. in [3] is one such example. It supports arithmetic on floating-point numbers which is why it has been extensively used for different machine learning tasks [1,5,6].

The problem with CKKS homomorphic scheme is that it only supports polynomial operations. So, we can't implement many non-polynomial functions such as logarithm, sqrt, sine, etc. directly. These functions need to be polynomially approximated. There have been several methods proposed to approximate non-polynomial functions, such as using Chebyshev's polynomial basis [13], minimax polynomials [12], Fourier series, etc. in [2–4,14]. But in most FHE-based algorithms using schemes similar to CKKS, the approximation of inverse functions is skipped and such computations are performed on plaintext. Few attempts have been made in this direction in [14] and [9]. In [14], authors suggested approximating division operation using Newton's method and Goldschmidt's algorithm. They used an initial guess $y_0 = 2^{1-k}$ for all values. In [9] too, the author used

Newton's method and Goldschmidt's algorithm to approximate the inverse sqrt and sqrt function simultaneously. They took inspiration from the fast inverse sqrt algorithm [7] and tried a similar approach in a homomorphic setting. Instead of fixing the initial guess, they used constrained linear regression to approximate the inverse sqrt function. The line served as a good initial guess for larger values of x. But for smaller values, it requires more iterations for convergence.

1.1 Contributions

We also draw inspiration from the fast inverse sqrt algorithm [7]. We can have some curve approximating the $\frac{1}{\sqrt{x}}$ as a good initial guess and then use Newton's iteration to improve upon that guess. This paper proposes a new method to find a better initial guess for the inverse sqrt function and apply Newton's iterations. The advantage of using Newton's iterations for inverse functions is that they are polynomial and can be evaluated homomorphically. Upon observing the shape of the inverse sqrt function, we notice that the value of the function keeps decreasing slowly to the right of $x = 1$ and increases drastically to the left of $x = 1$. Over a considerable interval, we can see that the shape of the function looks like the 'L' alphabet.

This gave us the intuition to approximate the curve $\frac{1}{\sqrt{x}}$ using two lines. One of the lines approximates the curve over a large interval capturing the *slow* decreasing trend of the values. The other line captures the *rapid* increasing trend in the values of the curve. The intersection of the two lines is called the pivot point. Approximation of the curve $\frac{1}{\sqrt{x}}$ can be written as a convex combination of the two lines about the pivot point using a sign function as described in [4]. In sections ahead, we will define how to find those two lines and what properties they must satisfy. Our method provides sufficient accuracy with multiplicative depth[1] comparable to the approximation in [9] and also reduces the number of iterations almost by half. Finally, we provide experimental results on the homomorphic implementation of our method.

2 Preliminaries

2.1 CKKS Homomorphic Scheme

The CKKS(Cheon-Kim-Kim-Song) scheme [3] is a leveled homomorphic encryption scheme. Unlike other HE schemes, CKKS supports approximate arithmetic on real and complex numbers with predefined precision. The main idea behind the CKKS scheme is that it treats noise generated upon decryption as an error in computation for real numbers. This makes it ideal for performing machine learning tasks where most of the calculations are approximate. The CKKS scheme becomes an FHE(fully homomorphic encryption) scheme with the bootstrapping technique.

[1] We consider non-scalar multiplicative depth i.e. ciphertext-ciphertext multiplication.

Let $N = \phi(M)$ be the degree of the M-th cyclotomic polynomial $\Phi_M(X)$. If N is chosen as a power of 2 then $M = 2N$ and the M-th cyclotomic polynomial $\Phi_M(X) = X^N + 1$. Let $\mathcal{R} = \mathbb{Z}[X]/\Phi_M(X) = \mathbb{Z}[X]/(X^N + 1)$ be the ring of polynomials defined for the plaintext space. Let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(X^N + 1)$ be the residue ring defined for the ciphertext space. Let $\mathbb{H}$ be a subspace of $\mathbb{C}^N$ which is isomorphic to $\mathbb{C}^{N/2}$. Let $\sigma : \mathcal{R} \to \sigma(\mathcal{R}) \subseteq \mathbb{H}$ be a canonical embedding. Let $\pi : \mathbb{H} \to \mathbb{C}^{N/2}$ be a map that projects a vector from a subspace of $\mathbb{C}^N$ to $\mathbb{C}^{N/2}$.

The CKKS scheme provides the following operations:-

- **KeyGen**(N) :- Generates secret polynomial $s(X)$, public polynomial $p(X)$.
- **Encode**(z) :- Encodes a message vector $z \in \mathbb{C}^{N/2}$ to a message polynomial $m(X) \in \mathcal{R}$ where $m(X) = \sigma^{-1}(\lfloor \Delta \cdot \pi^{-1}(z) \rceil) \in \mathcal{R}$.
- **Decode**$(m(X))$:- Decodes a message polynomial $m(X) \in \mathcal{R}$ back to a message vector $z \in \mathbb{C}^{N/2}$.
- **Encrypt**$(m(X), p(X))$:- Encrypts the message polynomial $m(X) \in \mathcal{R}$ to get ciphertext polynomial $c(X) = (c_0(X), c_1(X)) = (m(X), 0) + p(X) = (m(X) - a(X) \cdot s(X) + e(X), a(X)) \in (\mathbb{Z}_q[X]/(X^N + 1))^2$.
- **Decrypt**$(c(X), s(X))$:- Decrypts the ciphertext polynomial $c(X)$ to the corresponding message polynomial $m(X)$.

Apart from the above operations, it provides an evaluator function that can perform specialised ciphertext operations. This include:-

- **Add**$(c(X), c'(X))$:- to add 2 ciphertext polynomials.
- **Multiply**$(c(X), c'(X))$:- to multiply 2 ciphertext polynomials.
- **Rotate**$(c(X), i)$:- to rotate the ciphertext polynomial by i positions left.

2.2 Polynomial Approximation of Sign Function

The sign function is non-polynomial and can't be used directly in the CKKS scheme. So, we use a polynomial approximation of the sign function instead. In general, we approximate the sign function in the domain $x \in [-1, 1]$ and the sign of any other value can be found by scaling it inside the domain. In this paper, we will use the approximation proposed by Cheon et al. in [4]. We approximate the sign function as a composite polynomial $f^{(d)}$ where f is a polynomial with *similar shape* to the sign function in the interval $[-1, 1]$. The properties satisfied by f are:-

- $f(-x) = -f(x)$ (Origin symmetry)
- $f(1) = 1, f(-1) = -1$ (Range of sign function)
- $f'(x) = c(1 - x)^n(1 + x)^n$ for some $c > 0$ (Faster convergence)

Evaluating the polynomial f for different values of n we obtain :-

$$f_n(x) = \sum_{i=0}^{n} \frac{1}{4^i} \cdot \binom{2i}{i} \cdot x(1 - x^2)^i$$

Theorem 1. *If* $d \geq \frac{1}{log(c_n)} \cdot log(1/\epsilon) + \frac{1}{log(n+1)} \cdot log(\alpha - 1) + O(1)$, *then* $f_n^{(d)}(x)$ *is an* (α, ϵ)*-close polynomial to sgn(x) over* $[-1, 1]$ *implies* $|f_n^{(d)}(x) - sgn(x)| \leq 2^{-\alpha}$ *where* $x \in [-1, -\epsilon] \cup [\epsilon, 1]$.

Proof of Theorem 1 can be found in [4]. To speedup up the convergence further, we use a polynomial g instead of f that has a larger derivative at $0(c_n)$. The polynomial g would be a minimax polynomial satisfying the following properties :-

- $g(-x) = -g(x)$ (Origin Symmetry)
- $\exists\, 0 < \delta < 1$ s.t. $x < g(x) < 1\ \forall x \in (0, \delta]$ and $g([\delta, 1]) \subseteq [1 - \tau, 1]$

Its hard to represent g in closed form and is evaluated using Algorithm 2 in [4]. In this paper, we use $n = 3$ to approximate the sign function. The approximate sign function is computed as a composition $f_3^{d_f}(x) \circ g_3^{d_g}(x)$ where $d_g = \frac{1}{2log(c_n)} \cdot log(1/\epsilon) = 0.445 \cdot log(1/\epsilon)$ and $d_f = \frac{1}{log(n+1)} \cdot log(\alpha - 1) = 0.5 \cdot log(\alpha - 1)$. The polynomial $f_3(x)$ and $g_3(x)$ are:-

$$f_3(x) = \frac{1}{2^4}(35x - 35x^3 + 21x^5 - 5x^7)$$
$$g_3(x) = \frac{1}{2^{10}}(4589x - 16577x^3 + 25614x^5 - 12860x^7)$$

2.3 Inverse sqrt Approximation

In [9], the author mentioned a method to polynomially approximate $\frac{1}{\sqrt{x}}$ that could be used in FHE schemes. It first uses a line as an initial guess and then uses Newton's and Goldschmidt's algorithms to improve upon their guess. Goldschmidt's algorithm is used to compute $\sqrt{x}$ alongside with $\frac{1}{\sqrt{x}}$ which is required for their application. For the initial guess, they perform a linear approximation of $\frac{1}{\sqrt{x}}$ by formulating a constrained linear regression. It is formulated as the following minimization problem :-

$$\begin{aligned} &\min_{w} \quad \frac{1}{n}\sum_{i=1}^{n}(y_i - w^T x_i)^2 \\ &\text{subject to} \quad w^T x_i \geq 0 \quad \forall\, i = \{1, 2, \cdots n\} \end{aligned} \tag{1}$$

3 Approximation of $\frac{1}{\sqrt{x}}$

As mentioned before, we use Newton's method to approximate the value of $y = \frac{1}{\sqrt{x}}$. It is because in each iteration of Newton's method, the update equation is polynomial in nature. Let $f(x) = y^{-2} - x$. Then the update equation for $f(x)$ is :- $y_{i+1} = \frac{y_i}{2}(-xy_i^2 + 3)$. The only thing now to consider is the initial guess for each value.

3.1 A Good Initial Guess

A good initial guess for Newton's update equation would mean faster convergence[2]. So, a good initial guess would be a good approximation of the $\frac{1}{\sqrt{x}}$ function. Another thing to remember is that the initial guess must guarantee convergence. The range of values for which y_i guarantees convergence would be $\frac{y_i}{2}(-xy_i^2+3) > 0 \implies 0 < y_i < \sqrt{\frac{3}{x}}$.

Any value of y_i between 0 and $\sqrt{\frac{3}{x}}$ will eventually converge because the term $-xy_i^2+3$ is always greater than 0 pushing it towards the value $\frac{1}{\sqrt{x}}$. But, we wish that our algorithm would converge in a fixed number of iterations rather than converging eventually. So, we observe that for any x, the number of iterations needed for the initial guess $y_0 \in [\frac{1}{\sqrt{x}}, \sqrt{\frac{3}{x}}]$ to converge increases as we move away from $\frac{1}{\sqrt{x}}$ to $\sqrt{\frac{3}{x}}$. Same is the case for the interval $[0, \frac{1}{\sqrt{x}}]$. For a given number of iterations, we could use binary search on both of the intervals to find the new reduced range for the initial guess that guarantees convergence. To keep things simple and uniform, we assume that the reduced range of initial guess for each x would also be a function of $\frac{1}{\sqrt{x}}$ and the new range would be $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ where $0 < k_1 < 1$ and $1 < k_2 < \sqrt{3}$. Using Lemma 1, we show that for constants k_1 and k_2, we can guarantee convergence for all values of x with a certain error.

Lemma 1. *Let d be the given number of Newton's iterations. Let the absolute error at the point $x = 1$ for the initial guess $y_0 = k_1$ or $y_0 = k_2$ after d iterations be $\leq \mathcal{E}$ where $0 < k_1 < 1$ and $1 < k_2 < \sqrt{3}$. Then the absolute error at any point x after d iterations for any initial guess in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ would be $\mathcal{E}_x \leq \frac{\mathcal{E}}{\sqrt{x}}$.*

Proof. Let us first consider the lower bound k_1. At point $x = 1$, we have $y_0 = k_1$. Then, $y_1 = \frac{k_1}{2} \cdot (-k_1^2 + 3)$. Let's say $K_0 = k_1$ and $K_i = \frac{K_{i-1}}{2} \cdot (-K_{i-1}^2 + 3)$. After d iterations we would have $|1 - K_d| \leq \mathcal{E}$. Now for any x, $y_0 = \frac{k_1}{\sqrt{x}}$, $y_1 = \frac{y_0}{2} \cdot (-xy_0^2 + 3) = \frac{k_1}{2\sqrt{x}} \cdot (-k_1^2 + 3) = \frac{K_1}{\sqrt{x}}$. So, after d iterations we get $\mathcal{E}_x = |\frac{1}{\sqrt{x}} - \frac{K_d}{\sqrt{x}}| \leq \frac{\mathcal{E}}{\sqrt{x}}$. Since values of k_1 and k_2 are evaluated for fixed $\mathcal{E}$, we can similarly argue for the upper bound k_2 that it will guarantee convergence for any x with an absolute error $\frac{\mathcal{E}}{\sqrt{x}}$.

Corollary 1. *The mean absolute error over all x in the interval $[a, b]$ after d iterations would be $\bar{\mathcal{E}} = \frac{2\mathcal{E}}{\sqrt{a}+\sqrt{b}}$*

Corollary 2. *If we consider the two intervals $x \in [a, 1] \cup [1, b]$, where a, b are constants and $a < 1 < b$ then the mean absolute error over all x after d iterations would be $\bar{\mathcal{E}}' = \bar{\mathcal{E}_1} + \bar{\mathcal{E}_2} = \frac{\mathcal{E}}{1+\sqrt{a}} + \frac{\mathcal{E}}{1+\sqrt{b}}$.*

The Corollaries 1 and 2 can be easily verified using Mean value theorem i.e. $\int_a^b f(x)dx = f(c)(b-a)$. The value of k_1 and k_2 is found using the Algorithm 1 at $x = 1$ for a fixed number of Newton's iterations d and absolute error $\mathcal{E}$.

[2] By convergence we mean that the difference between the actual and predicted value is bounded by some predefined error.

Algorithm 1. Finding constants for initial guess range of $\frac{1}{\sqrt{x}}$

```
Input: $d, \mathcal{E}$: No.of iterations and error.
Output: $k_1$, $k_2$: Upper and lower bound of initial guess.
 1: $l \leftarrow 2\delta - 1$, $r \leftarrow 1$
 2: while $r - l \geq \delta$ do
 3:     $mid \leftarrow (r + l)/2$
 4:     $val \leftarrow mid$
 5:     for $i = 1$ to $d$ do
 6:         $val \leftarrow \frac{val}{2}(-val^2 + 3)$
 7:     end for
 8:     $diff \leftarrow |val - 1|$
 9:     if $diff \leq \mathcal{E}$ then
10:         $r \leftarrow mid$
11:     else
12:         $l \leftarrow mid + \delta$
13:     end if
14: end while
15: $k_1 \leftarrow l$
16: $l \leftarrow 0$, $r \leftarrow 2\sqrt{3} - 1$
17: while $r - l \geq \delta$ do
18:     $mid \leftarrow (r + l)/2$
19:     $val \leftarrow mid$
20:     for $i = 1$ to $d$ do
21:         $val \leftarrow \frac{val}{2}(-val^2 + 3)$
22:     end for
23:     $diff \leftarrow |val - 1|$
24:     if $diff \leq \mathcal{E}$ then
25:         $l \leftarrow mid$
26:     else
27:         $r \leftarrow mid - \delta$
28:     end if
29: end while
30: $k_2 \leftarrow l$
31: return $k_1, k_2$
```

3.2 2-Line Approximation

Now that we have the range for initial guess, we need to find an approximation of $\frac{1}{\sqrt{x}}$ that lies within this initial guess for all values of x. As mentioned earlier, we approximate the function $\frac{1}{\sqrt{x}}$ using two intersecting lines(L_1, L_2) by limiting the domain of x to $[a, b]$. The intersection point of the lines is called the *pivot* point(denoted as P). Let L_2 approximate $\frac{1}{\sqrt{x}}$ on larger values of x i.e. $x \in [P, b]$ and L_1 approximate $\frac{1}{\sqrt{x}}$ on the smaller values of x i.e. $x \in [a, P]$. The overall approximation of $\frac{1}{\sqrt{x}}$ can be written as convex combination in terms of L_1 and L_2 as:-

$$h(x) = (1 - \beta(x)) \cdot L_1(x) + \beta(x) \cdot L_2(x) \tag{2}$$

where $\beta(x) = comp(\frac{P}{b-a}, \frac{x}{b-a})$ and $comp(x, y) = \frac{1+sgn(x-y)}{2}$. We can evaluate the sign function polynomially as mentioned in [4]. So far, we have mentioned how to compute the approximate value of $\frac{1}{\sqrt{x}}$ using L_1 and L_2. Now we are going to discuss how to find these lines. Remember that approximate value of $\frac{1}{\sqrt{x}}$ for all x must be in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ to guarantee convergence. To ensure this, the maximum value for any point x on the line cannot exceed $\frac{k_2}{\sqrt{x}}$. That means both the lines are tangent to the curve $\frac{k_2}{\sqrt{x}}$. Similarly, the minimum value at any point x on the line cannot go below $\frac{k_1}{\sqrt{x}}$. This implies that the extreme points of both the lines must either lie on the curve $\frac{k_1}{\sqrt{x}}$ or above it.

Let x_1, x_2 be the points were lines L_1, L_2 are tangent to the curve $\frac{k_2}{\sqrt{x}}$ respectively. The slope of any tangent to the curve $\frac{k_2}{\sqrt{x}}$ at point γ is $-\frac{1}{2}k_2 \cdot \gamma^{-3/2}$. So, the equation of L_1 and L_2 will be:-

$$L_1 : y = -\frac{1}{2}k_2 x_1^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_1}} \tag{3}$$

$$L_2 : y = -\frac{1}{2}k_2 x_2^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_2}} \tag{4}$$

The last step in figuring out the lines L_1, L_2 are the points x_1, x_2 respectively. They are the points where the lines L_1 and L_2 touch the curve $\frac{k_2}{\sqrt{x}}$. We consider that these lines must pass through the extreme points of the domain of x i.e. L_2 must pass through the point $x = b$ and L_1 must pass through $x = a$ on the curve $\frac{k_1}{\sqrt{x}}$. Each of the lines L_1, L_2 also intersect the curve $\frac{k_1}{\sqrt{x}}$ at points different than $x = a$ and $x = b$ respectively. Let the line L_2 pass through a point x on the curve $\frac{k_1}{\sqrt{x}}$. Then the equation of L_2 becomes:-

$$\frac{k_1}{\sqrt{x}} = -\frac{1}{2}k_2 x_2^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_2}}$$

$$\implies k_2^2 x^3 - 6k_2^2 x^2 x_2 + 9k_2^2 x x_2^2 - 4k_1^2 x_2^3 = 0 \tag{5}$$

Solving the Eq. 5 with $x = b$, we would obtain the point x_2 and ultimately L_2. Similarly, we can evaluate x_1 at $x = a$ and get L_1.

3.3 Finding Pivot Point

In the previous section, we presented a method to obtain the lines L_1, L_2 so that they lie in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$. But we aim to approximate the original function $\frac{1}{\sqrt{x}}$ using two intersecting lines in the given range. The above approach for finding the lines doesn't guarantee the intersection of lines L_1 and L_2 inside the desired range i.e. P may or may not lie above or on the curve $\frac{k1}{\sqrt{x}}$. One way to ensure that the pivot point lies within the range of convergence is to increase the number of newton's iterations. Increasing the iterations would adjust the values of k_1 and k_2 accordingly, allowing L_1 and L_2 to intersect in the inside region. Another way to ensure that is by tweaking the process of finding the lines L_1, L_2 a little bit. Instead of increasing the number of iterations to adjust the values of k_1 and k_2, we fix the pivot point. We follow the following steps:-

- **Step 1:** Find x_2 using Eq. 5 at $x = b$ to evaluate L_2.
- **Step 2:** Find the other point of intersection of line L_2 with the curve $\frac{k_1}{\sqrt{x}}$ by solving for x in Eq. 5. Now this point becomes the pivot point P.
- **Step 3:** Find x_1 using the pivot point. Substitute $x_2 = x_1$ and $x = P$ in Eq. 5 and solve for x_1.

Note that if we follow the above steps to find lines L_1 and L_2, then they will always intersect at point P. But, the line L_1 no longer intersects the curve

$\frac{k_1}{\sqrt{x}}$ at $x = a$ but rather at $x = a'$ where $a < a'$. This is a trade-off between accuracy and the number of iterations. With practical results, we can argue that a sufficient level of accuracy can be achieved with a few iterations using the above method. Another fact to consider is that we choose to fix the pivot point as the point of intersection of the line L_2 to the curve $\frac{k_1}{\sqrt{x}}$ instead of L_1. This is because the curve $\frac{k_1}{\sqrt{x}}$ is closer to 0 on larger values of x. If we take the pivot point as the other point of intersection of line L_1 then line L_2 would intersect the curve $\frac{k_1}{\sqrt{x}}$ at $x = b'$ where $b' < b$. So, for some values of $x > b'$, the value of L_2 would be negative. When we apply Newton's iterations on these points, there values would converge to $-\frac{1}{\sqrt{x}}$ instead of $\frac{1}{\sqrt{x}}$.

4 Implementation Details

We implemented the secure inverse sqrt approximation using the SEAL library [11] for CKKS homomorphic scheme. The implementation can be found at [10]. For the approximate sign function, we fix the value of $d_f = 2$ which would give us the value of $\alpha = 17$. We know that the value of $d_g = 0.445 \cdot log(1/\epsilon)$ where the value of ϵ determines the precision of values for the *comp*() function. For any $z_1, z_2 \in [0, 1], |z_1 - z_2| \geq \epsilon$. The lower the value of ϵ, larger the precision, better accuracy of the approximation and larger the multiplicative depth. For our experiments mentioned in Table 1, we fixed $d_g = 7$ making $\epsilon \approx 2 \times 10^{-5}$. The multiplicative depth for both f_3 and g_3 is 3. So, the multiplicative depth required to compute the initial guess would be $3(d_g + d_f) + 1$. The maximum multiplicative depth required in a single Newton's iteration is 2. Hence, the maximum multiplicative depth required to compute the inverse operation would be $3(d_g + d_f) + 2d + 1$. While evaluating the function on encrypted data, we observed that the output of *comp*() was slightly > 1 due to additive noise. So, the points near the tangent exceeded the upper bound. To mitigate this, we introduced a constant error i.e. $err = 8.5 \times 10^{-7}$ that was subtracted from the value of the *comp*() so that the final value remained < 1. So on encrypted data, the convex combination of lines becomes - $h(x) = (1 + err - \beta(x)) \cdot L_1(x) + (\beta(x) - err) \cdot L_2(x)$.

To obtain the parameters of pivot-tangent method, we first fix the interval $[a, b]$ and then fix the values of number of iterations d and absolute error $\mathcal{E}$. The smaller the value of $\mathcal{E}$ the smaller the interval for initial guess $[\frac{k1}{\sqrt{x}}, \frac{k2}{\sqrt{x}}]$ would be. To increase the initial guess interval, we have to increase the number of iterations d. So, smaller $\mathcal{E}$ requires larger number of iterations d. We should keep this in mind while fixing d and $\mathcal{E}$. Also, notice that Eq. 5 is a cubic equation. So, while solving for x_2, we consider the largest root as x_2. Similarly while computing x_1, we consider the smallest root as x_1. For the pivot point P, we consider the root closest to 1 as P. The combined method for the polynomial approximation of $\frac{1}{\sqrt{x}}$ using pivot-tangent method is given in the Algorithm 2.

Algorithm 2. Finding approximate value of $\frac{1}{\sqrt{x}}$

```
Input: $[a, b], \mathcal{E}, d, d_g, d_f, x, err$.
Output: $y_d$: Approximate value of $\frac{1}{\sqrt{x}}$.
1: Find $k_1, k_2$ using Algorithm 1.
2: Find $x_2, P, x_1$ using pivot-tangent method.
3: Compute $\beta(P, x)$ and $y_0 = h(x)$.
4: Compute $d$ Newton's iterations to obtain $y_d$.
5: return $y_d$
```

5 Results and Comparison

Table 1 summarizes the value of various parameters computed using the pivot-tangent method with $[a, b] = [10^{-4}, 10^3]$ and $\mathcal{E} = 0.007$. The points are equally divided in two intervals i.e. $[10^{-4}, 1]$ and $[1, 10^3]$. Using Corollary 2, we get the theoretical upper bound on error $= \frac{0.007}{1+10^{-2}} + \frac{0.007}{1+\sqrt{1000}} = 0.00714$. We observe that the mean absolute error obtained after the experiments for fixed parameters is lower than theoretical upper bound. Figure 1 shows different components of the pivot tangent method.

Table 1. Values of different parameters for given no.of Newton's iterations

d	k_1	k_2	x_1	x_2	P	Mean Abs. Error (without encryption)	Mean Abs. Error (with encryption)	Depth
7	0.128	1.6645	0.3111	343.6645	0.9053	0.00265	0.0055	42
8	0.0855	1.6876	0.1322	340.035	0.3887	0.00083	0.00112	44
9	0.0702	1.6958	0.0876	338.781	0.2587	0.000091	0.0001	46

Now, to compare our method with the technique of finding $\frac{1}{\sqrt{x}}$ mentioned in [9], we conducted some experiments by fixing the interval for $x \in [a, b]$. For the method in [9], we take d_1, d_2 as the number of Newton's and Goldschmidt's

Table 2. Comparison of our method with that of in [9].

$[a, b]$	Iteration		Depth		$\mathcal{E}$	err	Error	
	(d_1, d_2)	(d, d_g)	$2d_1 + 3d_2$	$2d + 3d_g + 6$			[9]	Ours
$[10^{-3}, 750]$	(12, 3)	(7, 5)	33	35	7e-3	1.2e-6	8.587e-5	2.23e-4
	(10, 5)	(7, 5)	35	35	1e-3	1.2e-6	8.695e-5	1.01e-4
	(12, 4)	(8, 6)	36	40	1e-4	1.2e-6	8.571e-6	2.23e-5
$[10^{-4}, 10^3]$	(12, 5)	(7, 6)	39	38	7e-3	1.2e-6	1.421e-3	5.65e-3
	(14, 4)	(8, 6)	41	40	7e-3	1.2e-6	2.323e-4	2.98e-3
	(15, 5)	(9, 7)	45	45	1e-4	8.5e-7	5.824e-6	1.87e-5

iterations respectively. For the method in [9], the values of (slope, intercept) used for the initial guess line in the intervals $[10^{-3}, 750]$ and $[10^{-4}, 10^3]$ are $(-0.00019703, 0.14777278)$ and $(-1.29054537e-04, 1.29054537e-01)$ respectively. From Table 2, we can see that for similar multiplicative depth, method mentioned in [9] has comparatively lower mean absolute error. Note that our method significantly reduces(almost half) the number of iterations required for convergence(by comparing $d_1 + d_2$ and d in Table 2). The biggest bottleneck for our method is the sign function approximation. It takes up more depth than the Newton's iterations i.e. $2d < 3(d_g + d_f)$. While the overall pivot-tangent method reduces the number of iterations required for convergence, it wastes most of its multiplicative depth in initially computing the sign approximation. Thus, when we try to compare it with method in [9] in terms of similar multiplicative depth, the method in [9] would get double the number of iterations compared to our method and hence has slightly better results. It is also important to remember that while method in [9] has no convergence guarantees(it may diverge for some values or converge to -$\frac{1}{\sqrt{x}}$ for larger x), our method provides guaranteed convergence with a certain error.

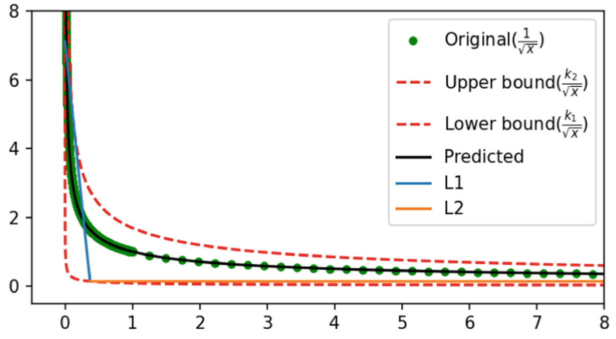


Fig. 1. Inverse sqrt approximation using pivot-tangent method

6 Conclusion and Future Work

In this paper, we presented the pivot-tangent method and approximated $\frac{1}{\sqrt{x}}$ function. Our approximation provides guaranteed convergence with sufficient accuracy compared to the previous method for the same multiplicative depth. Our method reduces the number of iterations required for convergence almost by half. However, the biggest bottleneck for our method is the computation of the approximate sign function. So, a new way to polynomially represent piece-wise lines without relying on sign function would significantly reduce our method's multiplicative depth. It is worth mentioning that in recent times there have been approaches such as Chimera[2] and Pegasus[8] that provide a bridge between different FHEs. This enables us to evaluate non-polynomial functions on CKKS ciphertexts. But these transformations are very costly in terms of memory. It would be better to have an inverse sqrt approximation in the CKKS scheme that provides sufficient accuracy and precision over a large interval. We also plan to extend our pivot-tangent method to approximate other inverse functions as a generalized approach. Now that we have a good approximation method of an inverse sqrt function, we also plan to apply this algorithm to different linear algebraic and machine learning algorithms in the future.

References

1. Boemer, F., Costache, A., Cammarota, R., Wierzynski, C.: Ngraph-he2: a high-throughput framework for neural network inference on encrypted data. In: Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography, pp. 45–56. WAHC 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338469.3358944
2. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: combining ring-LWE-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758 (2018). https://eprint.iacr.org/2018/758
3. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421 (2016). https://eprint.iacr.org/2016/421
4. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. Cryptology ePrint Archive, Report 2019/1234 (2019). https://ia.cr/2019/1234
5. Han, K., Hong, S., Cheon, J.H., Park, D.: Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662 (2018). https://eprint.iacr.org/2018/662
6. Lee, J.W., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. Cryptology ePrint Archive, Report 2021/783 (2021). https://ia.cr/2021/783
7. Lomont, C.: Fast inverse square root. Technical report Purdue University (2003). http://www.matrix67.com/data/InvSqrt.pdf
8. Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy (S&P), pp. 1057–1073. IEEE Computer Society, Los Alamitos, CA, USA, May 2021. https://doi.org/10.1109/SP40001.2021.00043
9. Panda, S.: Principal component analysis using CKKS homomorphic encryption scheme. Cyber Security Cryptography and Machine Learning, 5th International Symposium, CSCML 2021 (2021). https://eprint.iacr.org/2021/914
10. Panda, S.: Pivot-tangent method. https://github.com/pandasamanvaya/Pivot-tangent (2022)
11. Microsoft SEAL (release 3.7), Microsoft Research, Redmond, WA, September 2021. https://github.com/Microsoft/SEAL
12. Tasissa, A.: Function approximation and the Remez algorithm (2019)
13. Trefethen, L.N.: Approximation Theory and Approximation Practice. Extended Edition. SIAM (2019)
14. Çetin, G.S., Doröz, Y., Sunar, B., Martin, W.J.: Arithmetic using word-wise homomorphic encryption (2016)

Detecting Clickbait in Online Social Media: You Won't Believe How We Did It

Aviad Elyashar[1,2](✉), Jorge Bendahan[1,3], and Rami Puzis[1,3]

[1] Telekom Innovation Laboratories at Ben-Gurion University of the Negev, Beer-Sheva, Israel

[2] Department of Computer Science, Sami Shamoon College of Engineering, Beer Sheva, Israel
aviadel2@sce.ac.il

[3] Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel
jorgeaug@post.bgu.ac.il, puzis@bgu.ac.il

Abstract. This paper proposes a machine learning approach to detect clickbait posts published in social media. Clickbait posts are short, catchy phrases pointing into a longer online article. Users are encouraged to click on these posts to read the full article in many cases. The suggested approach differentiates between clickbait and legitimate posts based on training mainstream machine learning (ML) classifiers. The suggested classifiers are trained in various features extracted from images, linguistic, and behavioral analysis. For evaluation, we used two datasets provided by Clickbait Challenge 2017. The XGBoost classifier obtained the best performance with an AUC of 0.8, an accuracy of 0.812, a precision of 0.819, and a recall of 0.966. Finally, we found that counting the number of formal English words in the given content is helpful for clickbait detection.

Keywords: Clickbait detection · Social media · Machine learning

1 Introduction

In the past, offline media outlets, such as newspapers, were the primary information sources used to inform people. However, these traditional news outlets have been replaced with online resources in recent decades. We can attribute this change to the great diversity of options ranging from local, national, and international online media outlets to several niche blogs focusing on a specific area of interest offered [3]. This change is also attributed to the large numbers of readers using smart devices, which consume news online or using content generators, which provide users personalized news derived from a wide variety of news sources [11]. One of the reasons for this change is the nature of the online media website, which does not charge a fee for their services, as opposed to traditional media outlets [3]. As a result, online news rapidly replaces traditional media outlets [11].

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 377–387, 2022.
https://doi.org/10.1007/978-3-031-07689-3_28

Although online news provides numerous benefits, this domain suffers from a few problems. In most cases, the business model of online media websites is based on advertisements rather than subscribers' charges [3]. As a result, there is significant competition among the online media outlets for users' attention to increase their income. The techniques used include a high use of catchy headlines and attractive images.

In the last decade, one of the popular techniques online news websites use to increase user traffic includes massive publication of posts on online social networks (e.g., Facebook and Twitter). In most cases, each of these posts contains an attractive preview and a link pointing to the full article published on the given website [16]. These previews consist of a short headline, description, and image associated with the article's main topic. By clicking on the link attached, users are directed to the full article.

Alongside high accessibility, these short teasers cause problems for many users. In many cases, to increase traffic, such posts lure users into clicking and redirecting them to the online news website for reading an online news article using attractive headlines and images. However, many times users find these posts, also known as clickbait, as misleading, containing a gap between the clicked post and the full article [5]. An anticipated emotional reaction often characterizes it, as well as lack of knowledge (e.g., "15 surprising facts about Tesla cars you probably didn't know", and "Here's what people really thought about that Trump press conference"). In recent years, the use of clickbait even exacerbated and contributed to the rapid spread of rumors and misinformation online [5].

The reason associated with the success of clickbait is related to human behavior. In many cases, clickbait takes advantage of the cognitive phenomenon known as the Curiosity Gap [13]. Clickbait headlines provide referencing cues, which create curiosity among users. This curiosity encourages the readers to click on the link attached to address their knowledge gap [3].

The state-of-the-art solutions for automatic clickbait detection are currently based on machine learning (ML) techniques, yet many of these studies suffer low accuracy. This study proposes a method for detecting clickbait posts in social media based on an ML classifier capable of distinguishing between clickbait and legitimate posts published. The classifier is trained on various features extracted from images, linguistic analysis, and abuser detection. We used two datasets provided by Clickbait Challenge 2017 [31] to evaluate the proposed method. The contributions of this paper are as follows:

- We identified novel features that combine the information extracted from both posts and their associated news articles. These features were found significant in both datasets for detecting clickbait. To the best of our knowledge, we are the first to suggest features that use both components together;
- We found that the number of formal English words in a given content is useful for clickbait detection. To the best of our knowledge, this feature is also novel;

The rest of this paper is organized as follows: In Sect. 2, we review well-known methods for the detection of clickbait. We explain our approach for detecting clickbait in social media in Sect. 3. In Sect. 4, we present the data used for this study. Section 5 describes the experiment setup, and Sect. 6 presents the results obtained, and discusses them. Finally, we conclude the paper in Sect. 7.

2 Related Work

This section presents studies focused on clickbait and abuser detection on social media.

2.1 Clickbait Detection

In 2014, Vijgen [19] studied listicles, a portmanteau of ‘list’ and ‘article.’ In many cases, these listicles are suspected to be clickbait due to their titles, typically shared as teaser messages. Vijgen collected 720 listicles published by BuzzFeed[1] in January 2014. He found that all titles contain a cardinal number, which is the same as the number of items listed. In addition, the titles contained strong nouns and adjectives that convey authority and sensationalism. Gianotto [8] implemented a browser plugin that detects clickbait based on a rule set in the same year.

In 2015, Blom and Hansen [2] mapped the use of headlines pointing to full articles in online news by analyzing 100,000 headlines published on ten different Danish news websites. They found that commercialization and tabloidization seem to lead to the recurrent use of forward-referencing in Danish online news headlines.

Also, in 2015, Chen et al. [5] examined optional methods for the automatic detection of clickbait. They divided the methods into methods that rely on the content and non-text cues. The former includes lexical and semantic analysis and syntactic analysis, whereas the latter focuses on image and user behavior analysis. Chen et al. suggested that a hybrid approach that merges both methods may yield better results.

In 2016, Potthast et al. [16] proposed a machine learning classifier for automatic clickbait detection. They collected and annotated a corpus of 2,992 tweets and later trained an ML classifier, differentiating between legitimate and clickbait. Their classifier included 215 features, including image, sentiment, and linguistic analysis, as well as extracting Twitter-specific features and bag-of-words features. Their best classifier was Random Forest, which obtained an AUC score of 0.79, a precision score of 0.76, and a recall score of 0.76. Potthast et al. also analyzed the web pages linked from a given tweet. Their analysis included measurement of the main content word length.

In the same year, Chakraborty et al. [3] trained an ML classifier for detecting clickbait automatically and implemented a browser extension called ‘Stop Clickbait’ to prevent readers from reading clickbait. For training the classifier, they extracted the headlines from a corpus of 18,513 Wikinews articles as legitimate

[1] https://www.buzzfeed.com/.

posts. As clickbait, they crawled 8,069 Web articles from several Web domains, such as BuzzFeed, ViralNova[2], ScoopWhoop[3], and ViralStories[4]. They used a set of 14 features spanning linguistic analysis, word patterns, and N-gram for training their classifier. Their best classifier was Support Vector Machine (SVM), achieved 0.93 accuracy score in detecting and 0.89 accuracy score in blocking clickbaits.

In 2017, Chakraborty et al. [4] analyzed the social sharing patterns of clickbait and legitimate posts on Twitter by collecting a dataset from Twitter. They found that clickbait tweets include more entities, such as images, hashtags, and user mentions than non-clickbait tweets, assisting in capturing the consumers' attention. Also, they witnessed a higher percentage of clickbait tweets conveying positive sentiments than non-clickbait tweets. Other conclusions include that clickbait tweets are consumed more by women than men and more younger people than the consumers of non-clickbait. Additionally, these clickbaits have higher mutual engagement among each other. On the other hand, non-clickbait consumers are more reputed in the community and have a relatively higher follower base than clickbait consumers.

In recent years, with the development of deep learning many deep learning models were suggested. Traditional machine learning-based-methods suffered from heavy feature engineering. AS a result in the recent years, many researchers used deep learning model for clickbait detection [10,17,18,20].

3 Proposed Method

We propose an approach to differentiating between clickbait and legitimate posts. It is based on training a mainstream machine learning classifier to extract image-to-text, linguistic, and behavioral features.

3.1 Content-Based Features

For describing the features extracted follow the next definitions: Let P denotes the collection of tweets published; p is defined as a post in post collection P. For every post $p \in P$, $article(p)$ denotes the article, which post p points to. c is defined as the content, any text existing in p or $article(p)$. Additional functions are described below:

1. $img(p)$ returns a image attached in the given p.
2. $OCR(img(p))$ extracts the text exists in a given image p.
3. $title(p)$ and $title(article(p))$ returns the title of p and the article's title, respectively.
4. $description(article(p))$ extracts the article's description.
5. $keywords(article(p))$ returns the keywords of a given article.

[2] https://viralnova.com/.
[3] https://www.scoopwhoop.com/.
[4] https://viralstories.co.uk/.

6. $paragraphs(article(p))$ returns the text exist in the paragraphs of a given article.
7. $captions(article(p))$ extracts the caption of an image associated with a given post p.
8. $len_{characters}(c)$ returns the number of characters in a given content.
9. $words(c)$ returns a set of words, containing in a given content c.
10. $len_{words}(c)$ returns the number of words in a given content.
11. $lang - dict_{formal}(words(c))$ returns a set of the formal English words from the given content c.

3.2 Image-Based Features

In many cases, when facing a post, people usually watch the image before reading the headline [6]. As a result, we suggest extracting features that emphasize the gap between the image and the headline of the given post p. For this, we extract the text that exists in the post's image by the pytesseract[5] (Python-tesseract) package, an optical character recognition (OCR) tool available in Python aimed at recognizing the text embedded in images.

Presence of an Image in a Post. Clickbait tweets contained a significantly more significant proportion of images than legitimate posts [4]. Therefore, we suggest monitoring the existence of an image in a given post.

Presence of Text in a Post's Image. Here, we extract the text in the post's image using OCR. In this case, this is boolean feature indicates whether a text exists in a given post's image.

3.3 Linguistic-Based Features

Linguistic analysis is another well-known method for detecting clickbaits. This analysis includes semantic and syntactic analysis to find nuances that occur more frequently in clickbait posts than legitimate posts [3].

The features are the number of characters in the post's title, the number of characters in text extracted from the post's image, the number of characters in the article's title, the number of characters in the article's description, the number of characters in article's keywords, the number of characters in article's captions, and the number of characters in article's paragraphs.

Difference Between Number of Characters. This function measures the difference between characters in two content elements. As opposed to previous studies, which measured the length of the suspected post's title, here, we examine the relationship between each pair of content elements (see Eq. 1).

$$diff - num - of - characters(cont_x, cont_y) = |len_{characters}(cont_x) - len_{characters}(cont_y)| \tag{1}$$

[5] https://pypi.org/project/pytesseract/.

Using this function, we extracted 21 features. For example, the diff number of characters between post's title and article's title.

Number of Characters Ratio. We extract the ratio between the number of characters of two content elements (see Eq. 2).

$$num - of - characters - ratio(cont_x, cont_y) = |\frac{len_{characters}(cont_x)}{len_{characters}(cont_y)}| \tag{2}$$

The same as previous function, 21 features were extracted (e.g., number of characters ratio between post's title and article's title).

Number of Words. Chakraborty et al. [3] found that there are more words in clickbait titles than non-clickbait titles. Therefore, we used $words(c)$, which returns a set of words, containing in a given content c. Based on this function, we counted the words for each element in the post and the article.

Difference Between Number of Words. We also measure the word difference between two contents (see Eq. 3).

$$diff - num - of - words(cont_x, cont_y) = |num - of - words(cont_x) - num - of - words(cont_y)| \tag{3}$$

Number of Words Ratio. Another function which can be extracted is the ratio between the number of words in two content elements (see Eq. 3).

$$num - of - words - ratio(cont_x, cont_y) = |\frac{num - of - words(cont_x)}{num - of - words(cont_y)}| \tag{4}$$

Mutual Words Between Article Keywords and Post's Words. In many cases, clickbait posts contain misleading titles, exaggerating the content of the targeted article [1]. These exaggerating words exist in the posts and are necessarily not expressed in the article. The idea is to monitor the shared words between the article's keywords, which convey the main topic expressed in the article to the post's title (see Eq. 5).

$$num - of - common - article - words(cont_x, cont_y) = len(words(cont_x) \cap words(cont_y)) \tag{5}$$

Based on this function, we extracted features focusing on the common words from every element in the article, such as keywords, captions, paragraphs, etc.

Number of Formal and Informal English Words. In the advertisement environment, slang or profane words are commonly used to get users' attention [21]. As a result, we count the number of formal English words in each content item. We use the PyDictionary,[6] a dictionary module for Python, which provides meanings,

[6] https://pypi.org/project/PyDictionary/.

translations, synonyms, and antonyms of words. It uses WordNet[7] for definitions, Google[8] for translations, and thesaurus.com[9] for synonyms and antonyms. The following function counts the number of formal English words using the extraction of the words in each content element and searching each word in the English dictionary (see Eq. 6). Using this function, we also count the number of informal English words.

$$number - of - formal - words(c) = lang - dict_{formal}(words(c)) \tag{6}$$

Percent of Formal and Informal Words. We also measure how many of the words presented in the posts are informal. It also can lead us to understanding whether it is a clickbait or legitimate post (see Eq. 7).

$$percent - of - formal - words(c) = \frac{lang - dict_{formal}(words(c)}{words(c)} \tag{7}$$

3.4 Behavioral-Based Features

Clickbait is responsible for the rapid spread of rumors and misinformation online [5]. This malicious activity is also common among abusers in social media [7]. Therefore, we can extract features that are helpful for abuser detection also for the clickbait detection. For example, the average number of links per tweet, the average number of user mentions per tweet, etc. [12] In this study, we extracted the following features:

1. Number of mentioning users - User mention is used to call out user names in posts on Twitter.
2. Number of Hashtags - a hashtag is any word or phrase with a prefix consisting of the # symbol. This mechanism connects different posts to a specific topic.
3. Number of Retweets - a retweet is defined as a tweet that a user forwards to their followers. Retweets are often used to pass along news or other valuable information on Twitter
4. Number of Question Marks, Commas, Colons, and Ellipses one of the methods used to attract readers' attention to a specific article is the use of question marks, commas, colons, and ellipses in titles.

We extracted these features from all the content fields available in the given posts and articles.

Additional features that were reported to perform well in the past are those features that are related to the account properties (e.g., number of friends, number of followers) [12]. Thus, the related features were extracted, such as the number of article keywords, the number of paragraphs, and the number of article captions.

[7] https://wordnet.princeton.edu/.
[8] https://www.google.com/.
[9] https://www.thesaurus.com/.

4 Data Description

To evaluate the proposed approach, we used two datasets published as part of Clickbait Challenge 2017 [14]. The first dataset was perceived as a small initial training set [16]. It includes 2,495 posts consisting of 762 clickbait and 1,697 legitimate posts. The second dataset was more significant than the first dataset consisting of training and validation datasets [15]. It includes 19,538 posts, consisting of 4,761 clickbait, and 14,777 legitimate posts. Each post can include an attached image and point to a targeted article. The targeted article consists of a title, description, paragraphs, and captions attached to the photos. Figure 1 presents the structure of the posts and articles in the datasets.

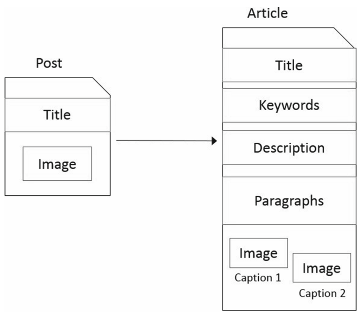


Fig. 1. Name suggestion steps.

5 Experiment Setup

In this study, we aimed at answering 2 research questions using the experiments described below: 1) *Are these novel features proposed contribute to clickbait detection?* 2) *Are there other differences between the article and post pointing to that can help in the detection of clickbaits?*

The evaluation process was carried out in TIRA (Testbed for Information Retrieval Algorithms) framework [9] as provided by the challenge organizers. In order to evaluate the predictive power of the extracted features, we measured the feature importance using information gain.

Next, we trained several mainstream machine learning classifiers based on the features extracted for differentiating between clickbait and legitimate posts. The evaluated machine learning classifiers were decision tree, random forest, and XGBoost. Each classifier was trained with the best 20 features and all features. The performance of each classifier was measured using AUC (area under ROC curve), accuracy, precision, and recall scores.

6 Results and Discussion

6.1 Feature Analysis

The most significant features according to their information gain score are presented in Table 1.

Table 1. Top features according to information gain score

Dataset	Rank	Feature	Info Gain
Training	1	Diff num of characters post title & article keywords	0.036
	2	Num of characters ratio post image text & post title	0.032
	3	Num of characters in post title	0.03
	4	Num of question marks in post title	0.021
	5	Diff num of characters post title & post image text	0.02
	6	Num of characters ratio article description & post title	0.019
	7	Num of characters ratio article paragraphs & post title	0.018
	8	Diff num of words post title & article keywords	0.018
	9	Num of words ratio article description & post title	0.017
	10	Num of characters ratio article paragraphs & article desc	0.017

Based on the top features obtained, we can notice that features that measure a gap between a post and its associated article are found as the most important. For example, the difference between the number of characters in a post title and article's keywords, obtained an information gain score of 0.036. Also, the length of the post title (number of characters in the post's title) and the ratio of number of characters between the image's text and the post's title were found as important features.

6.2 Performance

Based on these results, we can see that the best classifier in both datasets was XGBoost, which trained on all of the features with an information gain score higher than zero. This classifier obtained on the training set an AUC score of 0.715, accuracy of 0.732, precision and recall of 0.75 and 0.92, respectively. On the validation set, the classifier obtained an AUC of 0.8, an accuracy of 0.812, and precision and recall scores of 0.819 and 0.966, respectively (see Table 2).

Table 2. Trained classifiers' performance

Dataset	Algorithm	#Features	AUC	Accuracy	Precision	Recall
Training	XGBoost	All	**0.715**	**0.732**	**0.75**	**0.92**
	XGBoost	20	0.707	0.728	0.744	0.925
	Random Forest	All	0.707	0.732	0.752	0.913
	AdaBoost	20	0.702	0.721	0.74	0.917
	Random Forest	20	0.698	0.725	0.753	0.895
	Decision Tree	All	0.583	0.636	0.743	0.721
Validation	XGBoost	All	**0.8**	**0.812**	**0.819**	**0.966**
	Random Forest	All	0.789	0.811	0.823	0.955
	AdaBoost	All	0.777	0.802	0.818	0.951
	XGBoost	20	0.776	0.807	0.814	0.966
	Random Forest	20	0.771	0.804	0.823	0.944
	Decision Tree	All	0.635	0.725	0.824	0.811

7 Conclusions

This paper addressed the clickbait detection problem by training a mainstream ML classifier based on novel linguistic, image-based, and behavioral features. Using these novel features, the classifier differentiated between clickbait and legitimate posts. Based on the evaluation, we conclude the following: First, the linguistic features were found the most significant and useful for clickbait detection. The different number of characters between post's title and article keywords, the number of characters ratio between text extracted from post's image and post's title, and the number of characters in post's title were obtained the highest information gain score (see Table 1). We conclude that linguistic differences between the post and article can assist in clickbait detection.

Finally, we found that the post's title is the most important component for detecting clickbait. It sounds trivial, but we succeeded at showing this empirically. We created the same features for each competent equally. Of the twenty-four most influential features (based on the information gain score), only one feature is not related to the post's title (see Table 1).

References

1. Biyani, P., Tsioutsiouliklis, K., Blackmer, J.: "8 amazing secrets for getting more clicks": detecting clickbaits in news streams using article informality. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
2. Blom, J.N., Hansen, K.R.: Click bait: forward-reference as lure in online news headlines. J. Pragmat. **76**, 87–100 (2015)
3. Chakraborty, A., Paranjape, B., Kakarla, S., Ganguly, N.: Stop clickbait: detecting and preventing clickbaits in online news media. In: 2016 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM), pp. 9–16. IEEE (2016)

4. Chakraborty, A., Sarkar, R., Mrigen, A., Ganguly, N.: Tabloids in the era of social media? understanding the production and consumption of clickbaits in twitter. In: Proceedings of the ACM on Human-Computer Interaction 1(CSCW), pp. 1–21 (2017)
5. Chen, Y., Conroy, N.J., Rubin, V.L.: Misleading online content: recognizing clickbait as "false news". In: Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection, pp. 15–19 (2015)
6. Ecker, U.K., Lewandowsky, S., Chang, E.P., Pillai, R.: The effects of subtle misinformation in news headlines. J. Experiment. Psychol. Appl. **20**(4), 323 (2014)
7. Elyashar, A., Bendahan, J., Puzis, R., Sanmateu, M.A.: Measurement of online discussion authenticity within online social media. In: 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 627–629. IEEE (2017)
8. Gianotto, A.: Downworthy: a browser plugin to turn hyperbolic viral headlines into what they really mean. downworthy. snipe. net (2014)
9. Gollub, T., Stein, B., Burrows, S., Hoppe, D.: Tira: configuring, executing, and disseminating information retrieval experiments. In: 2012 23rd International Workshop on Database and Expert Systems Applications, pp. 151–155. IEEE (2012)
10. Gothankar, R., Di Troia, F., Stamp, M.: Clickbait detection in Youtube videos. arXiv preprint arXiv:2107.12791 (2021)
11. Lavie, T., Sela, M., Oppenheim, I., Inbar, O., Meyer, J.: User attitudes towards news content personalization. Int. J. Hum Comput Stud. **68**(8), 483–495 (2010)
12. Lee, K., Tamilarasan, P., Caverlee, J.: Crowdturfers, campaigns, and social media: tracking and revealing crowdsourced manipulation of social media. In: Proceedings of the International AAAI Conference on Web and Social Media, vol. 7 (2013)
13. Loewenstein, G.: The psychology of curiosity: a review and reinterpretation. Psychol. Bull. **116**(1), 75 (1994)
14. Potthast, M., Gollub, T., Hagen, M., Stein, B.: The clickbait challenge 2017: towards a regression model for clickbait strength. arXiv preprint arXiv:1812.10847 (2018)
15. Potthast, M., et al.: Crowdsourcing a large corpus of clickbait on Twitter. In: Proceedings of the 27th International Conference on Computational Linguistics, pp. 1498–1507 (2018)
16. Potthast, M., Köpsel, S., Stein, B., Hagen, M.: Clickbait detection. In: Ferro, N., et al. (eds.) ECIR 2016. LNCS, vol. 9626, pp. 810–817. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30671-1_72
17. Razaque, A., et al.: Blockchain-enabled deep recurrent neural network model for clickbait detection. IEEE Access **10**, 3144–3163 (2021)
18. Razaque, A., Alotaibi, B., Alotaibi, M., Hussain, S., Alotaibi, A., Jotsov, V.: Clickbait detection using deep recurrent neural network. Appl. Sci. **12**(1), 504 (2022)
19. Vijgen, B., et al.: The listicle: an exploring research on an interesting shareable new media phenomenon. Studia Universitatis Babes-Bolyai-Ephemerides **59**(1), 103–122 (2014)
20. Zheng, J., Yu, K., Wu, X.: A deep model based on lure and similarity for adaptive clickbait detection. Knowl.-Based Syst. **214**, 106714 (2021)
21. Zhou, K., Redi, M., Lalmas, M., Sangal, P.M.: Filtering machine for sponsored content, US Patent 11,113,714, 7 September 2021

Etherless Ethereum Tokens: Simulating Native Tokens in Ethereum

John Andrews[1], Michele Ciampi[2](✉), and Vassilis Zikas[3]

[1] Sunday Group, Las Vegas, USA
jandrews@sundaygroupinc.com
[2] The University of Edinburgh, Edinburgh, UK
michele.ciampi@ed.ac.uk
[3] Purdue University, West Lafayette, USA
vzikas@cs.purdue.edu

Abstract. Standardized Ethereum tokens, e.g., ERC-20 tokens, have become the norm in fundraising (through ICOs) and kicking off blockchain-based DeFi applications. However, they require the user's wallet to hold both tokens and ether to pay the gas fee for making a transaction. This makes for a cumbersome user experience, and complicates, from the user perspective, the process of transitioning to a different smart-contract enabled blockchain, or to a newly launched blockchain. We formalize, instantiate, and analyze in a composable manner a system that we call *Etherless Ethereum Tokens* (in short, EETs), which allows the token users to transact in a closed-economy manner, i.e., having only tokens on their wallet and paying any transaction fees in tokens rather than Ether/Gas. In the process, we devise a methodology for capturing Ethereum token-contracts in the Universal Composability (UC) framework, which can be of independent interest. Our system can be seen as a targeted instance of the more general paradigm put forth—without a formal study—by the Ethereum Gas Station Network (GSN). We have implemented and benchmarked our system and compared it to GSN. In addition to being the first system with a rigorous security analysis, we demonstrate that EETs are far easier to deploy and less gas intensive than the GSN.

1 Introduction

As applications of smart contracts, e.g., Decentralized Finance (DeFi) and Non-Fungible Tokens (NFTs), become mainstream, there is a need to make them as independent from the Ethereum chain as possible. This is particularly relevant for Ethereum tokens (e.g., ERC-20 tokens [20]). Indeed, for a token-holder to exchange or transfer such tokens, they need to also hold Ether for fuelling the Ethereum transaction. This is counter-intuitive and counter-productive: on the one hand, token creators need to provide a wallet which supports both their token and Ethereum, making it more challenging to transition to their own blockchain or switch token platforms while offering a smooth user experience. On the other hand, users need to make sure that they hold not only the token but also Ether,

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 388–407, 2022.
https://doi.org/10.1007/978-3-031-07689-3_29

which makes it more challenging to expand this technology to less tech-savvy audiences, thereby hindering wider societal adoption.

The easiest way to conceptualize the relevant bottleneck is through considering the life cycle of an ETH-based initial coin offering (ICO): in a first stage, the token creator solicits investment (typically in different cryptocurrencies), under the promise of a certain (prearranged) amount of tokens once the token launches.[1] In a second phase, the token creator initializes the promised new token by launching a token smart contract (e.g. an ERC 20 token) on the Ethereum chain. The token creator then would then have the investors create and provide an Ethereum address where the promised tokens can be transferred. This can be done by means of a wallet that offers generic support for Ethereum tokens.

Often, however, ICO-funded applications launch tokens which have the ultimate goal of eventually being disconnected from the main Ethereum blockchain, and/or which aim to create an ecosystem independent of Ethereum. In such cases, the token creator would typically also offer its users a token-specific wallet application. However, in order for anyone to use this application to transfer his tokens, the token-specific wallet needs to also support Ether as a currency. This leads to confusion for less tech-savvy investors, and makes the user experience of migrating the token to a different smart contract platform—e.g. a different smart-contract-enabled blockchain or a blockchain developed by the token creator—less intuitive. We note that such migration is becoming more relevant as more smart-contract-enabled blockchains are released, and as the gas price for Ethereum smart contracts rises to a point where its use makes the corresponding tokens less attractive.

In this work, we start by proposing a design methodology and formal treatment of Ethereum tokens which allow their creator to provide the option to its users of making transfers without the need to hold Ether in their wallet, a mechanism which we term *Etherless Ethereum Tokens (in short, EETs)*. The high-level idea is simple: allow the token creator to take on the cost (i.e., gas) for the token transaction, and have the token contract perform an on-the-fly exchange of token-to-ether at a pre-agreed rate, giving the user the experience of a native token. As one might expect, properly specifying, implementing, and proving such a protocol secure is a challenging task; in particular, it requires a model for token-enabled ledgers, which we provide and believe it will be helpful for all the future systems that might rely on token contracts in a composable manner. We remark that, as a concept, etherless transactions have been frequently discussed within the Ethereum community for several years, often under the term *meta transactions* [1,2,12,18]. However, to our knowledge, our work is the first to provide a formal treatment and security analysis of the concept.

The need for arguing the security of blockchain systems formally and in a composable manner is motivated by the fact that a blockchain does not live

[1] There are a number of legal issues regarding ICO's—in particular, how to hold the token creator to his promise and how to avoid scamming attacks—and there are technological advances that allow us to circumvent them; these topics are outside the scope of this paper.

in isolation, and that many applications might run on top of it. Hence, it is fundamental to argue the security of new blockchain applications in a setting where multiple protocols are running in concurrency. Indeed, many recent works have focused specifically on this task, proposing formal security models and proving that existing protocols (like Bitcoin [5,10]) satisfy some important and well-formalized security properties. On the same spirit, many other works have used the same rigorous approach to argue and define the security of other blockchain systems and applications. Just a few other examples are proof-of-stake blockchains, private blockchains, private smart contracts and the lighting network. [4,13–15].

At a less technical level, we believe that in addition to offering a more intuitive, closed-economy user experience, EET also provides assurance to the original ICO investors that the token creator indeed expects value on the token, as he is willing to make marginal exchanges. Indeed, in the system we design anyone (in particular the token creator) could pay the fee (in Ether) on the behalf on another party. Throughout we will generically refer to such an entity as *intermediary*. We note in passing that despite being explicitly implemented on the Ethereum blockchain, our design is generic and can be ported to any smart-contract-enabled blockchain platform, and thus can enable transferring the tokens from one blockchain to another.

We have implemented our EET design, and we demonstrate how it outperforms existing generic systems that enable etherless transactions, such as the Gas Station Network (GSN) [1], both in terms of simplicity of deployment and in terms of gas usage. We also compare such a deployment with how a native token could perform on Ethereum and demonstrate that the overhead makes the flexibility offered by black-box usage of smart-contract-based tokens a reasonable compromise for the moderate increase in the required gas it incurs over what a native token would require.

2 Our Contributions and Related Work

Our contribution is threefold: (1) A universally composable (UC) [6] treatment of ledgers supporting a broad class of smart contracts, which includes token contracts (e.g. ERC 20). (2) A design and UC security analysis of EETs. (3) An implementation of our EET, benchmarks, and comparison with alternative approaches. In the following, we expand on the key components of the above contributions, and put our results in perspective with existing literature and systems.

2.1 Smart-Contract-Enabled Transaction Ledgers

The first analyses of blockchain protocols showed that they satisfy certain desirable properties, such as common-prefix (also referred to as safety or consistency), chain-growth (also referred to as liveness), chain quality, etc. [4,5,9–11,16,17]. Badertscher *et al.* [5] put forth the first universally composable treatment of the

Bitcoin backbone (i.e. consensus layer) by introducing a UC functionality, called $\mathcal{F}_{\text{LEDGER}}$, which captures the interface that Bitcoin offers to external applications, rather than the way in which this interface is implemented. At a very high level, $\mathcal{F}_{\text{LEDGER}}$ takes as input transactions which are validated by means of a validation predicate $\mathsf{Validate}$. All valid transactions are then stored into a data structure denoted as *state*. The adversary has full control over the order in which transactions appear in `state`, and can define (in a limited way) the portion of the state that each party can access. However, once something is added to the state, it cannot be removed (not even by the adversary). We note that the advantage of proving security in UC is that it enables use of the ledger as an ideal primitive, and ensures that replacing this ideal ledger primitive by its implementation—the corresponding blockchain—does not compromise the security of primitives that make ideal calls to the ledger; nor does it affect the security of systems and protocols that run alongside the ledger. This property is often referred to as universal composability, and it allows for a constructive approach to cryptographic/security protocols, analogous to how programming uses libraries with fixed APIs without worrying about their implementation. Following that work, a number of papers on the design and analysis of blockchains have adopted UC as the model to prove their security and have devised systems implementing variants of the above ledger [4,14]. UC [5] has also been leveraged to describe how $\mathcal{F}_{\text{LEDGER}}$ may be used together with a digital signature scheme to derive a *transaction ledger*, abstracting the cryptocurrency aspects of Bitcoin in addition to its backbone guarantees.[2]

This was done by relying on digital signatures where, to ensure composability, the ideal adversary is allowed to choose the signing and verification keys.

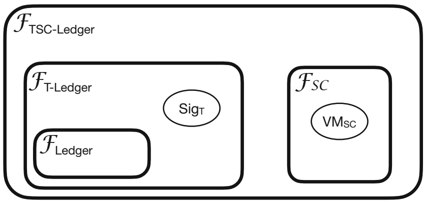
The Transaction Ledger. In this paper we consider a simpler, more UC-friendly approach that abstracts away the public-key infrastructure (PKI), analogous to how the UC signatures functionality [7] would. In a nutshell, instead of having $\mathsf{Validate}$ rely on a specific signature scheme, we define a new transaction ledger $\mathcal{F}_{\text{T-LEDGER}}$ that internally runs $\mathcal{F}_{\text{LEDGER}}$ and also emulates existentially unforgeable signatures, similar to [7]. $\mathcal{F}_{\text{T-LEDGER}}$ accepts transactions with the format $\texttt{tx} := (v, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{fee})$ where v represents the number of coins involved in the transaction, fee is the fee that the issuer of the transaction is willing to pay, and addr_i and addr_j represent the wallet addresses of the sender and the receiver respectively. Upon receiving a transaction, $\mathcal{F}_{\text{T-LEDGER}}$ checks the state of $\mathcal{F}_{\text{LEDGER}}$ to ensure that the wallet address addr_i has at least $v + \mathsf{fee}$ coins and that the fee is sufficient, i.e. that $\mathsf{fee} \geq f(\texttt{tx})$, where f is function specified in the description of $\mathcal{F}_{\text{T-LEDGER}}$ that determines the fee that needs to be payed for the input transaction. We note that it is straightforward to adapt the analysis of the transaction ledger [5]—using a specific existentially-unforgeable signatures scheme—to prove security of our ledger for a standard Bitcoin-style blockchain protocol, such as

[2] Unlike transaction ledgers, the bare $\mathcal{F}_{\text{LEDGER}}$ captures the consensus layer, and does not interpret its contents as transactions which need to be verified with respect to whether or not they are spending some already spent coin.

Bitcoin or the proof-of-work-based version of Ethereum. Nonetheless, as we shall see, this makes it more intuitive to add cryptocurrency-relevant features to the ledger–such as etherless tokens.

Adding Smart Contracts. The functionality $\mathcal{F}_{\text{T-LEDGER}}$ is sufficient to capture the base functionality of cryptocurrencies, but it does not support smart contracts. To achieve that, in this work we define an augmented functionality, which we denote $\mathcal{F}_{\text{TSC-LEDGER}}$. This represents our first contribution. $\mathcal{F}_{\text{TSC-LEDGER}}$ internally manages $\mathcal{F}_{\text{T-LEDGER}}$ and a functionality $\mathcal{F}_{\text{SC}}$ that abstracts a smart contract: $\mathcal{F}_{\text{SC}}$ maintains its own state `cstate`—corresponding to the state of a (virtual) machine VM[3]—and is parametrized by a function f_{CFee}, that takes as input the query to the contract (which contains also the fee that the caller is willing to pay to run the contract), and checks whether or not the fee is enough for the VM to process the input and update its state.

The construction of $\mathcal{F}_{\text{TSC-LEDGER}}$ from its components is illustrated in Fig. 1. $\mathcal{F}_{\text{TSC-LEDGER}}$ accepts either standard transactions in the native currency `E` (that are forwarded to $\mathcal{F}_{\text{T-LEDGER}}$) or inputs/transactions that are intended as queries to the contract $\mathcal{F}_{\text{SC}}$. Upon receiving such a query for the smart contract, $\mathcal{F}_{\text{TSC-LEDGER}}$ forwards the query to $\mathcal{F}_{\text{SC}}$, which checks if the fee specified in the query is sufficient to update its state, and if so it updates `cstate` by running the VM on input the given transaction and the state of $\mathcal{F}_{\text{T-LEDGER}}$ (which is handed to $\mathcal{F}_{\text{SC}}$ by $\mathcal{F}_{\text{TSC-LEDGER}}$)[4], and returns the updated state (including the received input) to $\mathcal{F}_{\text{TSC-LEDGER}}$. $\mathcal{F}_{\text{TSC-LEDGER}}$ then pushes the query and the updated state `cstate` to the state of $\mathcal{F}_{\text{T-LEDGER}}$ (by submitting it as a transaction). Consistently with the Ethereum smart contract mechanism, $\mathcal{F}_{\text{SC}}$ charges the contract caller only for the fee that is required to update its state, even if the contract's caller specified a higher fee. Moreover, if a contract caller did not specify a fee high enough to conclude an update on the contract's state, the fee will be deducted from the caller account, and the input used to query the contract will appear in the state of $\mathcal{F}_{\text{T-LEDGER}}$, though no change to the contract's state will be committed.



$\mathcal{F}_{\text{TSC-Ledger}}$
$\mathcal{F}_{\text{T-Ledger}}$
Sig_T
$\mathcal{F}_{\text{Ledger}}$
$\mathcal{F}_{SC}$
VM_{SC}

Fig. 1. The smart-contract-enabled transaction ledger functionality $\mathcal{F}_{\text{TSC-LEDGER}}$

Tokens as Smart Contracts. Given the above smart-contract-enabled ledger, it is straightforward to capture a smart contract for creating a standard (e.g. ERC 20 [20]) Ethereum token by instantiating $\mathcal{F}_{\text{TSC-LEDGER}}$ with contract functionality that stores and updates the state (balances for different addresses)

[3] We do not specify a model of computation for describing the VM; one can use any such model, e.g. Turing machines, RAMs, etc.

[4] Note that $\mathcal{F}_{\text{TSC-LEDGER}}$ also keeps track of the history of the state of $\mathcal{F}_{\text{T-LEDGER}}$.

of such a token. Note that this results in a token-enabled transaction ledger $\mathcal{F}^{\mathsf{Token}}_{\text{LEDGER}}$ which allows parties both to issue transactions in the native coin $\mathtt{E}$, and to exchange tokens $\mathtt{T}$.

In more detail, $\mathcal{F}^{\mathsf{Token}}_{\text{LEDGER}}$ instantiates $\mathcal{F}_{\text{TSC-LEDGER}}$ with a token-contract $\mathcal{F}^{\mathsf{T}}_{\text{SC}}$ which works as follows: $\mathcal{F}^{\mathsf{T}}_{\text{SC}}$ collects all token transactions, and upon receiving a read-request returns only the *valid* token transactions. Similarly to the way the ledger $\mathcal{F}_{\text{T-LEDGER}}$ deals with native transactions, a token transaction consists of the components $(v, \mathsf{addr}^{\mathsf{T}}_i, \mathsf{addr}^{\mathsf{T}}_j)$, where v is the number of tokens involved in the transaction, and addr_i and addr_j represent the token wallet addresses of the sender and the receiver respectively. Furthermore, $\mathcal{F}^{\mathsf{T}}_{\text{SC}}$ internally emulates an existentially-unforgeable signature scheme related to the token which is independent of the one that is used in $\mathcal{F}_{\text{T-LEDGER}}$.[5]

We observe that there is no fee appearing in the description of the token transaction. The reason is that the fee will be part of the query to the contract, and it is expressed in the native currency $\mathtt{E}$. Indeed, the issuer of the token transaction, in order to query the contract $\mathcal{F}^{\mathsf{T}}_{\text{SC}}$, needs to possess coins of type $\mathtt{E}$.

The EET Functionality. As discussed in the introduction, the above contract implementation of tokens—which has become a standard for Ethereum—has the undesireable property that a party who wants to send tokens requires coins of type $\mathtt{E}$ to do so, coins which they might not have. In this work, we introduce EETs to allow the token creator to offer, as a service, to take on the cost of the token transaction, in exchange for tokens at a pre-agreed $\mathtt{E}$-to-$\mathtt{T}$ rate. This is captured by tweaking the token-enabled ledger $\mathcal{F}^{\mathsf{Token}}_{\text{LEDGER}}$ toward an EET-enabled ledger, denoted as $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$, which supports an additional input called SUBMIT-DELEGATION. Upon receiveing SUBMIT-DELEGATION, $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ allows the user to issue a token transaction which pays a fee, in $\mathtt{T}$, to a special party, called *intermediary* (that we denote with M), in exchange for the intermediary submitting the token transaction to $\mathcal{F}^{\mathsf{T}}_{\text{SC}}$ and paying the $\mathtt{E}$ needed for the token contract to process the transaction. In our system anyone can be an intermediary. More precisely, there might be multiple intermediaries that are willing to pay the Ether fee for a transaction, but each of them will do that at a potentially different exchange rate. This means that any user that wants to delegate the payment of the fee can look at what rates are available and decide accordingly with what intermediary to interact with. The agreement on the rate is made completely off-chain, and for sake of simplicity in the paper we assume that there is only one intermediary and that the rate has been already pre-agreed between the parties.

2.2 EET Construction and Analysis

To realize $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ we rely only on $\mathcal{F}_{\text{T-LEDGER}}$ and signatures. In particular, any party that wants to issue a token transaction and has enough coins of type $\mathtt{E}$ to

[5] Note that we cannot generically use the same signature emulator procedure of $\mathcal{F}_{\text{T-LEDGER}}$, as a token address is typically overloaded to also be an Ethereum address.

cover for the fee can issue a transaction $\texttt{tx} = (0, \mathsf{addr}_i, 0^\lambda, (\mathsf{aux}, \sigma), \mathsf{fee})$, where $\mathsf{aux} = (v, \mathsf{addr}_i^\mathtt{T}, \mathsf{addr}_j^\mathtt{T})$ and σ is a signature of aux that verifies under $\mathsf{addr}_i^\mathtt{T}$.[6]

In a nutshell, $\texttt{tx}$ is a standard transaction for $\mathcal{F}_{\text{T-LEDGER}}$ that contains in its payload the information related to the token transaction properly signed by the sender. By definition, if the fee fee is high enough, then $\texttt{tx}$ will become part of $\mathcal{F}_{\text{T-LEDGER}}$'s state. Let $\mathsf{addr}_\mathsf{M}^\mathtt{T}$ be the token wallet address of M. To delegate a transaction, the sender P_i creates a special token transaction $\mathsf{aux} = ([v, \mathsf{del\text{-}fee}], \mathsf{addr}_i^\mathtt{T}, [\mathsf{addr}_j^\mathtt{T}, \mathsf{addr}_\mathsf{M}^\mathtt{T}])$ (where $\mathsf{del\text{-}fee}$ is a fee expressed in $\mathtt{T}$ that parametrizes $\mathcal{F}_{\text{LEDGER}}^{\mathsf{EET}}$) and signs it to obtain σ. aux is the atomic representation of two token transactions: the first moves v tokens from $\mathsf{addr}_i^\mathtt{T}$ to $\mathsf{addr}_j^\mathtt{T}$, and the second moves $\mathsf{del\text{-}fee}$ from $\mathsf{addr}_i^\mathtt{T}$ to $\mathsf{addr}_\mathsf{M}^\mathtt{T}$. M, upon receiving (aux, σ) submits a transaction to $\mathcal{F}_{\text{T-LEDGER}}$ that contains (aux, σ) in its payload. If a party wants to obtain only the valid token transaction, they need to filter out the payload of the transactions stored in $\mathcal{F}_{\text{T-LEDGER}}$'s state, and output only the valid transactions. Similarly to what we have described above, a token transaction $(v, \mathsf{addr}_i^\mathtt{T}, \mathsf{addr}_j^\mathtt{T})$ is valid if the sum of tokens with receiver address $\mathsf{addr}_i^\mathtt{T}$ minus the sum of tokens in the state with sender address addr_i (including the fees) is greater than or equal to v.

2.3 Implementation, Benchmarks, and Comparisons

The Gas Station Network (GSN) is a relatively recent development in the Ethereum community that shares some of our goals, but a broader scope. In particular, the GSN aims to create a decentralized, trustless network of *relay servers* which can pick up the transaction fees for any GSN-enabled contract. The GSN is built around a RelayHub smart contract that:

1. Records available relay servers and their service fees,
2. Keeps ether deposits from GSN-enabled contracts for repayment of relay servers,
3. Facilitates the interaction between relays and GSN-enabled contracts, and punishes any detected bad actors.

This is in contrast to our mechanism, in which there is no separate smart contract to manage the delegation of transactions. Additionally, each GSN-enabled contract must interact with a separate *paymaster* contract, which is responsible for performing any action needed to extract or verify payment from users. Paymaster contracts may be written generically and shared between multiple contracts, or purpose-written for particular contracts.

The outward functionality of the GSN is similar to our mechanism: a gasless user submits a transaction to an intermediary relay server instead of directly to the blockchain, and the relay submits the transaction on the user's behalf, receiving an ether repayment from the target contract. The target contract, in turn, is allowed to extract any payment it wishes from the user, e.g. tokens.

[6] In the protocol, the addresses become verification keys for a signature scheme.

The primary difference is in the complexity of implementation and development; where the GSN aims to be fully generic and decentralized, and admits a great deal of complexity in service of that aim, we have endeavored to keep our efforts very self-contained in order to ease implementation, simplify formal analysis, and keep operational costs manageable.

As is common in designs that aim for maximally generic functionality, the GSN pays for its genericity with increased complexity. This complexity manifests both in development effort—anecdotally, we found setting up a testing environment for a GSN-enabled contract to be significantly more cumbersome than for other contracts—and in gas consumption. Our experiments indicate a 4-5x overhead in gas consumption when using the GSN as opposed to using our EET contract. (Note that gas is pretty much the only relevant measurable unit of comparison. Other metrics—e.g. running time, settlement time, etc.—are either very difficult to test in a controlled way, are irrelevant for a contract which aims only to facilitate token exchange, or are negligible compared to other confounding factors.) We note in passing that, to our knowledge, there is no formal security analysis of the GSN, making our work the first rigorous treatment of the etherless token paradigm.

Contract-Based vs Native Tokens. Recently, the blockchain/cryptocurrency community has been entertaining the idea of making tokens native to the cryptocurrency chain. In parallel and independent work [8] the authors propose a solution that allows users posting a token transaction along with a token-to-native exchange rate he is willing to pay; at the same time anyone could issue transactions aimed at covering the fee of such token transactions in exchange of tokens coins. Then any miner/minter that can match such transactions (if any valid match exists) can create a block that contains both transactions, in which the fee for the token transaction has been payed by a third party. A similar solution has been proposed in [19]. Such approach yields an advantage in terms of fees needed for the transaction, but it does come at a cost: (1) The block miner are in an advantageous position and can always front-run other users proposing their own transactions to cover for the fees of token transaction; (2) The token functionality is limited to what is hardwired on the token chain, and is therefore far less flexible than a smart-contract-based solution. For example, it is unclear if or how such a solution would allow the use of amortization/batching to save on bulk transactions. (3) If one adopts the natural "pay-per-use" principle for fees—i.e. you pay more for a more complex transaction—as Ethereum does, then adding this functionality would increase the cost of all transactions, including those that only involve the native cryptocurrency. Although this increase is expected to be minimal, it is unclear how the implicit auction for the submitted token transaction created by such a mechanism would affect fees.

In the full version [3], we have included an attempt to estimate the overhead this might incur in a hypothetical implementation on Ethereum, and compare it with using a smart contract. We note that in the absence of a (platform or blockchain supporting an) actual implementation of native tokens, the relevant experiments are somewhat artificial and speculative. Thus, we do not consider these experiments an important part of our contributions (and we defer them to the appendix). Nonetheless, we do believe they give an interesting perspective to the discussion on native tokens, and a pointer for experiments once such a functionality is implemented on a mainstream blockchain. Finally, we stress that our solution works on all blockchains that support token contracts (i.e., no need for turing completeness) like Cardano, Dfinity and Ethereum, whereas the solution proposed in [8] would require to fork an existing blockchain to accommodate for a new validation rule.

3 Preliminaries and Model

We denote a randomized assignment is denoted with $a \overset{\$}{\leftarrow} A$, where A is a randomized algorithm. We use existentially unforgeable and non-repudiable signatures [7]. A signature scheme is a triple of PPT algorithms $\Sigma = (\texttt{Gen}, \texttt{Sign}, \texttt{Ver})$ where $(s, v) \overset{\$}{\leftarrow} \texttt{Kgen}(1^\lambda)$ generates a secret-key/public key pair, $\sigma \overset{\$}{\leftarrow} \texttt{Sign}(s, m)$ generates a signature and $\texttt{Ver}(v, m, \sigma)$ verifies that σ is a valid signature. We refer to the full version for the formal definition. We provide our protocols and security proofs in Canetti's universal composition (UC) framework [6]. We assume that the reader is familiar with simulation-based security and has basic knowledge of the UC framework. For more detail, we refer to the full version [3]. We now elaborate on the main hybrid functionality used in our paper.

The functionality $\mathcal{F}_{\textbf{ledger}}$. The main functionality (in fact, a global setup) we rely on is a cryptographic distributed transaction ledger. We use the (backbone) ledgers proposed in the recent literature [4,5] in order to describe a transaction ledger and its properties. As proved in [4,5], such a ledger is implemented by known permissionless blockchains based on either proof-of-work (PoW), e.g. Bitcoin, or poof-of-stake (PoS), e.g. Ouroboros Genesis. The ledger stores an immutable sequence of blocks called *state*—each block containing several messages typically referred to as *transactions* and denoted by tx—which is accessible from the parties under some restrictions discussed below. It enforces the following basic properties that are inspired by [10,16]:

- *Ledger growth.* The size of the ledger's state should grow—new blocks should be added—as the rounds advance.
- *Chain quality.* It is guaranteed that a percentage of honest blocks are created in a sufficiently long sequence of blocks.
- *Transaction liveness.* Old enough (valid) transactions are included in the next block added to the ledger state.

We next give a brief overview of the ledger functionality $\mathcal{F}_{\text{LEDGER}}$ proposed in [4,5], focusing on the properties of $\mathcal{F}_{\text{LEDGER}}$ that are relevant for the understanding our results. Along the way we also introduce some useful notation and terminology. We refer the reader interested in the low-level details of the ledger functionality and its UC implementation to the full version [3] and [4,5]. We note that with minor differences related to the nature of the resource used to implement the ledger, PoW vs PoS, the ledgers proposed in these works are identical.

The functionality $\mathcal{F}_{\text{LEDGER}}$ is parametrized by three main functions $\mathsf{Validate}$, $\mathsf{ExtendPolicy}$ and $\mathsf{Blockify}$. At a high level, anyone (honest miner or the adversary) may submit a transaction to $\mathcal{F}_{\text{LEDGER}}$. The transaction is validated by means of a filtering predicate $\mathsf{Validate}$, and if it is found to be valid it is added to a *buffer* that we denote `buffer`. Taking a peak at the actual implementation of the ledger, this buffer contains transactions that, although validated, are either not yet inserted into a valid block, or are in a block which is not yet deep enough in the blockchain to be considered immutable for an adversary. The adversary $\mathcal{A}$ is informed that the transaction was received and is given its contents. Periodically, $\mathcal{F}_{\text{LEDGER}}$ does the following: (1) fetches some of the transactions in the buffer under the influence of the adversary (more on this will follow), (2) modifies them by means of a procedure $\mathsf{Blockify}$, (3) creates a block including the output of $\mathsf{Blockify}$, and (4) adds this block to its permanent state, denoted as `state`. `state` is a data structure that includes the sequences of blocks that the adversary can no longer change. (In [10,16] this corresponds to the *common prefix.*) Any miner or the adversary is allowed to request a read of the contents of the state and, every honest miner will eventually receive `state` as its output.[7] To enforce transaction liveness and chain-quality, $\mathcal{F}_{\text{LEDGER}}$ relies on the function $\mathsf{ExtendPolicy}$. At a high level, $\mathsf{ExtendPolicy}$ makes sure that the adversary cannot create too many blocks with arbitrary (but valid) contents (chain quality) and that if a transaction is old enough, and still valid with respect to the actual state, then it is included into the state. In more detail, $\mathsf{ExtendPolicy}$ takes the current contents of the buffer, along with the adversary's recommendation `NxtBC`, and the block-insertion times vector $\tau_{\texttt{state}}$. The latter is a vector listing the times when each block was inserted into the state. The output of $\mathsf{ExtendPolicy}$ is a vector including the blocks to be appended to the state during the next state-extend time-slot. Each of these blocks is then given as input to $\mathsf{Blockify}$. We conclude the discussion by providing a high-level description of the main input command of $\mathcal{F}_{\text{LEDGER}}$ used in our protocols/definitions, and refer to Sect. 3 for a formal description of the functionality.

– The input $(\text{READ}, \text{sid})$ is used to request the content of the ledger's `state`. Concretely, upon receiving $(\text{READ}, \text{sid})$ from some party (or the adversary on behalf of a corrupted party), the ledger returns (a prefix of) `state` to the caller.

[7] As observed in [5], it is not possible to guarantee with existing constructions that at any given point in time all honest parties see exactly the same state (blockchain) length, so each party may have a different view of the state which is defined by the adversary. However, the adversary can restrict the view of the honest parties only by a bounded number of blocks. The parameter that defines such a bound is called `windowSize`.

– The input $(\text{SUBMIT}, \text{sid}, \mathtt{tx})$ is used to request that a transaction $\mathtt{tx}$ be added to the buffer. That is, upon receiving a $(\text{SUBMIT}, \text{sid}, \mathtt{tx})$ message from any party (or the adversary), the ledger adds the transaction $\mathtt{tx}$ to the buffer $\mathtt{buffer}$. If the validation predicate $\mathsf{Validate}$, on input $\mathtt{state}, \mathtt{buffer}, \mathtt{tx}$ outputs 1, then $\mathtt{tx}$ will be included in $\mathtt{state}$.[8] The time required for the transaction to be part of $\mathtt{state}$ and visible to all honest parties who query $\mathcal{F}_{\text{LEDGER}}$ depends on the transaction liveness parameter defined in $\mathsf{ExtendPolicy}$.

4 The Cryptocurrency-Ledger Functionality $\mathcal{F}_{\text{T-LEDGER}}$

The ledger $\mathcal{F}_{\text{LEDGER}}$ does not itself realize a cryptocurrency (unless if couple with a signature scheme as described in [5]). To this direction we define and instantiate a cryptocurrency (transaction) ledger $\mathcal{F}_{\text{T-LEDGER}}$ hosting a coin denoted by $\mathtt{E}$. As discussed in the introduction, in contrast to the transaction ledger from [5] our construction does not assume an external signature functionality. This makes it more useful for defining smart contracts (see Sect. 5).

The validation predicate of $\mathcal{F}_{\text{LEDGER}}$, in this case, is defined to always output 1, and it is $\mathcal{F}_{\text{T-LEDGER}}$'s responsibility to make sure that only valid transactions are submitted to $\mathcal{F}_{\text{LEDGER}}$. $\mathcal{F}_{\text{T-LEDGER}}$ also generates and manages the *wallets* of the parties. A transaction supported by $\mathcal{F}_{\text{T-LEDGER}}$ consists of five main components $(v, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{aux}, \mathsf{fee})$, where v represents the amount of coins of type $\mathtt{E}$, addr_i is the sender's wallet address, addr_j is the receiver's wallet address, aux is a payload, and fee represents the fee. At a high level, a transaction is valid if the fee fee is high enough and if the amount of coins stored in the wallet with address addr_i is at least $v + \mathsf{fee}$. How high the fee should be in order for the transaction to be considered is specified by a function f that is part of the description of $\mathcal{F}_{\text{T-LEDGER}}$. f takes as input the transaction $\mathtt{tx}$ and computes the required fee. In the case where the output of f is greater than fee, the transaction is immediately discarded. Otherwise, $\mathcal{F}_{\text{T-LEDGER}}$ replaces fee with the output of the function and submits it. This captures the fact that $\mathcal{F}_{\text{T-LEDGER}}$ charges the issuer of the transaction only for the cost of processing the transaction, even if the transaction specifies a higher fee. In more detail, each party has an associated wallet address, and different parties have different wallet addresses. $\mathcal{F}_{\text{T-LEDGER}}$ manages a table $\mathcal{T}$ that, for each party P_i, stores P_i's wallet address addr_i. We initialize $\mathcal{F}_{\text{T-LEDGER}}$ with a party P_0 which initially holds all the coins (e.g., V coins) of type $\mathtt{E}$[9]. To do so, $\mathcal{F}_{\text{T-LEDGER}}$ generates an address addr_0 and sends $(\text{SUBMIT}, \text{sid}, \mathtt{tx})$ to the wrapped $\mathcal{F}_{\text{LEDGER}}$ with $\mathtt{tx} := (V, 0^\lambda, \mathsf{addr}_0, \perp, 0)$, where V is the initial amount of coins held by P_i and 0^λ is a special address used only for the initialization. Upon receiving a registration request from a party P_i, $\mathcal{F}_{\text{T-LEDGER}}$ creates a new wallet address addr_i and adds (addr_i, P_i) to the table

[8] We have the guarantee that any transaction (either generated by a malicious or honest party) that manages to go in $\mathtt{buffer}$ will eventually be included in $\mathtt{state}$.

[9] It is easy to intialize the functionality with an arbitrary number of parties that hold an initial amount of coin. To simplify the description on the functionality, we decided to use only one party in this phase.

$\mathcal{T}$. $\mathcal{F}_{\text{T-LEDGER}}$, upon receiving $(\text{SUBMIT}, \text{sid}, \texttt{tx})$ from a party P_i, performs the following steps.

– Parse $\texttt{tx}$ as $(v, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{aux}, \mathsf{fee})$ and continue if and only if $(P_i, \mathsf{addr}_i) \in \mathcal{T}$ and $\mathsf{fee} \geq f(\texttt{tx})$.
– Get $\texttt{state}$ and $\texttt{buffer}$ of $\mathcal{F}_{\text{LEDGER}}$ and check that the balance of transactions to/from the wallet address addr_i is at least $v' \geq v + f(\texttt{tx})$ coins. That is, the sum of coins with receiver address addr_i minus the sum of coins in the state with sender address addr_i (including the fees) is greater than or equal to $v + f(\texttt{tx})$. If this is not the case, deem the transaction invalid; otherwise, submit $\texttt{tx}$ to $\mathcal{F}_{\text{LEDGER}}$ with the fee $f(\texttt{tx})$.

$\mathcal{F}_{\text{T-LEDGER}}$ is also parametrized with the identifier of an ideal functionality $\mathcal{F}_{\mathsf{trap}}$. Whenever $\mathcal{F}_{\text{T-LEDGER}}$ receives the command $(\text{SUBMIT-TRAPDOOR}, \text{sid}, \texttt{tx}, P_i)$ from $\mathcal{F}_{\mathsf{trap}}$, it forwards the transaction $\texttt{tx}$ on behalf of P_i to $\mathcal{F}_{\text{LEDGER}}$ without checking anything about $\texttt{tx}$ in terms of balances and fees. This simple mechanism allows $\mathcal{F}_{\text{T-LEDGER}}$ to interact with other ideal functionalities when required. This becomes particularly helpful when we want to enhance the behavior of $\mathcal{F}_{\text{T-LEDGER}}$ with smart contracts, and in the next section we show how to do that. For all the other input commands, $\mathcal{F}_{\text{T-LEDGER}}$ just acts as a proxy between $\mathcal{F}_{\text{LEDGER}}$ and its external interface. To conclude the description of $\mathcal{F}_{\text{T-LEDGER}}$, we need to specify how $\mathsf{Blockify}$ works. $\mathsf{Blockify}$ is a simple procedure that takes as input the next block to be added to the state, and outputs a concatenation of the transactions contained in the block. This means that the state of $\mathcal{F}_{\text{LEDGER}}$ (which will correspond also to the state of $\mathcal{F}_{\text{T-LEDGER}}$) is represented by just list of transactions. We do not specify how $\mathsf{ExtendPolicy}$ works, as any realization of $\mathsf{ExtendPolicy}$ can be used in our formalization. We provide a more detailed description of $\mathcal{F}_{\text{T-LEDGER}}$ in the full version [3]. We note that $\mathcal{F}_{\text{T-LEDGER}}$ does not specify who gets the fee, but this would not be difficult to do since $\mathcal{F}_{\text{LEDGER}}$ keeps track of the party that generated each block. Hence, it would be easy to modify $\mathcal{F}_{\text{T-LEDGER}}$ to keep track of which party gets the fees of the transactions that constitute a block. Another simplification we make is to consider fixed relation between the cost required to execute a transaction (or call a contract as we will see) and the complexity of the transaction (or the contract call). In system like Ethereum this is not the case, as the fee that a party pays depends on the complexity of the transaction (which determines the amount of gas) and on the gas price. This means that how fast and if a transaction will be executed depends on the product of gas price and amount of required gas. We could modify $\mathcal{F}_{\text{T-LEDGER}}$ (and the other functionalities we will consider) to accommodate for an additional mechanism that allows the adversary communicating to the functionality the average gas price, in such a say that we can use this gas cost to decide whether to accept or reject a transaction. However, since these aspects are not relevant for our results, to simplify the description of our already involved ideal functionalities, we have decided to not include such mechanisms in our model.

5 The Smart-Contract-Enabled Transaction Ledger

In this section we define the functionality $\mathcal{F}_{\text{TSC-LEDGER}}$ that, in addition to $\mathcal{F}_{\text{T-LEDGER}}$, captures a ledger that enables a large class of smart contracts. $\mathcal{F}_{\text{TSC-LEDGER}}$ internally runs $\mathcal{F}_{\text{T-LEDGER}}$ and a smart contract (formally defined by means of an additional ideal functionality). The contract has a state that can be updated by any party that can afford to pay a fee (that depends on the contract and on the input). After any valid update, the new contract state is pushed onto the $\mathcal{F}_{\text{T-LEDGER}}$'s state. As we have alluded, in order for the contract to freely interact with $\mathcal{F}_{\text{T-LEDGER}}$, the parameter $\mathcal{F}_{\mathsf{trap}}$ of $\mathcal{F}_{\text{T-LEDGER}}$ is set to be equal to the identity of $\mathcal{F}_{\text{TSC-LEDGER}}$, which will act as a bridge between the contract functionality and $\mathcal{F}_{\text{T-LEDGER}}$. To simplify the description of the functionality, we describe the case where only one smart contract is running; however, it is easy to extend the functionality to the case where multiple smart contracts are running at the same time. A smart contract $\mathcal{F}_{\text{SC}}$ is a small functionality managed by $\mathcal{F}_{\text{TSC-LEDGER}}$ that maintains its own state $\mathtt{cstate}$. The behavior of $\mathcal{F}_{\text{SC}}$ is fully determined by three procedures: f_{CFee}, f_{filter} and f_{trans}.

- f_{CFee} (the contract fee function) takes as input the contract state $\mathtt{cstate}$, the ledger state of $\mathcal{F}_{\text{T-LEDGER}}$, a transaction, (which represents the input received by the contract's caller) and the fee specified in the input transaction. If the fee indicated is sufficient to update the contract state, then f_{CFee} returns the actual fee required to run the contract (which could be less than the fee indicated by the contract's caller). If the submitted fee is not sufficient, then the function returns $\perp$.
- f_{trans} (the state transition function) takes as input the payload of the input transaction, $\mathcal{F}_{\text{T-LEDGER}}$'s state, and the contract state $\mathtt{cstate}$, and returns a new contract state updated according to its inputs.
- f_{filter} (the filtering function) takes as input (1) the view that the contract's caller has of $\mathcal{F}_{\text{T-LEDGER}}$'s state $\mathtt{state}_i$ and (2) the contract state, and returns an arbitrary sub-set of the information contained in $\mathtt{state}_i$.

The functionality $\mathcal{F}_{\text{TSC-LEDGER}}$ is also parametrized by Fee, which represents the minimum fee that a party should pay in order to query a contract (to update the contract the fee might be higher). In more detail, $\mathcal{F}_{\text{TSC-LEDGER}}$ accepts transactions with the following format: $\mathtt{tx} := (v, \mathsf{addr}_i^{\mathtt{E}}, \mathsf{addr}_j^{\mathtt{E}}, \mathsf{aux}, \mathsf{fee}, \mathsf{type})$, where $\mathsf{type} \in \{\mathtt{E}, \mathsf{SC}\}$ denotes whether the transaction should be treated as a normal transaction or as a call to the contract. In particular, $\mathcal{F}_{\text{TSC-LEDGER}}$ checks whether $\mathsf{type} = \mathtt{E}$ or $\mathsf{type} = \mathsf{SC}$. In the former case, $\mathcal{F}_{\text{TSC-LEDGER}}$ removes the field type from the transaction and forwards it to $\mathcal{F}_{\text{T-LEDGER}}$. In the latter, $\mathcal{F}_{\text{TSC-LEDGER}}$ checks that $\mathsf{fee} \geq \mathsf{Fee}$ and that the issuer of the transaction has at least fee coins of type $\mathtt{E}$ in its wallet[10]. If this check is successful, then $\mathcal{F}_{\text{TSC-LEDGER}}$ forwards the transaction and the current ledger state to $\mathcal{F}_{\text{SC}}$, which does the following: It uses f_{CFee} to check whether the fee specified in $\mathtt{tx}$ minus the fee required to query the contract (denoted with Fee) would be sufficient to update the contract state using

[10] $\mathcal{F}_{\text{TSC-LEDGER}}$ can do this check since it has full access to $\mathcal{F}_{\text{T-LEDGER}}$'s state and buffer.

the input aux. If f_{CFee} returns $\perp$, then the contract returns $(\mathtt{ko}, \mathtt{cstate}, \mathsf{fee})$. Else, if f_{CFee} returns $\mathsf{fee}^{\mathsf{SC}}$, $\mathcal{F}_{\mathrm{SC}}$ computes the updated contract state $\mathtt{cstate}$ by running f_{trans} on input the payload of $\mathtt{tx}$ (denoted with aux), the ledger state, and the contract state, and returns $(\mathtt{ok}, \mathtt{cstate}, \mathsf{fee}^{\mathsf{SC}} + \mathsf{Fee})$. $\mathcal{F}_{\text{TSC-LEDGER}}$ upon receiving $(\mathsf{Flag}_{\mathsf{C}}, \mathtt{cstate}, \mathsf{actualfee})$ from $\mathcal{F}_{\mathrm{SC}}$, constructs and sends to $\mathcal{F}_{\text{T-LEDGER}}$ the transaction $\mathtt{tx}^{\mathtt{E}} := (0, \mathsf{addr}_i^{\mathtt{E}}, 0^{\lambda}, (\mathsf{Flag}_{\mathsf{C}}, \mathsf{aux}, \mathtt{cstate}, \mathcal{F}_{\mathrm{SC}}.\mathsf{id}), \mathsf{actualfee})$ using the command SUBMIT-TRAPDOOR, where we recall that aux is the payload of $\mathtt{tx}$, $\mathsf{Flag}_{\mathsf{C}} \in \{\mathtt{ok}, \mathtt{ko}\}$, and $\mathcal{F}_{\mathrm{SC}}.\mathsf{id}$ is the identifier of SC. We note that the transaction $\mathtt{tx}^{\mathtt{E}}$ is a standard $\mathcal{F}_{\text{T-LEDGER}}$ transaction that contains in its payload the updated state of the contract (or the old state if the fee was not sufficient), the input used to eventually update the contract's state, and the fee $\mathsf{actualfee}$ such that:

– if $\mathsf{Flag}_{\mathsf{C}} = \mathtt{ko}$ (i.e. the fee specified by the contract's caller was not sufficient to update the contract state) then $\mathsf{actualfee} = \mathsf{fee}$
– if $\mathsf{Flag}_{\mathsf{C}} = \mathtt{ok}$ (i.e. fee was sufficient to update the contract's state) then $\mathsf{actualfee} \leq \mathsf{fee}$.

Note that it might be that $\mathsf{actualfee} < \mathsf{fee}$ in the case where the fee required to update the contract state is less that fee. That is, $\mathcal{F}_{\text{TSC-LEDGER}}$ only charges the contract caller exactly for the fee required to run the contract. When fee is insufficient to complete execution of the contract, the issuer of the transaction pays the full amount of fee even though no change to the contract state is committed. (This is consistent with Ethereum and other blockchains that support Turing-complete smart contracts.) We refer to the full version [3] for a more detailed description of $\mathcal{F}_{\text{TSC-LEDGER}}$ and for the abstraction of $\mathcal{F}_{\mathrm{SC}}$.

6 The EET Ledger

We can now define the functionality $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$. $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ internally runs $\mathcal{F}_{\text{TSC-LEDGER}}$, parametrized by a contract $\mathcal{F}^{\mathtt{T}}_{\mathrm{SC}}$. $\mathcal{F}^{\mathtt{T}}_{\mathrm{SC}}$ maintains a token $\mathtt{T}$, and allows parties to issue transactions with respect to such a token. Any party that has some tokens can sent it to another party by querying the contract $\mathcal{F}^{\mathtt{T}}_{\mathrm{SC}}$. However, invoking the contract requires payment of a fee in the native currency $\mathtt{E}$, even if the transaction involves only tokens. To mitigate this problem, our functionality allows a sender P_i to send tokens to another party P_j, even if P_i does not have native coins. In particular, the sender will pay a fee of at least $\mathsf{del\text{-}fee}$ tokens $\mathtt{T}$ to a special party M, called the intermediary, and M will pay the fee in $\mathtt{E}$ on the behalf of the sender ($\mathsf{del\text{-}fee}$ is a fixed amount of tokens that parametrizes our functionality). The functionality guarantees that either the transaction by P_i becomes part of the ledger state *and* M gets a fixed amount of tokens $\mathsf{del\text{-}fee}$, or nothing happens. We propose a more detailed description of $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ and $\mathcal{F}^{\mathtt{T}}_{\mathrm{SC}}$ to the full version [3], and provide a more high level (but still formal) description of those functionalities below. The functionality $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$, interacts with a set of parties, with the adversary, and with a special party that we denote with M (the intermediary), and manages the *token wallet* addresses of the registered parties. We assume that a party P_0 initially holds all of the available tokens[11]. We denote the token wallet addresses of P_0 and M with $\mathsf{addr}^{\mathtt{T}}_0$ and $\mathsf{addr}^{\mathtt{T}}_{\mathsf{M}}$ respectively. Any time $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ receives a registration command from a party P_i, it registers P_i to the ledger $\mathcal{F}_{\text{TSC-LEDGER}}$, thus obtaining $\mathsf{addr}^{\mathtt{E}}_i$. It then generates a token wallet address $\mathsf{addr}^{\mathtt{T}}_i$ and returns $(\mathsf{addr}^{\mathtt{E}}_i, \mathsf{addr}^{\mathtt{T}}_i)$ to P_i. $(\mathsf{addr}^{\mathtt{E}}_i, \mathsf{addr}^{\mathtt{T}}_i)$ represents respectively the wallet addresses for the native currency $\mathtt{E}$ and for the token $\mathtt{T}$. $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ tolerates two types of transactions: *standard* and *delegated* transactions. Any registered party P_i can issue a standard transaction $\mathtt{tx}^{\mathtt{T}} := (v, \mathsf{addr}^{\mathtt{E}}_i, \mathsf{addr}^{\mathtt{T}}_i, \mathsf{addr}^{\mathtt{T}}_j, \mathsf{fee})$, where v denotes the amount of tokens, $(\mathsf{addr}^{\mathtt{E}}_i, \mathsf{addr}^{\mathtt{T}}_i)$ are the addresses of the sender, $\mathsf{addr}^{\mathtt{T}}_j$ is the token wallet address of the receiver, and fee is the fee expressed in coins of type $\mathtt{E}$. $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ takes $\mathtt{tx}^{\mathtt{T}}$ and creates a transaction $\mathtt{tx}^{\mathtt{E}}$ for the ledger $\mathcal{F}_{\text{TSC-LEDGER}}$ that (1) has as a sender address $\mathsf{addr}^{\mathtt{E}}_i$, (2) has a fee fee, and (3) calls the contract $\mathcal{F}^{\mathtt{T}}_{\text{SC}}$ and includes in its payload what we call a *token transaction* $\mathtt{tx}' := (v, \mathsf{addr}^{\mathtt{T}}_i, \mathsf{addr}^{\mathtt{T}}_j)$.[12] $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ then forwards $\mathtt{tx}^{\mathtt{E}}$ to the ledger $\mathcal{F}_{\text{TSC-LEDGER}}$ on behalf of P_i. The contract $\mathcal{F}^{\mathtt{T}}_{\text{SC}}$ maintains a set $\mathtt{token\text{-}set}$ as part of its state, and if the fee specified in $\mathtt{tx}^{\mathtt{E}}$ is sufficient, it updates its state by adding $\mathtt{tx}'$ to $\mathtt{token\text{-}set}$ and returns $(\mathtt{ok}, \mathtt{cstate}, \mathsf{actualfee})$. Note that this means that the $\mathtt{tx}'$ is part of the contract state and appears in the $\mathcal{F}_{\text{TSC-LEDGER}}$'s state by definition. To complete this first part of the description of $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$, it remains to specify the function f_{filter} (and f_{CFee}, which we describe later in this section) of $\mathcal{F}^{\mathtt{T}}_{\text{SC}}$. f_{filter} receives as input the contract state and the state of $\mathcal{F}_{\text{TSC-LEDGER}}$ (which we denote $\mathtt{state}$) and, for each transaction $\mathtt{tx}$ in $\mathtt{state}$ such that $\mathtt{tx}^{\mathtt{E}} := (0^\lambda, \mathsf{addr}^{\mathtt{E}}_i, 0^\lambda, \mathsf{aux}^{\mathtt{E}}, \mathsf{fee}^{\mathtt{E}})$ (where $\mathsf{aux}^{\mathtt{E}} = (\mathtt{ok}, \mathtt{tx}', \mathtt{cstate}^\star, \mathcal{F}^{\mathtt{T}}_{\text{SC}}.\mathsf{id})$), adds $\mathtt{tx}'$ to $\mathtt{state}^{\mathtt{T}}$ if and only if:

1. $\mathtt{tx}'$ appears in $\mathtt{token\text{-}set}$ (which is part of the token state).
2. $\mathtt{tx}' := (v, \mathsf{addr}^{\mathtt{T}}_i, \mathsf{addr}^{\mathtt{T}}_j)$ and the sum of tokens in the token transactions stored so far in $\mathtt{state}^{\mathtt{T}}$ with receiver address $\mathsf{addr}^{\mathtt{T}}_i$, minus the sum of coins in the state with sender address $\mathsf{addr}^{\mathtt{T}}_i$, is greater than or equal to v.

$\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ captures the main characteristics of a token, relying on the smart contract to filter out invalid transactions. Unfortunately, the mechanism that we have discussed so far has a major drawback: if a party wants to issue a token transaction, they must have the required amount of coins of type $\mathtt{E}$ to query the contract. To get rid of this requirement, $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ admits what we call *delegated transactions*. A party that wants to issue a delegated transaction submits $\mathtt{tx}^{\mathtt{T}} := (v, \mathsf{addr}^{\mathtt{T}}_i, \mathsf{addr}^{\mathtt{T}}_j, \mathsf{fee}^{\mathtt{T}})$ to $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$, which in turns asks the special party denoted M to pay the fee in $\mathtt{E}$ in exchange of (at least) $\mathsf{del\text{-}fee}$ tokens $\mathtt{T}$, which will be taken from P_i's account. If M is honest and $\mathsf{fee}^{\mathtt{T}} \geq \mathsf{del\text{-}fee}$, (where we recall that $\mathsf{del\text{-}fee}$ is the minimum fee required for the delegation to be considered,) then $\mathcal{F}^{\mathsf{EET}}_{\text{LEDGER}}$ submits a call to the contract $\mathcal{F}^{\mathtt{T}}_{\text{SC}}$ on behalf of M with the input (the payload of the transaction) $\mathsf{aux} := (([v, \mathsf{fee}^{\mathtt{T}}], \mathsf{addr}^{\mathtt{T}}_i, [\mathsf{addr}^{\mathtt{T}}_j, \mathsf{addr}^{\mathtt{T}}_{\mathsf{M}}]))$. If M

[11] As before, we could have multiple addresses having different amounts of tokens, but for simplicity, we assume that only one party initially holds tokens.

[12] The payload also includes an identifier chosen by the adversary, which we omit in this informal description.

has enough coins of type $\mathtt{E}$ to afford the call to $\mathcal{F}_{\text{SC}}^{\mathtt{T}}$, then aux will become part of the contract state. To accommodate for this special input, we modify the filtering function f_{filter} of $\mathcal{F}_{\text{SC}}^{\mathtt{T}}$ in such a way that the value aux can also be understood as two atomic token transactions: the first moves v tokens from the wallet address $\mathsf{addr}_i^{\mathtt{T}}$ to the wallet address $\mathsf{addr}_j^{\mathtt{T}}$, and the second moves $\mathsf{fee}^{\mathtt{T}}$ from the wallet address $\mathsf{addr}_i^{\mathtt{T}}$ to the wallet address $\mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}$. It remains to specify how the contract computes the fee. The function f_{CFee} charges Fee coins of type $\mathtt{E}$ for each token transaction encoded in aux (the input that is used to update the contract state). Hence, for a non-delegated token transaction, f_{CFee} would return Fee, and for a delegated token transaction, it would return $2\mathsf{Fee}$. In addition to this fee, we need to consider the fee required simply to query the contract. Hence, the total cost of a non-delegated transaction would be of $2\mathsf{Fee}\mathtt{E}$, and the total cost of a delegated transaction would be $3\mathsf{Fee}\mathtt{E}$. We stress that this is a simplified method of computing the fee, and that a more fine-grained calculation could be used to capture what actually happens in the real world.

Constructions and Experimental Evaluation. We have already highlighted how our construction works in Sect. 2.2. We compared our system with GSN and with users that make only *self-funded transactions* (i.e., user that do not want to interact with the intermediary and have his own Ether to afford for the token transaction). Our experiments indicate a 4-5x overhead in gas consumption when using the GSN as opposed to using our EET contract. This is the cost of the complexity of the GSN, a cost that is very unattractive for projects that do not require the genericity of the GSN. We also show that our contract consumes less than twice the gas of a standard self-funded token transaction, which we believe is a reasonable compromise for the added user experience. We refer the reader to the full version [3] for more detail on our experimental results.

A Our Protocol: How to Realize $\mathcal{F}_{\text{LEDGER}}^{\mathsf{EET}}$

Our protocol is described in the $\mathcal{F}_{\text{T-LEDGER}}$-hybrid world, where $\mathcal{F}_{\text{T-LEDGER}}$ is parametrized by $\mathcal{F}_{\mathsf{trap}} = \perp$, and the fee function f which, upon receiving an input transaction $\mathtt{tx}^{\mathtt{E}}$, does the following: 1) Parse $\mathtt{tx}$ as $(v, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{aux}, \mathsf{fee})$; 2) if $\mathsf{aux} = \perp$, then return Fee; 3) Otherwise, return $\mathsf{Fee} + |\mathsf{aux}|/\kappa\mathsf{Fee}$. In a nutshell, the fee required for a transaction to settle in the $\mathcal{F}_{\text{T-LEDGER}}$'s state is Fee, plus and additional Fee for each κ bits contained in the payload, where Fee and κ are part of the description of f. We provide the formal description of our protocol in Fig. 2. At a very high level, the protocol works as follows: Each party registers with $\mathcal{F}_{\text{T-LEDGER}}$ and runs $\mathtt{Kgen}(1^{\lambda})$ to obtain $(\mathsf{sk}_i^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}})$, where $\mathsf{addr}_i^{\mathtt{T}}$ represents the token wallet address. A party P_i that wants to send $v\mathtt{T}$ to P_j and has at least $2\mathsf{Fee}$ coins of type $\mathtt{E}$ can do so by issuing a transaction for $\mathcal{F}_{\text{T-LEDGER}}$ that contains in its payload $\mathsf{aux} := (v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{id}, \sigma_i^{\mathtt{T}})$, where id is a random value, and $\sigma_i^{\mathtt{T}}$ is a signature of $(v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{id})$ that verifies under the verification key $\mathsf{addr}_i^{\mathtt{T}}$. We require P_i to pay a fee of at least $2\mathsf{Fee}$ because we assume that, in this case, $|\mathsf{aux}| = \kappa$. When an honest party P_i receives the command $(\textsc{read}, \mathrm{sid}, \mathtt{T})$, they shall retrieve $\mathcal{F}_{\textsc{T-Ledger}}$'s state, filter out the payload of each transaction (thus obtaining only the information related to token transactions), and output only the *valid* token transactions. A token transaction $(v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{id}, \sigma_i^{\mathtt{T}})$ is valid if $\mathsf{addr}_i^{\mathtt{T}}$ has received at least v tokens, $\sigma_i^{\mathtt{T}}$ is a signature of $(v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{id})$ that verifies under the verification key $\mathsf{addr}_i^{\mathtt{T}}$, and there does not exist any other token transaction with the same sender address and identifier id. Our protocol allows any party P_i that does not have coins of type $\mathtt{E}$ to delegate the payment of the fee to M, paying M with at least $\mathsf{del\text{-}fee}$ tokens $\mathtt{T}$. To do so, P_i creates $m := ([v, \mathsf{del\text{-}fee}], \mathsf{addr}_i^{\mathtt{T}}, [\mathsf{addr}_j^{\mathtt{T}}, \mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}], \mathsf{id})$ and signs it, thus obtaining $\sigma_i^{\mathtt{T}}$. P_i then sends $(m, \sigma_i^{\mathtt{T}})$ to M. The honest M then creates a transaction for $\mathcal{F}_{\textsc{T-Ledger}}$ that includes $(m, \sigma_i^{\mathtt{T}})$ in its payload and has a fee of at least $3\mathsf{Fee}$, and submits it. We require M to pay a fee of at least $3\mathsf{FeeE}$ because we assume that, in this case, the payload of the transaction is 2κ bits (as, indeed, the payload of this type of transaction contains more information). The honest M would immediately create and submit such a transaction, whereas the corrupted M might decide when (and if) to create the transaction. We require each token transaction to contain a random identifier in order to avoid replay attacks; without such an identifier, the adversary could take the payload of any transaction from $\mathcal{F}_{\textsc{T-Ledger}}$'s state, (for instance, the payload of a transaction that moves v tokens from the address $\mathsf{addr}_i^{\mathtt{T}}$ of an honest party to some potentially adversarial address,) copy this payload, and use it to generate a new transaction for $\mathcal{F}_{\textsc{T-Ledger}}$. In this way, the adversary could empty the token wallet of the honest party without their knowledge. The other advantage of using identifiers is that an honest party that has delegated a transaction to a malicious intermediary can at any point decide to withdraw the delegation. Indeed, if M is not responding to a party that has delegated the transaction $m := ([v, \mathsf{del\text{-}fee}], \mathsf{addr}_i^{\mathtt{T}}, [\mathsf{addr}_j^{\mathtt{T}}, \mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}], \mathsf{id})$ for a long time, and m does not appear in the payload of any transaction that appears in the ledger's state, then P_i can withdraw the delegation by submitting (or delegating) a token transaction with the same identifier; then, at most one of these transactions will be valid and accepted by the functionality. We refer to Fig. 2 for the formal description of Π^{Token}.

Protocol Π^{Token}

- The issuer P_0 does the following:
  1. Register to $\mathcal{F}_{\text{T-LEDGER}}$, thus obtaining addr_0.
  2. Compute $(\mathsf{sk}_0^{\mathtt{T}}, \mathsf{addr}_0^{\mathtt{T}}) \xleftarrow{\$} \mathtt{Kgen}(1^\lambda)$.
- The intermediary M registers to $\mathcal{F}_{\text{T-LEDGER}}$, thus obtaining $\mathsf{addr}_{\mathsf{M}}$, and computes $(\mathsf{sk}_{\mathsf{M}}^{\mathtt{T}}, \mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}) \xleftarrow{\$} \mathtt{Kgen}(1^\lambda)$.

- Upon receiving $(\text{REGISTER}, \text{sid})$, the party P_i sends $(\text{REGISTER}, \text{sid})$ to $\mathcal{F}_{\text{T-LEDGER}}$, thus obtaining addr_i, and computes $(\mathsf{sk}_i^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}}) \xleftarrow{\$} \mathtt{Kgen}(1^\lambda)$.

- P_i, upon receiving $(\text{SUBMIT-DELEGATION}, \text{sid}, \mathtt{tx}^{\mathtt{T}})$, parses $\mathtt{tx}^{\mathtt{T}}$ as $(v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{fee}^{\mathtt{T}})$ and does the following:
  1. If $\mathsf{fee}^{\mathtt{T}} < \mathsf{del\text{-}fee}$, then ignore the command. Otherwise, continue.
  2. Sample $\mathsf{id} \xleftarrow{\$} \{0,1\}^\lambda$ and define $m := (([v, \mathsf{fee}], \mathsf{addr}_i^{\mathtt{T}}, [\mathsf{addr}_j^{\mathtt{T}}, \mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}]), \mathsf{id})$.
  3. Compute $\sigma_i^{\mathtt{T}} \xleftarrow{\$} \mathtt{Sign}(\mathsf{sk}_i^{\mathtt{T}}, m)$.
  4. Send $(\mathsf{delegate}, m, \sigma_i^{\mathtt{T}})$ to M.
- M, upon receiving $(\mathsf{delegate}, m, \sigma_i^{\mathtt{T}})$ from P_i, does the following:
  1. Parse m as $(([v, \mathsf{fee}^{\mathtt{T}}], \mathsf{addr}_i^{\mathtt{T}}, [\mathsf{addr}_j^{\mathtt{T}}, \mathsf{addr}_{\mathsf{M}}^{\mathtt{T}}]), \mathsf{id})$.
  2. If $\mathtt{Ver}(\mathsf{addr}_i^{\mathtt{T}}, m, \sigma_i^{\mathtt{T}}) = 0$ or $\mathsf{fee}^{\mathtt{T}} < \mathsf{del\text{-}fee}$, then ignore the message. Otherwise, continue.
  3. Define $\mathtt{tx}_{\mathsf{M}} = (0, \mathsf{addr}_{\mathsf{M}}, 0^\lambda, (m, \sigma_i^{\mathtt{T}}), 3\mathsf{Fee})$.
  4. Send (ACCEPT, P_i) to P_i and $(\text{SUBMIT}, \text{sid}, \mathtt{tx}_{\mathsf{M}})$ to $\mathcal{F}_{\text{T-LEDGER}}$.
- P_i, upon receiving $(\text{SUBMIT}, \text{sid}, \mathtt{tx}^{\mathtt{T}})$, parses $\mathtt{tx}^{\mathtt{T}}$ as $(v, \mathsf{addr}_i, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j, \mathsf{addr}_j^{\mathtt{T}}, \mathsf{fee}, \mathsf{Coin})$ and does the following:
  - If $\mathsf{Coin} = \mathtt{T}$ and $\mathsf{fee} \geq 2\mathsf{Fee}$ then:
    - Sample $\mathsf{id} \xleftarrow{\$} \{0,1\}^\lambda$, define $m := ((v, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addr}_j^{\mathtt{T}}), \mathsf{id})$ and compute $\sigma_i^{\mathtt{T}} \leftarrow \mathtt{Sign}(\mathsf{sk}, m)$.
    - Define $\mathtt{tx} = (0, \mathsf{addr}_i, 0^\lambda, (m, \sigma_i^{\mathtt{T}}), \mathsf{fee})$.
  - If $\mathsf{Coin} = \mathtt{E}$ and $\mathsf{fee} \geq \mathsf{Fee}$ then
    Define $\mathtt{tx} := (v, \mathsf{addr}_i, \mathsf{addr}_j, \perp, \mathsf{fee})$.
  - Send $I = (\text{SUBMIT}, \text{sid}, \mathtt{tx})$ to $\mathcal{F}_{\text{T-LEDGER}}$.

- Upon receiving $(\text{READ}, \text{sid}, \mathsf{type})$, P forwards the command $(\text{READ}, \text{sid})$ to $\mathcal{F}_{\text{T-LEDGER}}$.
- Upon receiving $\mathtt{state}$ from $\mathcal{F}_{\text{T-LEDGER}}$, if $\mathsf{type} = \mathtt{E}$, then P does the following:
  - Initialize an empty list $\mathtt{state}^{\mathtt{E}}$.
  - For each $\mathtt{tx} \in \mathtt{state}$ such that $\mathtt{tx} = (v, \mathsf{addr}_i^{\mathtt{E}}, \mathsf{addr}_j^{\mathtt{E}}, \perp, \mathsf{fee})$, add $\mathtt{tx}$ to $\mathtt{state}^{\mathtt{E}}$.
  - Return $(\text{READ}, \text{sid}, \mathtt{state}^{\mathtt{E}})$.

  Otherwise, P does the following:
  - Initialize the the list $\mathtt{state}^{\mathtt{T}}$ with $(y, 0^\lambda, \mathsf{addr}_0, 0)$ and, for each $\mathtt{tx}^{\mathtt{E}} = (0, \mathsf{addr}_i^{\mathtt{E}}, 0^\lambda, \mathsf{aux}^{\mathtt{E}}, \mathsf{fee})$ in $\mathtt{state}$ where $\mathsf{aux}^{\mathtt{E}} = (\mathsf{v}^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addrs}, \mathsf{id}^{\mathtt{T}}, \sigma_i^{\mathtt{T}})$, do the following:
    - If $\mathsf{checkvalidity}(\mathsf{aux}^{\mathtt{E}}, \mathtt{state}^{\mathtt{T}}) = 1$, then add $(\mathsf{v}^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addrs}, \mathsf{id}^{\mathtt{T}})$ to $\mathtt{state}^{\mathtt{T}}$.
  - Return $(\text{READ}, \text{sid}, \mathtt{state}^{\mathtt{T}})$.

$\mathsf{checkvalidity}(\mathsf{aux}^{\mathtt{E}}, \mathtt{state}^{\mathtt{T}})$

- Parse $\mathsf{aux}^{\mathtt{E}}$ as $(\mathsf{v}^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addrs}, \mathsf{id}_i^{\mathtt{T}}, \sigma_i^{\mathtt{T}})$ and define $m := (\mathsf{v}^{\mathtt{T}}, \mathsf{addr}_i^{\mathtt{T}}, \mathsf{addrs}, \mathsf{id}_i^{\mathtt{T}})$.
- If $\mathtt{Ver}(\mathsf{addr}_i^{\mathtt{T}}, m, \sigma_i^{\mathtt{T}}) = 0$, then return 0. Otherwise, continue.
- Initialize $\mathsf{balance} \leftarrow 0$.
- For each $\mathtt{tx}^\star = (\mathsf{v}^\star, \mathsf{addr}_i^\star, \mathsf{addrs}^\star, \mathsf{id}_i^\star)$ in $\mathtt{state}^{\mathtt{T}}$:
  - If $\mathsf{addr}^\star = \mathsf{addr}_i^{\mathtt{T}}$ and $\mathsf{id}_i^\star = \mathsf{id}_i^{\mathtt{T}}$, then return 0.
  - If $\mathsf{addr}^\star = \mathsf{addr}_i^{\mathtt{T}}$, then compute $\mathsf{balance} \leftarrow -\sum_k \mathsf{v}^\star[k]$.
  - For each k such that $\mathsf{addrs}^\star[k] = \mathsf{addr}_i^{\mathtt{T}}$, compute $\mathsf{balance} \leftarrow +v^\star[k]$.
- If $\sum_k \mathsf{v}^{\mathtt{T}}[k] \geq \mathsf{balance}$, then return 1. Otherwise, return 0.

Fig. 2. Our protocol.

References

1. Ethereum gas station network (GSN) documentation. https://docs.opengsn.org/
2. Al-Balaghi, A.: The state of meta transactions - 2020 (2020). https://medium.com/biconomy/the-state-of-meta-transactions-2020-506840e37e75
3. Andrews, J., Ciampi, M., Zikas, V.: Etherless ethereum tokens: Simulating native tokens in ethereum. Cryptology ePrint Archive, Report 2021/766 (2021). https://ia.cr/2021/766
4. Badertscher, C., Gazi, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: composable proof-of-stake blockchains with dynamic availability. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 913–930. ACM Press (2018). https://doi.org/10.1145/3243734.3243848
5. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: a composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 324–356. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-63688-7_11
6. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001). https://doi.org/10.1109/SFCS.2001.959888
7. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003). https://eprint.iacr.org/2003/239
8. Chakravarty, M.M., Karayannidis, N., Kiayias, A., Jones, M.P., Vinogradova, P.: Babel fees via limited liabilities. arXiv preprint arXiv:2106.01161 (2021)
9. Daian, P., Pass, R., Shi, E.: Snow white: robustly reconfigurable consensus and applications to provably secure proof of stake. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 23–41. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-32101-7_2
10. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
11. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 291–323. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_10
12. Griffith, A.: Ethereum meta transactions (2018). https://medium.com/@austin_48503/ethereum-meta-transactions-90ccf0859e84
13. Kerber, T., Kiayias, A., Kohlweiss, M.: KACHINA - foundations of private smart contracts. In: 34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, 21–25 June 2021, pp. 1–16. IEEE (2021). https://doi.org/10.1109/CSF51468.2021.00002
14. Kerber, T., Kiayias, A., Kohlweiss, M., Zikas, V.: Ouroboros crypsinous: Privacy-preserving proof-of-stake. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, 19–23 May 2019, pp. 157–174. IEEE (2019). https://doi.org/10.1109/SP.2019.00063
15. Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. In: 33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, 22–26 June 2020, pp. 334–349. IEEE (2020). https://doi.org/10.1109/CSF49147.2020.00031, https://doi.org/10.1109/CSF49147.2020.00031

16. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 643–673. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-56614-6_22
17. Pass, R., Shi, E.: The sleepy model of consensus. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 380–409. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-70697-9_14
18. Seres, I.A.: On blockchain metatransactions. In: 2020 IEEE International Conference on Blockchain (Blockchain), pp. 178–187. IEEE (2020)
19. Team, A.: Algorand developer documentation (2021). https://developer.algorand.org/docs/
20. Vogelsteller, F., Buterin, V.: Eip 20: Erc-20 token standard (2015). https://eips.ethereum.org/EIPS/eip-20

A Linear-Time 2-Party Secure Merge Protocol

Brett Hemenway Falk[1], Rohit Nema[2](✉), and Rafail Ostrovsky[2]

[1] University of Pennsylvania, Philadelphia, USA
fbrett@cis.upenn.edu
[2] UCLA, Los Angeles, USA
rnema@ucla.edu, rafail@cs.ucla.edu

Abstract. We present a linear-time, space and communication *data-oblivious* algorithm for securely merging two private, sorted lists into a single sorted, secret-shared list in the *two* party setting. Although merging two sorted lists can be done *insecurely* in linear time, previous *secure* merge algorithms all require super-linear time and communication. A key feature of our construction is a novel method to *obliviously* traverse permuted lists in sorted order. Our algorithm only requires black-box use of the underlying Additively Homomorphic cryptosystem and generic secure computation schemes for comparison and equality testing.

Keywords: Secure computation · Homomorphic encryption · Oblivious protocols

1 Introduction

Securely merging two sorted lists into a single, globally sorted list with the same asymptotic complexity as in the insecure setting has been a long-standing open problem. It is a fundamental tool in many machine learning and data-processing applications [6,42,57], Oblivious RAM [31,45], and Private Set Intersection (PSI) [36]. A series of works [1,13,32,33] have shown that securely *sorting* a list can be done with the same asymptotic complexity as insecure sorting. On the other hand, for *merging*, a gap remains. In the past, it has been solved with complicated techniques that either run in super-linear time or communication, or make unnatural assumptions.

In the insecure setting, and in the three-party ORAM setting, where there are three servers and a *trusted* client, merging two sorted lists of length n can be done in $O(n)$ time, [10], whereas in the *secure* setting, the best existing 2-party secure merge algorithm requires $O(n \log \log n)$ communication [26].

Our main result is to close this gap. More explicitly, we show

Theorem 1 (Main Theorem). *There exists a 2-party protocol for merging two locally sorted lists in linear-time, space and communication that provides*

B. H. Falk, R. Nema and R. Ostrovsky—Work done while consulting for Stealth Software Technologies, Inc.

security against semi-honest adversaries. The protocol only requires black-box use of an Additively-Homomorphic cryptosystem and a generic secure computation protocol for comparison and equality-testing on secret shares.

Secure 2-party merge protocols arise naturally, since the two participants can each sort their list locally before the protocol begins. Three-party protocols for secure merge are less natural, since there are still only two lists being merged, but these lists are secret-shared amongst the *three* computation parties. If the two lists being merged were initially held in the clear by two parties, then it's unnatural to require a third party to aid in the secure merge procedure. On the other hand, if the two lists were initially secret-shared among two parties (e.g. as the output of a previous 3-party computation) it becomes less natural to assume that they are pre-sorted (since they cannot have been sorted locally).

One application of two-party merge protocols is in Private Set Intersection (PSI). There are many PSI protocols, but most output the intersection *in the clear* (e.g. [11,12,16,20–22,24,28,35,37–41,47,48,51–53]). In many applications, however, PSI is only a first step in a larger computation, and in these settings the PSI must return *secret shares* of the intersection, rather than the list itself – but these secret-shared PSI protocols (e.g. [17,49,50]) tend to be less efficient than protocols that reveal the intersection in the clear. One of the earliest methods for secret-shared PSI is the *sort-compare* paradigm [36], where the participants sort their joint list, then compare adjacent elements in a linear pass, deleting singletons. The problem with this approach is that the initial sorting step takes $O(n \log n)$ communication. Using our novel linear-time secure merge protocol, the sort-compare paradigm gives a simple, efficient *linear-communication* secret-shared PSI protocol.

Our protocol is inspired by the 3-server ORAM merge protocol of [10], where the two sorted lists are treated as linked lists, then each linked list is shuffled with a collection of "dummy" elements using a linear-time three-party secure shuffle [43]. Thereafter, the trusted client can traverse the shuffled linked lists, comparing one element at a time, as in the standard insecure merge protocol.

There are several obstacles that need to be overcome in order to eliminate the trusted client and one of the servers from the [10] merge protocol. We can use a linear-time 2-party secure shuffle [26] to replace the 3-party shuffle, but updating the pointers in the shuffled lists is challenging without a trusted client.

To overcome this obstacle, we develop a technique for converting values encrypted under the key of one participant into additive secret shares of the same underlying plaintext (See Sect. 5.2). This conversion process is extremely efficient, and only relies on the cryptosystem being additively homomorphic. Moreover, the trusted client in [10] can easily switch from the real to dummy list obliviously once the real list is exhausted; however, this is non-trivial in our 2-party setup since obviously neither party should learn when a real list has been exhausted. We combat this issue by creating a unique, partially circular linked list (Sect. 5.1 and Fig. 1) such that the protocol can seamlessly switch from the real to dummy list.

Using this novel linked list construction and ciphertext-to-secret-sharing tool, we give a two party secure merge protocol, where each participant treats their input as a linked list, then allows the other participant to shuffle this linked list (while updating the pointers). The parties then re-share these permuted linked lists, and compare elements one at a time (using a secure comparison protocol), while the exact sequence of data accesses from each list is independent of the underlying data. See Sect. 5 for the full construction.

The detailed security proofs and analysis are presented in the full version of the paper [25].

2 Previous Work

2.1 Secure Sorting

Merging two sorted lists can be seen as a special case of sorting, and thus any sorting protocol is also a merge protocol. When security is not required, a simple counting argument shows that any comparison-based sorting algorithm requires $O(n \log n)$ comparisons, whereas two sorted lists can be merged using only $O(n)$ comparisons. Although secure merge protocols are a building block for many secure multiparty computations, most applications focus on the more general (and more difficult) problem of secure sorting.

One route for building a secure sorting protocol is to securely implement a data-oblivious *sorting network* using a generic circuit-based secure multiparty computation (MPC) protocol (e.g. GMW [30], BGW [7] or Garbled Circuits [59,60]). Asymptotically, the best sorting network is the AKS network [1], which requires $O(n \log n)$ comparisons. Although the AKS network is asymptotically optimal, the hidden constants are *extremely* large [2], and so the AKS network has little practical value. In practice, Batcher's bitonic sort [5] which requires $O(n \log^2 n)$ comparisons is much faster and is widely implemented in practice, including in the ABY [23], Obliv-C [61] and EMP [58] compilers. Batcher's sorting network is defined recursively, and thus when using Batcher's network to merge two pre-sorted input lists, all but the final level of the recursion can be omitted. Unfortunately, this does not improve the asymptotic complexity, but it does increase the concrete performance by about a factor of 2.

One problem with implementing traditional sorting algorithms (e.g. quicksort, mergesort, radix sort) using generic secure computation, is that the they are not data-oblivious – even if the comparisons are implemented securely, the *data movement* depends on the underlying values being sorted. The *shuffle-then-sort* paradigm [13,32,33], solves this problem by first *obliviously shuffling* the input lists, then securely executing a traditional sorting algorithm. The initial shuffle ensures that the data movement (which is not hidden by the secure computation) is independent of the underlying data. These techniques yield an asymptotically optimal ($O(n \log n)$) sorting algorithms, that are also efficient in practice.

The efficiency of the shuffle-then-sort paradigm rests on the efficiency of the secure shuffle protocol. In the 3-party setting there are linear-time secure shuffles (based on one-way functions) [43], and in the 2-party there are linear-time secure shuffles (based on additively homomorphic encryption) [29].

Applying the shuffle-then-sort paradigm to the problem of merging immediately yields $O(n \log n)$-communication oblivious merge protocols, but does *not* achieve the $O(n)$-time merging that is possible in the insecure setting. In fact, the $\Omega(n \log n)$ lower bound on comparison-based sorting means that this approach will never yield a linear-time secure *merge* algorithm – unless we can take advantage of the fact that the initial lists being merged are pre-sorted.

Alternative sorting schemes (e.g. Radix sort) avoid the $\Omega(n \log n)$ lower bounds on comparison-based sorting. Another example is [34] in the randomized setting which sorts integers in $O(n\sqrt{\log \log n})$ expected running time. These sorting algorithms are efficient, but rely on the RAM model of computation, and their data-dependent access patterns cannot be efficiently implemented in the circuit model. One exception is [4], which uses non-comparison based techniques to beat the $\Omega(n \log n)$ lower bound, but still remains in the circuit model.

2.2 Secure Merging

Secure, multiparty merge protocols have been studied separately from secure sorting protocols, and just as in the insecure case, focusing on the problem of *merging* allows us to circumvent the $\Omega(n \log n)$ lower bound for sorting.

The first secure merge protocol with (asymptotically) less communication than a corresponding secure sort was given in the 3-server ORAM setting (which requires 3-servers and a trusted client), where there is an information-theoretic secure merge protocol with only $O(n)$ communication [10]. In general, any k-server ORAM protocol, the client can be emulated using secure multiparty computation (MPC), thus the protocol of [10] also yields a 3-server secure merge protocol. Unfortunately, using MPC to securely emulate an ORAM client can dramatically hurt performance since the ORAM client may not be "MPC friendly", e.g. the client may have a very large circuit complexity, which leads inefficiencies when emulating the ORAM client under MPC.

The key idea of [10] is to apply "shuffle-then-sort" [13,32,33] to the idea of merging. Essentially, the participants shuffle the two (sorted) linked-lists – updating the pointers to each element's new, shuffled location. Then the participants apply a standard (non-oblivious) merge protocol to traverse these shuffled linked lists (without needing to hide the data movement). These techniques yield a linear-communication secure merge protocol, but the construction of [10] only works in the 3-party ORAM setting, i.e., when there are four parties, three servers and a trusted client.

The "shuffle-then-merge" paradigm is a bit more delicate than the "shuffle-then-sort" paradigm, since the input lists in a merge are pre-sorted, and the merge protocol must process them in this sorted order (even after the oblivious shuffle). To overcome this difficulty, the pre-sorted input lists can be turned into *linked lists*, and the oblivious shuffle can update each item's pointer to point to the permuted position of its successor [10].

In the two-party setting, [26] gives a protocol based on additively homomorphic encryption that securely merges two lists using $O(n \log \log n)$ communication. The key idea of [26] is that since the input lists are pre-sorted, we can divide the entire list into poly-logarithmic sized blocks, and focus on moving these blocks into (nearly) the correct positions. Once the large blocks are in place, the small number of remaining "strays" that are out of place, can be identified and moved efficiently.

Although our solution is fundamentally different, like [26], we also rely on a linear-time 2-party shuffle.

Our protocol follows a shuffle-then-merge paradigm that is similar to [10], but in order to adapt this to the two-party setting, we create a new protocol for shuffling linked lists in the two-party setting (which can be seen as an extension of the two-party oblivious shuffle of [26,29]).

3 Overview

3.1 Challenges

In the insecure setting, two parties can merge their locally sorted lists by simply comparing their smallest elements and advancing the list with the smaller element. This operation is linear in the length of the two lists. The core issue in translating this linear-time merge algorithm to a secure version is that advancing a list is not *data-oblivious* – it reveals which list contained the smaller element.

Protocol 1. A basic, *data-dependent* merge.

```
Input: Two sorted input lists $A$, $B$ of lengths $n_A$ and $n_B$
Output: A sorted output list $C$ of length $n_A + n_B$
 1: Initialize $i_A = i_B = i_C = 0$
 2: while $i_C < n_A + n_B$ do
 3:     if $A[i_A] < B[i_B]$ or $i_B \geq n_B$ then
 4:         $C[i_c] = A[i_A]$
 5:         $i_A = i_A + 1$
 6:     else
 7:         $C[i_c] = B[i_B]$
 8:         $i_B = i_B + 1$
 9:     end if
10:     $i_C = i_C + 1$
11: end while
```

There are two key challenges when trying to adapt the non-oblivious naïve merge protocol (Protocol 1), into an oblivious variant.

1. **Which list is being accessed**: Whether the algorithm reaches Line 4 or Line 7 reveals which *list* is being accessed.
2. **Which location is being accessed**: When the algorithm reaches Line 4 (resp. Line 7), it reveals which element of A's (resp. B's) list is being accessed at iteration i_C.

We also face an additional challenge: we have only *two* participants in the protocol unlike these prior works which had three, either two servers and a trusted client [44] or three servers and trusted client [10].

3.2 Intuition and Construction Overview

Oblivious Shuffle with Linked List: To address challenge 2, we rely on an oblivious permutation. In the multiparty setting, it is possible to perform efficient (linear-time), oblivious shuffles of secret-shared lists [43]. Similarly, in the two-party scenario, the participants can use additively homomorphic encryption to obliviously shuffle ciphertexts in linear time [26,29]. These linear-time multiparty shuffles are a key building block of many secure multiparty sorting protocols [13,32,33], and secure merge algorithms [10,26].

By viewing each participant's sorted input as a linked list, then shuffling that list, the parties can decouple the locations being accessed from the iteration of the loop – for example, at Line 4 the protocol would read location $\Pi_A(i_A)$ for some random permutation Π_A, instead of directly reading i_A.

There are some subtleties here, as the parties need to obliviously permute their linked lists, and then obliviously traverse them.

In order to allow the parties to traverse the permuted linked lists in the original (sorted) order, the parties must also update the pointers. Thus if π is a permutation of $[n]$, and the original list is $(v[0], \ldots, v[n-1])$, the parties will create two new arrays

$$\begin{aligned} w &= \left(v\left[\pi^{-1}(0)\right], \ldots, v\left[\pi^{-1}(n-1)\right]\right) && \text{Permuted data} \\ t &= \left(\pi\left(\pi^{-1}(0)+1\right), \ldots, \pi\left(\pi^{-1}(n-1)+1\right)\right) && \text{Permuted tags} \end{aligned}$$

With $t[\pi(n-1)] = \perp$. Thus if $w[i] = v[j]$, then $w[t[i]] = v[j+1]$.

This structure allows the parties to traverse the permuted list, w, by first revealing $\pi(0)$ and then, selectively revealing elements of t, starting with $t[\pi(0)]$, $t[\pi(1)]$, ...

Our goal is for each party to achieve a secret-shared, permutation of their own list permuted (as well as the updated pointers) by the other party. In our construction, the second party acts as a *permuting* party for the first and generates both the permuted list and the corresponding linked list to traverse it. To maintain privacy of the data and obliviousness of the memory accesses, the second party's permutation, and the first party's data must remain private.

Now, if the permuting party holds on to its share of the owner party's list, it is not clear how to obliviously traverse the permutation since the permuting party knows the position of each accessed share, and thus each element.

When there are three participants this can be done information-theoretically, by having each participant generate a permutation and secret-share to the other two participants [10]. In the two party setting, we can use additively homomorphic encryption to (obliviously) permute a private list [26,29], but we cannot use those constructions in a black-box manner, since they do not allow us to create the shared tags needed to traverse the permuted list.

Instead, we recombine the shares at the owner party but to maintain obliviousness, i.e. to hide the data itself so as to not leak the permutation, both parties somehow convert their shares into shares encrypted using the *permuting party's* public key. The owner party can then use the additive homomorphism of the encryption scheme to add the encrypted shares and obtain an encryption of the element under the *other* (permuting) party's public key. Therefore, it cannot decrypt to learn the underlying value (and thus, permutation).

Adding Dummies and Oblivious Pointer Advancement: To address challenge 1, we add "dummy" elements to each party's list so that we are able to advance both lists every iteration of the loop. For simplicity, suppose both parties' lists are of size n. Then, both parties can generate n dummy elements and maintain two separate pointers to keep track of the real and dummy elements respectively. These dummies are interspersed with the real elements to create a list of $2n$ elements. At every iteration, the party with the smaller element advances its real pointer, while the other party advances its dummy pointer. This ensures that an element is consumed from both lists every iteration of the merge.

Finally, we are left with two more operations: (1) comparing encrypted real values efficiently and (2) advancing lists obliviously. We achieve (1) using a trick to convert ciphertexts into secret shares which can be passed to any *state-of-the-art* 2-party comparison protocol [18,55] to avoid executing an expensive decryption circuit jointly; and we accomplish (2) by a clever construction of the linked list. The detailed shuffle and merge protocol is shown in Sect. 5.

4 Preliminaries

4.1 Secret Sharing

Our protocol makes use of an additive secret sharing scheme, where a secret $x \in \mathcal{G}$ is shared as $(x - r, r)$, for some random $r \leftarrow \mathcal{G}$ where $\mathcal{G}$ is the finite group that parameterizes the Group Homomorphic Encryption scheme. In the two-party setting all linear secret-sharing schemes are essentially equivalent [19], so we can focus on this scheme without loss of generality.

As is standard, we use the notation $[\![x]\!]$ to denote a secret sharing of the plaintext x. Using the linearity of the secret sharing scheme, the participants can compute $[\![x + y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$ with *no communication*.

For more complex calculations on shares, we rely on secure multiparty computation (MPC), described below.

4.2 Secure Computation

Our protocol makes use of a few simple primitives for processing on secret shares, *comparisons*, *multiplexing* and *equality tests*. These basic primitives are implemented in essentially every secure computation framework, including ABY [23], EMP [58], SCALE-MAMBA [3] and MPyC [54].

We assume that there is an underlying ordering on the elements of $\mathcal{G}$ – this is a necessary assumption since the parties want to *sort* their elements.

Our construction is compatible with both arithmetic and boolean secure computation protocols, although comparisons and equality tests are likely to be more efficient in boolean-circuit-based secure computation protocols.

4.3 Additively Homomorphic Encryption

Our construction makes use of a semantically secure, additively homomorphic cryptosystem, $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$. Our system is compatible with classical

Comparisons

$$[\![x < y]\!] = \begin{cases} [\![0]\!] & \text{if } x \geq y \\ [\![1]\!] & \text{if } x < y \end{cases}$$

Multiplexing

$$\mathsf{mux}\left([\![b]\!], [\![x]\!], [\![y]\!]\right) = \begin{cases} [\![x]\!] & \text{if } b = 0 \\ [\![y]\!] & \text{if } b = 1 \end{cases}$$

Multiplexes are often implemented as a simple multiplication

$$\mathsf{mux}\left([\![b]\!], [\![x]\!], [\![y]\!]\right) = [\![x]\!] + [\![b]\!] \cdot \left([\![y]\!] - [\![x]\!]\right)$$

Equality tests

$$[\![x = y]\!] = \begin{cases} [\![0]\!] & \text{if } x \neq y \\ [\![1]\!] & \text{if } x = y \end{cases}$$

additively homomorphic schemes like Paillier [46], or lattice-based schemes that natively work over $\mathbb{Z}/2\mathbb{Z}$, e.g. BFV [9,27] or CGGI [14,15], both of which are widely supported by current FHE implementations [56]. Note that the security we require for the $\mathsf{Add}(\cdot,\cdot,\cdot)$ is much weaker than full circuit privacy [8], since in our application the summations being computed are known to both parties, and only the *summands* are private.

In order for our final merge protocol to achieve linear communication, the underlying additively homomorphic cryptosystem must have *constant ciphertext expansion.*

4.4 Notation

As there are only two parties, and each party has a unique public key (for the additively homomorphic cryptosystem), when we say "key i" we mean the public key of party i, pk_i.

We denote each party as P_i where $i \in \{0,1\}$. As all our protocols are two-party protocols (and most are completely symmetric), we take all subscripts modulo 2, thus if P_i is one party, P_{i+1} is the other party.

Several protocols below must be run twice, one time for each party, so we give such protocols an index with respect to which we write the steps within the protocol. For example, $\mathsf{Protocol}_i$ will be called twice, for $i \in \{0,1\}$ and we use index i within the protocol to identify the parties. Similarly, we use the same index to define the ideal functionality.

We introduce some more notation in Table 1.

Additively Homomorphic Encryption

Semantic security: for all $x, y \in \mathcal{G}$

$$\left\{ (pk, c_x) : \begin{array}{l} pk, sk \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ c_x \leftarrow \mathsf{Enc}(pk, x) \end{array} \right\} \approx_c \left\{ (pk, c_y) : \begin{array}{l} pk, sk \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ c_y \leftarrow \mathsf{Enc}(pk, y) \end{array} \right\}.$$

Security of Add**:** for all $x, y \in \mathcal{G}$

$$\left\{ \begin{array}{c} c, \\ c_x, c_y, \\ pk, sk \end{array} : \begin{array}{l} pk, sk \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ c_x \leftarrow \mathsf{Enc}(pk, x) \\ c_y \leftarrow \mathsf{Enc}(pk, y) \\ r \leftarrow \mathcal{G} \\ c_r \leftarrow \mathsf{Enc}(pk, r) \\ c \leftarrow \mathsf{Add}(pk, c_x, c_r) \end{array} \right\} \approx_c \left\{ \begin{array}{c} c, \\ c_x, c_y, \\ pk, sk \end{array} : \begin{array}{l} pk, sk \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ c_x \leftarrow \mathsf{Enc}(pk, x) \\ c_y \leftarrow \mathsf{Enc}(pk, y) \\ r \leftarrow \mathcal{G} \\ c_r \leftarrow \mathsf{Enc}(pk, r) \\ c \leftarrow \mathsf{Add}(pk, c_y, c_r) \end{array} \right\}.$$

Decrypting the sum of two ciphertexts yields nothing about the individual summands.

Correctness: for any $x, y \in \mathcal{G}$, and $c > 0$

$$\Pr\left[\left\{ \mathsf{Dec}(sk, c_{x+y}) : \begin{array}{l} pk, sk \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ c_x \leftarrow \mathsf{Enc}(pk, x) \\ c_y \leftarrow \mathsf{Enc}(pk, y) \\ c_{x+y} \leftarrow \mathsf{Add}(pk, c_x, c_y) \end{array} \right\} = x + y \right] > 1 - O\left(\lambda^{-c}\right)$$

Table 1. More notation

$[\![x]\!]$	A secret sharing of the value x
$[\![x]\!]_i$	Party i's secret share of the value x
$\langle\!\langle m \rangle\!\rangle_i$	An encryption of the message m under public key of party i

5 Construction and Protocol Definitions

In this section we describe our construction. First, we present a two-party algorithm for creating and shuffling linked lists. Second, we present a technique for converting encryptions (encrypted by one party) into secret shares. Third, we show how to combine these tools into our main construction which is a linear-communication secure merge protocol.

We assume that party i has a key pair (pk_i, sk_i) for an additively homomorphic cryptosystem $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$.

5.1 Obliviously Shuffling Input Lists

In this section, we describe our novel two-party protocol for padding and shuffling private linked lists. $\mathsf{ShuffleLL}_i$ (Protocol 2). The goal of the $\mathsf{ShuffleLL}_i$ protocol is for party i to achieve a random permutation of its input list with dummies encrypted under party $(i+1)$'s public key. The protocol takes a parameter, m, defining how many "dummy" elements are created. Although $\mathsf{ShuffleLL}_i$ takes m as a parameter, in our final merge protocol, P_1 should set m equal to the length of its input list. The $\mathsf{ShuffleLL}_i$ protocol realizes the ideal functionality, $\mathcal{F}^i_{\text{shuffle}}$ below.

Ideal Functionality $\mathcal{F}^i_{\text{shuffle}}$

1. *Input:* P_i with sorted list v of size n, and P_{i+1} with permutation $\pi \colon [m+n] \to [m+n]$ for some $m > 0$.
2. Create v' by concatenating m dummy elements to the end of v and shuffle v' using π, $w[j] \leftarrow v'\left[\pi^{-1}(j)\right]$ for $j \in \{0, \ldots, n+m-1\}$.
3. Create linked list t to traverse w, such that if $w[j] = v[k]$, then $w[t[i]] = v[k+1]$.
4. For $j \in \{0, \ldots, n+m-1\}$, *output* $\langle\!\langle w[j] \rangle\!\rangle_{i+1}$, and $\langle\!\langle t[j] \rangle\!\rangle_{i+1}$ to P_i, and $\perp$ to P_{i+1}.
5. *Output* $(\llbracket \pi(n+1) \rrbracket_i, \llbracket \pi(0) \rrbracket_i)$ to P_i, and $(\llbracket \pi(n+1) \rrbracket_{i+1}, \llbracket \pi(0) \rrbracket_{i+1})$ to P_{i+1}.
6. *Output* $\llbracket \pi(n) \rrbracket_i$ to P_i and $\llbracket \pi(n) \rrbracket_{i+1}$ to P_{i+1}.

In the second last step, we output a 2-tuple which are secret shares of the head pointers (positions) of the dummy and real list respectively. In the last step, we output the secret share of the position of a special *end-of-list* dummy element. This special element is used to obliviously switch between the real and dummy list. It is explained in detail in Sect. 5.1 and 5.3.

Below, we describe the shuffle for party P_0 but in the final protocol they also swap positions and rerun. Assume that P_0 holds a sorted list v of length n, and P_1 generates a random permutation π over $[m+n]$. Then, the protocol proceeds as follows,

1. *Encrypt sorted list:* To hide its real elements (input list), P_0 encrypts each element using its public key pk_0 and sends the list of ciphertexts (in sorted order of the underlying value) to P_1.
2. *Generate shares:* Given a value v', party 1 can create an additive sharing of v' as $(v'-r, r)$ for some random value $r \in \mathcal{G}$. In our setting, however, P_1 does not have the plaintext value, v', but instead has an encryption $\langle\!\langle v' \rangle\!\rangle_0$. Using the additively homomorphism, given a ciphertext $\langle\!\langle v' \rangle\!\rangle_0$, party 1 creates the encrypted pair $(\langle\!\langle v'-r \rangle\!\rangle_0, \langle\!\langle r \rangle\!\rangle_1)$. See Line 2.
3. *Concatenate encrypted dummies:* Party P_1 creates a special dummy known as the *end-of-list* element, and $m-1$ random dummy elements. The *end-of-list* element marks the end of both the real and dummy list but also points to the first element of the dummy list. Therefore, the *end-of-list* element along with the dummy elements form a cycle. The *end-of-list* element stores the largest real value in sorted order instead of a random number as its value. P_1 easily constructs the *end-of-list* element encrypted under pk_0 by

just duplicating $\langle\!\langle v[n-1] \rangle\!\rangle_0$. Instead of a linked list terminating by pointing to $\perp$, we will have it point to the this *end-of-list* element. The purpose of the special element becomes apparent when either party's real list is exhausted and we must obliviously *switch* to traversing the dummy list while we access the remaining real elements from the other party (See Sect. 5.3).

4. *Permute ciphertexts and create linked list:* Party P_1 permutes the pair of shares using π by assigning the k^{th} element of the permuted list to the $\pi^{-1}(k)^{\text{th}}$ element of the concatenated list as shown in Line 5. To traverse the permuted list in sorted order, P_1 also generates a linked list such that the i^{th} element is the position of the *next* element in sorted order (see Line 6). We also point the *last* dummy element to the *end-of-list* element. Therefore, the real (resp. dummy) list reaches the *end-of-list* element after n (resp. m) steps. See Fig. 1 below which illustrates this construction.
   To hide the linked list from party P_0 (and thus, the underlying permutation), P_1 encrypts each element of the linked list using its public key, pk_1. Finally, it secret shares the position of the first dummy and the first real element as a 2-tuple *head pointer*, *and* the *end-of-list* element.
5. *Recombine shares:* P_1 sends both the shuffled ciphertext pairs and the encrypted linked list to P_0. Party P_0 first decrypts the ciphertexts which were encrypted under its own public key, pk_0 and then *re-encrypts* them using pk_1, P_1's public key. Using the additive property of the encryption scheme, P_0 adds the newly obtained ciphertexts to their corresponding ciphertexts in the pair. Due to the homomorphic property, P_0 obtains an encryption of the sum of the underlying value which is in fact, the original set of real/dummy elements as the pairs were constructed precisely from those values.

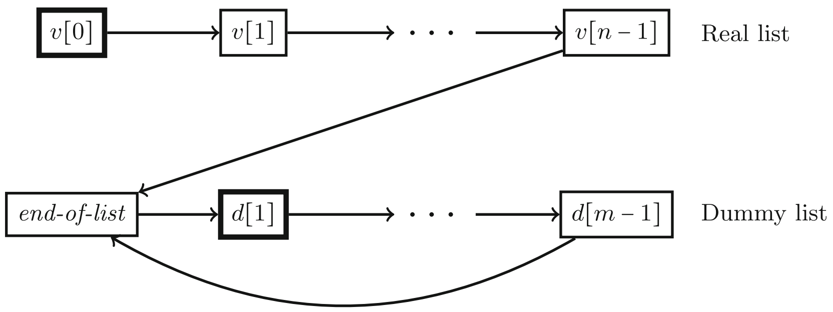


Fig. 1. Construction of the linked list. $d[1]$ and $v[0]$ (as pair of encrypted shares) are the at the head of the dummy and real pointer respectively. Both the last real element, $v[n-1]$ and last dummy element, $d[m-1]$ point to the *end-of-list* element.

Therefore, at the end of Protocol 2, P_0 obtains a permutation (oblivious to itself) of its original list with dummies encrypted under P_1's public key, along with an encrypted linked list to traverse it. Note that the *end-of-list* element is treated as a *dummy* element but stores a *real* value which is crucial in proceeding obliviously when either party exhausts its real list. We further elaborate on this in Sect. 5.3.

We prove $\mathsf{ShuffleLL}_i$ securely computes the ideal functionality $\mathcal{F}^i_{\text{shuffle}}$ in [25].

Protocol 2. $\mathsf{ShuffleLL}_i$: Pad and Permute Linked Lists

Input: Party P_i holds sorted list v of size n; P_{i+1} holds random permutation $\pi\colon [m+n] \to [m+n]$ for some $m > 0$.

Output: P_i obtains a permutation (under π) of its elements (with m dummies) and linked list, both encrypted using P_{i+1}'s public key.

```
    (index $j \in \{0, \ldots, n+m-1\}$)
 1: For $k \in \{0, \ldots, n-1\}$, $P_i$ encrypts $c[k] \leftarrow \langle\!\langle v[k] \rangle\!\rangle_i$, and sends $c$ to $P_{i+1}$
 2: For $k \in \{0, \ldots, n-1\}$, $P_{i+1}$ generates random value $r_k \leftarrow \mathcal{G}$, and creates
    $c_i[k] \leftarrow (c[k] - \langle\!\langle r_k \rangle\!\rangle_i, \langle\!\langle r_k \rangle\!\rangle_{i+1})$    ▷ 2-tuples of the form $(c_i[k][0], c_i[k][1])$
 3: $P_{i+1}$ generates random $r \leftarrow \mathcal{G}$ and sets $c'_i[0] \leftarrow (c[n-1] - \langle\!\langle r \rangle\!\rangle_i, \langle\!\langle r \rangle\!\rangle_{i+1})$
                     ▷ end-of-list element. $c[n-1] - \langle\!\langle r \rangle\!\rangle_i = \langle\!\langle v[n-1] - r \rangle\!\rangle_i$
 4: For $k \in \{1, \ldots, m-1\}$, $P_{i+1}$ generates dummies, $d[k] = d_0[k] + d_1[k]$ where
    $d_0[k], d_1[k] \leftarrow \mathcal{G}$ are random, and creates, $c'_i[k] \leftarrow (\langle\!\langle d_i[k] \rangle\!\rangle_i, \langle\!\langle d_{i+1}[k] \rangle\!\rangle_{i+1})$
 5: $P_{i+1}$ permutes, $c_i^\pi[j] \leftarrow (c_i \| c'_i)\left[\pi^{-1}(j)\right]$
 6: $P_{i+1}$ creates linked list, $t'[\pi(j)] \leftarrow \pi(j+1)$ with $t'[\pi(n+m-1)] = \pi(n)$    ▷ point
    the last dummy to the end-of-list element
 7: $P_{i+1}$ encrypts $t_i[j] \leftarrow \langle\!\langle t'[j] \rangle\!\rangle_{i+1}$
 8: $P_{i+1}$ secret shares $\mathfrak{p}_i = (\pi(n+1), \pi(0))$    ▷ head pointers tuple
 9: $P_{i+1}$ secret shares $\mathfrak{e}_i = \pi(n)$    ▷ end-of-list element
10: $P_{i+1}$ sends $c_i^\pi$, and $t_i$ to $P_i$
11: $P_i$ recombines $c^\pi[j] \leftarrow c_i^\pi[j][1] + \langle\!\langle \mathsf{Dec}(sk_0, c_i^\pi[j][0]) \rangle\!\rangle_{i+1}$
```

5.2 Converting Ciphertexts to Secret Shares

In this section, we give an efficient 2-party protocol for converting ciphertexts from an additively homomorphic cryptosystem into secret shares of the same underlying value. A similar idea was used implicitly for creating "blinded permutations" [29].

In principle, a general-purpose MPC protocol can always be used to convert ciphertexts to secret shares by evaluating the decryption circuit for the encryption scheme within the MPC, but, in general, this is extremely inefficient. $\mathsf{EncToSS}_i$ (Protocol 3) gives an extremely efficient two-party protocol for achieving the same result when the underlying cryptosystem is additively homomorphic. $\mathsf{EncToSS}_i$ realizes the ideal functionality, $\mathcal{F}^i_{\text{decrypt}}$ defined below.

Ideal Functionality $\mathcal{F}^i_{\text{decrypt}}$

1. *Input:* P_i with ciphertext, $\langle\!\langle v \rangle\!\rangle_{i+1}$.
2. *Output* secret shares of value v: $[\![v]\!]_i$ to P_i, and $[\![v]\!]_{i+1}$ to P_{i+1}.

In our setting, party i holds a ciphertext $c = \langle\!\langle v \rangle\!\rangle_{i+1}$ of a private value, v, encrypted under party $(i+1)$'s key. At the end of the protocol, the parties hold additive secret shares of the underlying value v, and neither party learns anything about v.

We prove that $\mathsf{EncToSS}_i$ securely computes $\mathcal{F}^i_{\text{decrypt}}$ in [25].

Protocol 3. $\mathsf{EncToSS}_i$: Convert Ciphertext to Secret Share

Input: Party P_i inputs ciphertext, $c = \langle\!\langle v \rangle\!\rangle_{i+1}$ (encrypted using pk_{i+1}).
Output: Returns secret sharing of the underlying plaintext, v.

```
1: $P_i$ generates random value, $r_i \leftarrow \mathcal{G}$
2: $P_i$ encrypts $\langle\!\langle r_i \rangle\!\rangle_{i+1}$
3: $P_i$ uses the additive homomorphism to compute $\langle\!\langle v + r_i \rangle\!\rangle_{i+1}$
4: $P_i$ sends $c' = \langle\!\langle v + r_i \rangle\!\rangle_{i+1}$ to $P_{i+1}$
5: $P_{i+1}$ decrypts $v' \leftarrow \mathsf{Dec}(sk_{i+1}, c')$
6: $P_{i+1}$ shares $v'$
7: $P_i$ sets $[\![v'']\!]_i = [\![v']\!]_i - r_i$
8: return $[\![v'']\!]$
```

5.3 Securely Merging Obliviously Shuffled Lists

We are finally ready to *securely* merge the two parties' lists. Our Merge protocol realizes the ideal functionality, $\mathcal{F}_{\text{merge}}$ defined below.

Ideal Functionality $\mathcal{F}_{\text{merge}}$

1. *Input:* For $i \in \{0, 1\}$, P_i with list v_i of size n_i.
2. $\mathcal{F}_{\text{merge}}$ merges the two lists v_1 and v_2 such that the resultant list, v is sorted.
3. *Output* secret shares of each element of v, $[\![v\,[j]]\!]_0$ to P_0, and $[\![v\,[j]]\!]_1$ to P_1, for $j \in \{0, \ldots, n_0 + n_1\}$.

Suppose party P_i holds list v_i of size n_i. The protocol proceeds as described below.

1. *Obliviously shuffle padded list with linked list:* First, both parties call $\mathsf{ShuffleLL}_i$ (for $i \in \{0, 1\}$ (as described in Protocol 2) to obtain an encrypted, permuted version of their input list padded with dummies (including the *end-of-list* element). $\mathsf{ShuffleLL}_i$ also outputs an encrypted linked list that party i

later uses to traverse their list without leaking the accessed positions to party $i + 1$ (who knows the permutation).

2. *Access elements from shuffled list:* The parties maintain a secret-shared bit for each party, $[\![b_i]\!]$, and $b_i = 1$ at iterations where P_i needs to access a real element, and $b_i = 0$ at iterations where P_i needs to access a dummy element. In the first step, both parties access their first real element, in all subsequent steps $b_0 \neq b_1$ since only one party advances its real list.[1] The bit, b_i, allows the parties to select and update the appropriate values obliviously using the mux operation (e.g. Protocol 5 line 9).

   At every step in the protocol, the parties also maintain a secret sharing of the last observed real value in P_i's list, cur_i. In any iteration where a dummy element must be consumed from party i's list, we use b_i to obliviously select cur_i over the dummy value, effectively discarding it in place of the actual real value to be compared. See Line 14 of Protocol 5.
3. *Compare real values:* Using b_i, we obtain the real values at the head of each real list. To find the smaller element, we use a generic comparison protocol (Sect. 4.2) which returns a (secret-shared) bit equal to 1 if party 0's real value was smaller than party 1's. Therefore, we set b_0 to the result of the comparison protocol (line 15) and $b_1 \leftarrow 1 - b_0$ (line 16) allowing us to appropriately update the head pointer for the next step.
4. *Update head pointer:* Now, we advance one party's real list and the other party's dummy list as follows. First, we find the next position from the encrypted linked list using $\mathsf{EncToSS}_i$. Then, we update the appropriate entry of the head pointer using bit, b_i (line 1). If $b_i = 1$, then this means that P_i's real value was smaller and we must advance the real (resp. dummy) pointer to obtain the next real (resp. dummy) value from P_i's (resp. P_{i+1}'s) list. Protocol 4 details how the head pointer is advanced. We prove in [25] that that every memory location in the shuffled list is accessed exactly once, which makes the overall access pattern independent of the underlying data.
5. *Switching from an exhausted list:* When either party exhausts their real list, we must somehow *notify* the protocol and secret-share the remaining values of the other real list.

   We keep track of when a real list is exhausted by checking when the real pointer reaches the *end-of-list* element. We do so securely using a generic equality testing MPC protocol as described in Sect. 4.2. We maintain another secret-shared bit, fin initialized to 0, which acts like a boolean flag and is inverted as soon as either real pointer reaches its corresponding *end-of-list* element. See line 10 of Protocol 5.

   Without loss of generality, suppose that party 0 exhausted its real list first. This implies that $b_0 = 1$ (and $b_1 = 0$) from the previous iteration, and the real pointer has been advanced to store the position of the *end-of-list* element. Recall that the underlying value of the *end-of-list* element is exactly the same as the largest real value, i.e., the most recent element that party 0 accessed in

[1] Since $b_0 = \neg b_1$ at every iteration after the first, we could increase efficiency by storing only a single bit, but the exposition is simpler if we forego this minor optimization.

the previous iteration. So on Line 14, val_0 will equal the *end-of-list* element i.e., the largest real value of party 0, and val_1 will equal cur_1, the most recent real value from party 1 that has not been advanced and secret-shared yet. Therefore, essentially, we will perform the same comparison as the previous iteration and conclude that val_0 is smaller. However, val_0 is a duplicate of the most recent real value that was secret-shared in the previous iteration. This is where we use the fin bit to "reverse" the bits so that we instead select val_1 as the next real value, and advance the real pointer of party 1 (and dummy pointer of party 0) as required since we're only left with real values from party 1's list. As val_0 is smaller than every remaining real value in party 1's list, every comparison hereafter will always return $b_0 = 1$ which we always *invert* hereafter using fin. We prove fin remains 1 once set in [25], thus proving the correctness of the algorithm. In summary, performing these *dummy* comparisons allows the protocol to remain oblivious by still accessing elements from the permuted list, and using the fin bit allows the protocol to *correctly* compute the merge.

Lastly, notice that if party 1 exhausts it real list first, then by construction, party 0's dummy pointer will reach the *end-of-list* element as we consume one dummy for each real element after the first one and thus, cycle back from the last dummy element to the *end-of-list* element. And since party 1 just exhausted its real list, we know $b_0 = 0$ and $b_1 = 1$. So, pos_0 is equal to the position of the dummy pointer, i.e., the position of the *end-of-list* element. Therefore, in either case (whether party 0 or 1 exhausts a real list), pos_0 will always equal the position of the *end-of-list* element and it is sufficient to only test pos_0 for setting fin (line 10).

Protocol 4. $\mathsf{UpdateHead}_i$: Update Head Pointer to Linked Lists

Input: Bit, $[\![b]\!]$; Head pointer tuple, $[\![\mathfrak{p}]\!]$; linked list, t held by party P_i.
Output: Head pointer tuple updated with the next real or dummy position from t according to bit, b.

```
1: $[\![pos]\!] \leftarrow \mathsf{mux}([\![b]\!], [\![\mathfrak{p}[0]]\!], [\![\mathfrak{p}[1]]\!])$
2: $\mathsf{Reveal}_i(pos)$                ▷ The revealed $pos$ is an index in the shuffled list
3: $[\![next]\!] \leftarrow \mathsf{EncToSS}_i(t[pos])$
4: $[\![\mathfrak{p}_{new}[1]]\!] \leftarrow \mathsf{mux}([\![b]\!], [\![\mathfrak{p}[1]]\!], [\![next]\!])$
5: $[\![\mathfrak{p}_{new}[0]]\!] \leftarrow \mathsf{mux}([\![b]\!], [\![next]\!], [\![\mathfrak{p}[0]]\!])$
6: return $[\![\mathfrak{p}_{new}]\!]$
```

In the end, both parties obtain element-wise secret shares of the merge of their two sorted lists such that the resulting list is also in sorted order. We prove Merge securely computes $\mathcal{F}_{\text{merge}}$ in [25].

Our algorithm runs in time linear in the length of the two lists requires only linear communication between the two parties assuming the underlying encryption scheme produces ciphertexts with constant factor expansion. The concrete costs are outlined in [25].

Protocol 5. Merge: Securely Merge Sorted Lists

Input: Party P_i holds input list v_i of size n_i.

Output: Parties obtain a secret sharing of the merge of the lists in sorted order.

```
1: For $i \in \{0,1\}$, $P_i$ locally generates random permutation, $\pi_i : [n_0 + n_1] \to [n_0 + n_1]$.
2: For $i \in \{0,1\}$, run $\mathsf{ShuffleLL}_i(v_i, \pi_{i+1})$ so that $P_i$ obtains ciphertext list, $c_i$, linked
   list, $t_i$ and secret shares, $[\![\mathfrak{p}_j]\!]_i$ and $[\![\mathfrak{e}_j]\!]_i$ for $j \in (0,1)$.
3: For $i \in \{0,1\}$, $[\![b_i]\!] \leftarrow [\![1]\!]$    ▷ $b_i$ indicates real or dummy list
4: For $i \in \{0,1\}$, $[\![cur_i]\!] \leftarrow [\![\perp]\!]$    ▷ $cur_i$ is the current value in the *real list*
5: $[\![end]\!] \leftarrow [\![\mathfrak{e}_0]\!]$    ▷ position of the *end-of-list* element
6: $[\![fin]\!] \leftarrow [\![0]\!]$    ▷ $fin = 1$ if either real list is exhausted
7: $k \leftarrow 0$
8: **while** $k < n_0 + n_1$ **do**
9:     For $i \in \{0,1\}$, $[\![pos_i]\!] \leftarrow \mathsf{mux}([\![b_i]\!], [\![\mathfrak{p}_i[0]]\!], [\![\mathfrak{p}_i[1]]\!])$    ▷ Choose $pos_i$ based on $b_i$
10:    $[\![fin]\!] \leftarrow [\![fin]\!] \oplus [\![pos_0 = end]\!]$    ▷ If $fin = 1$ it will remain 1
11:    For $i \in \{0,1\}$, $\mathsf{Reveal}_i([\![pos_i]\!])$
12:    For $i \in \{0,1\}$, $[\![\mathfrak{p}_i]\!] \leftarrow \mathsf{UpdateHead}_i([\![b_i]\!], [\![\mathfrak{p}_i]\!], t_i)$    ▷ Move to new head
13:    For $i \in \{0,1\}$, $[\![temp_i]\!] \leftarrow \mathsf{EncToSS}_i(c_i[pos_i])$    ▷ Access next position
14:    For $i \in \{0,1\}$, $[\![val_i]\!] \leftarrow \mathsf{mux}([\![b_i]\!], [\![cur_i]\!], [\![temp_i]\!])$    ▷ Choose real values
15:    $[\![b_0]\!] \leftarrow [\![val_0 < val_1]\!] \oplus [\![fin]\!]$    ▷ Compare real values
16:    $[\![b_1]\!] \leftarrow [\![1 - b_0]\!]$
17:    $[\![l[k]]\!] \leftarrow \mathsf{mux}([\![b_0]\!], [\![val_1]\!], [\![val_0]\!])$    ▷ $l[k]$ is the smaller value
18:    For $i \in \{0,1\}$, $[\![cur_i]\!] \leftarrow [\![val_i]\!]$    ▷ Store most recent real value
19:    $k \leftarrow k + 1$
20: **end while**
21: **return** $([\![l[0]]\!], \ldots, [\![l[n_0 + n_1 - 1]]\!])$    ▷ secret-sharing of sorted merged list
```

6 Conclusion

In this paper, we presented the first linear-communication 2-party secure merge protocol. The protocol is asymptotically optimal, and efficient enough for practical applications. To achieve this protocol, we introduced a 2-party method to obliviously traverse a permuted list using a novel linked list construction and an extremely efficient technique to convert ciphertexts to secret shares.

Our secure merge protocol makes only black-box use of an additively homomorphic cryptosystem, and a secure computation protocol supporting comparisons, equality tests, and multiplexing on secret shared values.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: Sorting in $c\log(n)$ steps. Combinatorica **3**, 1–19 (1983)
2. Al-Haj Baddar, S., Batcher, K.: The AKS sorting network. In: Designing Sorting Networks: A New Paradigm, pp. 73–80. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-1851-1_11
3. Aly, A., Keller, M., Rotaru, D., Scholl, P., Smart, N.P., Wood, T.: SCALE-MAMBA (2019). https://homes.esat.kuleuven.be/~nsmart/SCALE/

4. Asharov, G., Lin, W., Shi, E.: Sorting short keys in circuits of size o(n log n). In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, 10–13 January 2021. pp. 2249–2268. SIAM (2021)
5. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference, pp. 307–314. ACM (1968)
6. Bater, J., Elliott, G., Eggen, C., Goel, S., Kho, A., Rogers, J.: SMCQL: secure querying for federated databases. Proc. VLDB Endow. **10**(6), 673–684 (2017)
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, pp. 1–10. ACM, New York (1988)
8. Bourse, F., Del Pino, R., Minelli, M., Wee, H.: FHE circuit privacy almost for free. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 62–89. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_3
9. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50
10. Chan, T.-H.H., Katz, J., Nayak, K., Polychroniadou, A., Shi, E.: More is less: perfectly secure oblivious algorithms in the multi-server setting. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 158–188. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_7
11. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: CCS, pp. 1223–1237. ACM (2018)
12. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS, pp. 1243–1255 (2017)
13. Chida, K., Hamada, K., Ikarashi, D., Kikuchi, R., Kiribuchi, N., Pinkas, B.: An efficient secure three-party sorting protocol with an honest majority. IACR ePrint 2019/695 (2019)
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
15. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 377–408. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_14
16. Chongchitmate, W., Ishai, Y., Lu, S., Ostrovsky, R.: PSI from ring-OLE. In: CCS 2022. ACM (2022)
17. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 464–482. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_25
18. Couteau, G.: New protocols for secure equality test and comparison. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 303–320. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93387-0_16
19. 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

20. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_8
21. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_13
22. De Cristofaro, E., Tsudik, G.: Experimenting with fast private set intersection. In: Katzenbeisser, S., Weippl, E., Camp, L.J., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust 2012. LNCS, vol. 7344, pp. 55–73. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30921-2_4
23. Demmler, D., Schneider, T., Zohner, M.: ABY-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
24. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: CCS, pp. 789–800 (2013)
25. Falk, B.H., Nema, R., Ostrovsky, R.: A linear-time 2-party secure merge protocol. Cryptology ePrint Archive, Report 2022/380 (2022)
26. Falk, B.H., Ostrovsky, R.: Secure merge with $o(nloglogn)$ secure operations. In: 2nd Conference on Information-Theoretic Cryptography (ITC 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)
27. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR ePrint 2012/144 (2012)
28. 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
29. Gentry, C., Halevi, S., Jutla, C., Raykova, M.: Private database access with HE-over-ORAM architecture. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 172–191. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_9
30. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC, pp. 218–229 (1987)
31. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM (JACM) **43**(3), 431–473 (1996)
32. Hamada, K., Ikarashi, D., Chida, K., Takahashi, K.: Oblivious radix sort: an efficient sorting algorithm for practical secure multi-party computation. IACR ePrint 2014/121 (2014)
33. Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically efficient multi-party sorting protocols from comparison sort algorithms. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 202–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_15
34. Han, Y., Thorup, M.: Integer sorting in 0(n sqrt (log log n)) expected time and linear space. In: Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS 2002, pp. 135–144. IEEE Computer Society (2002)
35. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. J. Cryptol. **23**(3), 422–456 (2010)
36. Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: NDSS (2012)

37. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_34
38. 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
39. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. PoPETs **4**, 97–117 (2017)
40. 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
41. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS, pp. 818–829 (2016)
42. Laud, P., Pankova, A.: Privacy-preserving record linkage in large databases using secure multiparty computation. BMC Med. Genom. **11**(4), 84 (2018)
43. Laur, S., Willemson, J., Zhang, B.: Round-efficient oblivious database manipulation. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 262–277. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24861-0_18
44. Lu, S., Ostrovsky, R.: Distributed oblivious RAM for secure two-party computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 377–396. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_22
45. Ostrovsky, R.: Efficient computation on oblivious RAMs. In: STOC, pp. 514–523 (1990)
46. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
47. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 401–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_13
48. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: USENIX Security Symposium, pp. 515–530 (2015)
49. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 122–153. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_5
50. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 125–157. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_5
51. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX, pp. 797–812 (2014)
52. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. IACR Cryptology ePrint Archive (2016)
53. Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 235–259. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_9
54. Schoenmakers, B.: MPyC: secure multiparty computation in Python. Github, February 2019

55. Veugen, T., Blom, F., de Hoogh, S.J., Erkin, Z.: Secure comparison protocols in the semi-honest model. IEEE J. Sel. Top. Signal Process. **9**(7), 1217–1228 (2015)
56. Viand, A., Jattke, P., Hithnawi, A.: SoK: fully homomorphic encryption compilers. arXiv preprint arXiv:2101.07078 (2021)
57. Volgushev, N., Schwarzkopf, M., Getchell, B., Varia, M., Lapets, A., Bestavros, A.: Conclave: secure multi-party computation on big data. In: EuroSys, p. 3. ACM (2019)
58. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: efficient multiparty computation toolkit (2016). https://github.com/emp-toolkit/emp-sh2pc
59. Yao, A.: Protocols for secure computations (extended abstract). In: FOCS 1982, pp. 160–164 (1982)
60. Yao, A.: How to generate and exchange secrets. In: FOCS 1986, pp. 162–167 (1986)
61. Zahur, S., Evans, D.: Obliv-C: a language for extensible data-oblivious computation. IACR Cryptology ePrint Archive 2015/1153 (2015)

FairMM: A Fast and Frontrunning-Resistant Crypto Market-Maker

Michele Ciampi[1], Muhammad Ishaq[2(✉)], Malik Magdon-Ismail[3], Rafail Ostrovsky[4], and Vassilis Zikas[2]

[1] The University of Edinburgh, Edinburgh, UK
michele.ciampi@ed.ac.uk
[2] Purdue University, West Lafayette, USA
{ishaqm,vzikas}@cs.purdue.edu
[3] Rensselaer Polytechnic Institute (RPI), Troy, USA
[4] University of California, Los Angeles (UCLA), Los Angeles, USA
rafail@cs.ucla.edu

Abstract. Frontrunning is a major problem in DeFi applications, such as blockchain-based exchanges. Albeit, existing solutions are not practical and/or they make external trust assumptions. In this work we propose a market-maker-based crypto-token exchange, which is both more efficient than existing solutions and offers provable resistance to frontrunning attack. Our approach combines in a clever way a game theoretic analysis of market-makers with new cryptography and blockchain tools to defend against all three ways by which an exchange might front-run, i.e., (1) reorder trade requests, (2) adaptively drop trade requests, and (3) adaptively insert (its own) trade requests. Concretely, we propose novel light-weight cryptographic tools and smart-contract-enforced incentives to eliminate reordering attacks and ensure that dropping requests have to be oblivious (uninformed) of the actual trade. We then prove that with these attacks eliminated, a so-called monopolistic market-maker has no longer incentives to add or drop trades. We have implemented and benchmarked our exchange and provide concrete evidence of its advantages over existing solutions.

Keywords: Front-running · Market maker · Blockchain · Fairness

1 Introduction

Since Bitcoin's introduction in 2008, *crypto-currencies* have become a significant market in global finance[1]. Several tools and platforms have been built to facilitate crypto-currency trading. The early exchanges e.g. Binance, Coinbase, Bittrex, etc. have two undesirable properties. First, they were *custodial*, meaning that traders transfer their assets to the exchange, and trading activity translates

[1] Market capitalization of approx. $2 trillion during all of 2021.

to updating the exchange's internal mapping of traders and their assets. Second, they were *order-book* based i.e. they only match buyers with sellers (collectively, traders), so trading halts when there are no sellers whose ask-prices match the buyer bid-prices. The prices are fully controlled by traders and therefore can be volatile. The later exchanges e.g. Uniswap, Balancer, Curve, etc.—collectively called Decentralized Exchanges or *DEXes*—are *non-custodial*, have their own inventory of assets, and use a market making (MM) algorithm to adjust prices. The latter type of exchanges, colloquially also known as *market makers*, leverage machine learning (ML) to increase liquidity along with additional desirable properties for the market maker (e.g., maximizing profit) or the market itself (e.g., stability by incentivizing traders to report true valuations).

A perennial problem with both types of exchanges (and traditional markets, too) is *frontrunning*, where an adversary reorders trades to gain a price advantage. For example, say a market maker (MM) is selling an asset t at \$100 each, and the pricing function is such that for each unit sold, the price increases by \$100. Say a trader P submits a trade T_P to buy one unit, a (possibly adversarial) MM A sees this pending trade and executes, before processing T_P, a different trade T_A to buy one unit, i.e. the adversary A "front-runs" the trader P. Executing T_A before T_P raises the price to \$200 for P, \$100 more than he would otherwise pay. Trades should be executed in the order they were submitted.

Frontrunning penalizes honest players, and also has a detrimental effect on the health of the underlying crypto-currency blockchain. As an example, adversarial bots might flood the blockchain with frontrunning orders when an opportunity for profit arises, with only few of these orders being executed. This flooding creates a denial of service (DoS) attack. The above effects of frontrunning are worsened in decentralized exchanges. E.g., when the exchange is implemented by a smart contract on a chain like Ethereum, e.g., Uniswap, frontrunning raises transaction fees (*gas price* on Ethereum) as traders compete to get their trades in first. And, in theory, it can also affect the security of the underlying blockchain. Indeed, in DEXes, where the trade ordering is affected by miners through transaction reordering, frontrunning might create incentives for forking as the miners are more likely to pursue a chain on which they make more profit.

Traditional markets mitigate frontrunning through legislation. However, such legislation is tricky to enforce as most frontrunning attacks do not leave an indisputable evidence, which would be necessary to apply fines. As such, several mature markets have embedded certain controlled forms of frontrunning in their allowed operations. The classical example of this principle is embodied in traditional stock exchange markets, where high-profile clients might get so called *privileged access*, allowing them to react faster to market changes. The above solution is unsatisfactory for crypto-currency markets. For starters, legislation lags behind and is typically not tailored to crypto-currencies, making deterrence by regulation much harder. But more importantly, the egalitarian nature of these assets makes preferential frontrunning an undesirable feature for the majority of its users. The main question addressed (to the affirmative) by our work is

Can we leverage the public-observability of blockchain ledgers together with cryptographic and machine learning tools to devise a practical frontrunning resistant market maker?

Informally, solving the above problem requires: (i) ensuring a strict ordering on the trades, and (ii) making sure the system is fast enough for high-frequency trading. The first requirement gets rid of frontrunning attack and the second ensures the solution's practicality. The first requirement can be further broken down into a) preventing traders from frontrunning each other and b) preventing MM from frontrunning. For private-state blockchains (e.g. ZCash, Dash, Monero, etc.), preventing traders from frontrunning is easy because the traders cannot see each other's trades (due to transaction privacy). However, the mainstream blockchains (e.g. Bitcoin, Ethereum, Cardano, etc.) are not private and preventing traders from frontrunning each other requires additional mechanisms.

Several such mechanisms have been proposed, e.g. commit-reveal, encryption, zk-rollups, speed-bumps/retro-active pricing, and commit secret sharing. However they all cause a slowdown of the system and are, therefore, in conflict with the second requirement (the system needs to be fast). Alternative approaches have proposed to simulate MM as a trusted third party (TTP) through secure multi-party computation (MPC), trusted execution environments (TEEs) and zero-knowledge proofs (ZKPs). These do solve the frontrunning problem, but once more, conflict with the second requirement and/or make additional trust assumptions (see also the related research section for a detailed comparison). Indeed, MPC and ZKPs are expensive cryptographic primitives that negatively affect speed of the system and, TEEs place additional demands on application hardware (and assumptions thereof). A viable solution needs to satisfy both of the above requirements, ideally without additional trust assumptions.

This work, FairMM, proposes a market maker (MM) that resolves the frontrunning problem using off-chain communication and inexpensive hash functions. This is done through a combination of techniques from cryptography, game theory, blockchain, and machine leaning for financial economics.

At a high level, FairMM operates as follows: Traders and MM communicate off-chain via secure communication channels (e.g. through TLS), traders form a queue, and a trade is processed as follows: 1) MM issues a ticket to the trader T_i at the front of the queue, this ticket is identified by a cryptographic hash and signed by the MM (think of the hash as a serial number), then, 2) trader T_i may decide to respond with trade (trade) or not trade (no_trade), 3) MM processes T_i's response and moves to the next trader T_{i+1}. The nature of this ticket hash (or serial number) is such that it incorporates the complete trading history up to that point. If the MM tries to talk to more than one trader at a time, all but one of the traders will get their trade incorporated into the trading history (trading history is linear, there is no way to keep all serial numbers valid without creating branches). MM posts the trading activity to a public bulletin board at regular intervals. The traders can read this bulletin board at any time and if they find any invalid tickets or that their ticket is missing, they can complain to a smart contract. This smart contract, established by the MM before its operation starts, locks a large collateral on behalf of MM. On a valid complaint, the complainant is rewarded (with a sufficiently large but less than the collateral amount) and MM loses all collateral. This ticketing mechanism is extremely efficient to compute.

The above ticketing mechanism already takes care of the worst-case scenario, i.e., reordering attacks. One could plug in any market making algorithm (to adjust asset prices) and obtain a reordering-resilient system. However, the MM can still drop trades, although it will be doing so without the knowledge of subsequent trades. We resolve this problem by carefully choosing a market maker—a *monopolistic profit seeking* market maker—that has economic incentive to not manipulate trading activity in this manner (e.g. by dropping trades obliviously of future trade requests). A monopolistic profit seeking MM uses trade requests as signals to determine where the true value of an asset lies (more buy trades $\implies$ true value of the asset is higher, more sell trades $\implies$ true value is lower). Its core principle is that, because a monopolistic profit seeking MM makes most profit when trading activity happens around the true value of an asset, it has no incentive to manipulate trading requests that would make the signals from trading activity less reliable. See Sect. 5 for formal discussion.

In addition to proving the security of FairMM, we have implemented and evaluated our design. We show that this design is extremely competitive. Concretely, we achieve a throughput of over 200 trades/minute. This is despite strict serialization of trade requests and the fact that we are running off-the-chain part on a relatively weak, consumer laptop (which communicates with an actual Ethereum node in test environment). These figures are about 50% higher than the maximum daily volume of Uniswap [23], arguably the most popular DEX and an order of magnitude higher than P2DEX [33], an order-book exchange implemented using MPC. We are also better than TEX [27], also an order-book exchange, which either does not support high frequency trading or is not frontrunning resilient. While Tesseract [25], another orderbook exchange, reports much higher throughput, it requires both trusted hardware and a consensus group assumption for its security guarantees. We, on the other had, require no such assumption.

In summary, our work makes the following contributions:

- We provide design of a non-custodial frontrunning resilient market-making crypto-currency exchange that does not require sophisticated cryptographic machinery (MPC, ZKPs, etc.), special hardware (TEEs) or additional assumptions (e.g. an additional consensus group in addition to the blockchain).
- We extract a useful abstraction—Σ-trade protocols—for asset exchanges that facilitates modular design of blockchain trading systems.
- We provide an instantiation of the system on Ethereum blockchain and demonstrate that it is very fast, and practical for real world applications.

1.1 Related Works

Market Makers for Crypto-token Markets. New emerging markets, e.g. prediction markets [8] or crypto-token markets, are typically thin and illiquid and often have to be bootstrapped through intelligent market makers to provide liquidity and price discovery [10]. A market maker algorithm aims at maximizing liquidity in the market and/or maximizing its own profit. The *zero-profit competitive market maker* model [1,4,9] considers multiple MMs that compete with each other by lowering their marginal profit to eliminate competition—such a system converges to a zero-profit. The *monopolist* market-maker, has been shown to provide greater liquidity than zero-profit competitive market makers [1,4,9,11]. We adopt the extension by Das [9] (cf. Sect. 5).

Fair Exchange and Blockchains. There is a large amount of literature on fair exchange including early MPC works [2,3,5,6,12], which has been re-ignited with the adoption of blockchains and cryptocurrencies [13,15–17,24,26]. Due to the relevance of these works to ours, we include a detailed review in the full version [35]. However, these works are not suitable for reuse in our design. Informally, the reason is that in our setting, fair exchange is a subroutine of the Market Maker (MM) protocol, and MM needs to know immediately whether a trade will settle or not on the blockchain. Therefore, we designed our own fair exchange protocol, a Σ-trade protocol, that we proved amenable to such composition.

Decentralized Exchanges. Popular decentralized exchanges e.g. Uniswap, Curve, Kyber, etc. [18–23,28,41] do not defend against frontrunning. To our knowledge, Tesseract by Bentov et al. [25] is the first work that addresses frontrunning in the crypto-currency space. It is orderbook based, custodial, and simulates a trusted third party (TTP) through trusted execution environments (TEEs). The assumption here is that since the exchange is a TTP, frontrunning does not happen. Since it relies on players to provide it with blockchain data, there is a check-pointing mechanism on trusted blocks, and if the exchange becomes unavailable, there is a *consensus group* of TEE backed nodes that can enforce/cancel transactions so that traders do not lose funds. In a similar vein, orderbook based P2DEX [33] by Baum et al. simulates TTP through outsourced MPC, their technique is similar to the work by Charanjit et al. [14] for traditional markets. TEX (Trustless Exchange) by Khalil et al. [27] is another orderbook exchange. It uses Zero Knowledge Proofs (ZKPs) for its guarantees. ZKPs are an expensive primitive and, in TEX, there is a trade-off as it either does not fully support high frequency trading or does not provide frontrunning resilience. In contrast to the above works, our construction is a *market maker*, it is *non-custodial*, does away with expensive primitives (MPC, ZKPs), additional requirements on hardware and/or additional check-pointing/consensus mechanisms, and provides frontrunning defense in a high frequency trading environment (as demonstrated by our detailed comparison with Uniswap in Sect. 6.3). Note however, that Tesseract supports cross-chain trading, our work does not.

Fairy by Stathakopoulou et al. [38] solve frontrunning for Byzantine Fault Tolerant (BFT) systems by augmenting Total Order Broadcast (TOB) protocols with *input causality* and *sender obfuscation*. They also require TEEs. Moreover, adapting this work to address frontrunning in crypto-currencies is non-trivial.

GageMPC [31] by Almashaqbeh et al. tackles privacy preserving auctions using non-interactive MPC (NIMPC). This work could be adapted into an exchange, but it is unclear whether it could handle high frequency trading. A^2MM by Zhou et al. [39] optimizes onchain swaps to *mitigate* frontrunning attacks. They study two point arbitrages for two assets. Their analysis holds assuming that all exchanges on a blockchain will be handled by A^2MM. This assumption is too strict for practical applications. Flashbots [36] and Gnosis Protocol V2 [37] both claim to resolve frontrunning. Flashbots requires strong trust (in the players to follow protocol) and is therefore not comparable to our work. Gnosis Protocol V2 claims that it will have a defense against frontrunning when it is built but currently there is no description of how it will be achieved. We refer to Chainlink 2.0 [34] whitepaper for details on existing techniques to achieve strict ordering of transactions. It also proposes a Fair Sequence Service (FSS) for Distributed Oracle Networks (DONs) that should solve this problem in general. However, exactly how such FSS will be implemented is not specified in the whitepaper.

There are some complementary works to ours which studies frontrunning. Flash Boys 2.0 [29] by Daian et al. give evidence that frontrunning is a serious problem on Ethereum. Bartoletti et al. [32] provide a theoretical framework to maximize miner extractable value (MEV), Sobol et al. [30] discuss frontrunning on proof of stake blockchains, and Zhou et al. [40] study sandwich attacks.

Next we dive into technical details of the paper, due to space constraints, we have provided relevant background in the full version [35].

2 Σ-Trade Protocols

A Σ-trade protocol Π is an interactive protocol run by a seller S and potentially many buyers $B_1, \ldots, B_m$ (seller S need not know m) where the exchange of tokens happens on blockchain E (We can think of E as the Ethereum blockchain).

Assume two tokens t_1 and t_2, each buyer wants to buy tokens of type t_2 in exchange of tokens of type t_1. Assume also that, each buyer B_i has an upper bound, denoted with $\mathtt{z}_i$, of type t_1 tokens that he can spend. The amount of t_2 tokens the buyer wants to buy is decided adaptively in the last round of interaction. A Σ-trade protocol Π consists of the following steps:

1. Each buyer B_i creates a smart contract $\mathtt{SC}_i$ on E that locks $\mathtt{z}_i$ tokens of type t_1 (more details on $\mathtt{SC}_i$ are provided later).
2. B_i and S exchange three off-chain messages. First, B_i sends his identities to the seller S. Note that B_i does not yet disclose his desired quantity t_2 tokens. In response, S proposes the exchange rate, `askedPrice`, for the tokens.
3. Let y be the quantity of tokens of type t_2 that the buyer wants to buy s.t. $y \cdot \mathtt{askedPrice} \leq \mathtt{z}_i$. If B_i agrees with `askedPrice`, then B_i sends a *certificate* c. This c can be used by S to invoke $\mathtt{SC}_i$ and withdraw $x = y \cdot \mathtt{askedPrice}$ tokens of type t_1 from B_i's account. However, $\mathtt{SC}_i$ will move the x tokens from B_i's account if S has moved to B_i's account y tokens of type t_2. $\mathtt{SC}_i$ ensures atomic transactions but can only be triggered by the seller.

Any instantiation of Σ-trade protocol can be used in our $\Pi^{\mathtt{trade}}$ protocol. We now show an insantiation of Σ-trade protocol to trade $\check{T}$ for Ξ.

2.1 Selling Tokens for Ethers

For a buyer B_i, denote with $(\mathtt{sk}_i^C, \mathtt{pk}_i^C)$ the signing-verification keys associated with account $C \in \{E, \check{T}\}$, where E represents Ethereum and $\check{T}$, a token on Ethereum. $\varXi$ denotes Ethereum currency. Similarly for seller, $(\mathtt{sk}_S^C, \mathtt{pk}_S^C)$ denote the signing-verification keys associated with account $C \in \{E, \check{T}\}$.

A formal description of the smart-contract and our protocol $\varPi$ is in [35]. Here, we give the intuition. The smart contract $\mathtt{SC}_i$ locks for T_i rounds $\mathtt{z}_i\varXi$ and manages a list of transaction identifiers. Upon receiving an input $(x, y, \mathtt{ID})$ that has been authenticated by both the buyer and the seller, $\mathtt{SC}_i$ moves $x\varXi$ to seller's account if 1) a transaction trx that moves $y\check{T}$ from the seller's account to the buyer's account has been made and 2) trx contains the identifier ID in its payload. In addition, to prevent replay attacks, $\mathtt{SC}_i$ does not allow reusing ID. The same contract $\mathtt{SC}_i$ can be used for multiple trades if $\mathtt{z}_i$ is big enough.

We now describe the protocol. The buyer sends his Ethereum public key to the seller, who replies with the exchange rate, askedPrice between $\varXi$ and $\check{T}$. If the buyer agrees with askedPrice and wants to buy $y\check{T}$ tokens, he generates an identifier ID, computes $x = y \cdot \mathtt{askedPrice}$ in $\varXi$, and signs $x||y||\mathtt{ID}$. He sends the signed values (and signature) to the seller. The seller, 1) posts a transaction trx that pays $y\check{T}$ into the buyer's account, trx contains ID in its payload, and 2) signs $x||y||\mathtt{ID}$, and uses the resulting signature, along with signature from the buyer, to invoke $\mathtt{SC}_i$. Note that the seller could post trx and also sends it to the buyer to indicate that the trade will occur.

3 (Fair) Ordering of Transactions

Our main contribution is the Universally Composable (UC) [7] formalization and realization of the *trade functionality* $\mathcal{F}^{\mathtt{trade}}$. $\mathcal{F}^{\mathtt{trade}}$ formally specifies the only ways in which the market maker can reorder the trades. For simplicity, assume that there are only two assets: $\varXi$ and $\check{T}$. Denote with $\mathtt{price}^{\check{T}\to\varXi}$ (and $\mathtt{price}^{\varXi\to\check{T}}$) the price at which MM sells $\check{T}$ (or $\varXi$) for $\varXi$ (for $\check{T}$). Assume that trader P_i's trade information is encoded in $\mathtt{trade}_i$. That is, $\mathtt{trade}_i$ describes the type and the amount of assets, the prices, trade direction (sell or buy) and etc. Moreover, assume that all the parties share the procedure MMalgorithm (the MM algorithm), which on input of a trade outputs the updated prices ($\mathtt{price}^{\check{T}\to\varXi}$ and $\mathtt{price}^{\varXi\to\check{T}}$). At a high level, $\mathcal{F}^{\mathtt{trade}}$ works as follows. Upon receiving a request from a trader P_i, $\mathcal{F}^{\mathtt{trade}}$ sends the prices to P_i, and signals to MM that P_i wants to trade. If P_i agrees with the prices, he sends trade information, $\mathtt{trade}_i$, to $\mathcal{F}^{\mathtt{trade}}$. Upon receiving $\mathtt{trade}_i$, $\mathcal{F}^{\mathtt{trade}}$ forwards $\mathtt{trade}_i$ to MM who has two choices: 1) decide not to trade with P_i by sending a command NO-TRADE to $\mathcal{F}^{\mathtt{trade}}$, or 2) accept trade with P_i. If MM does any other action before doing one of these two (e.g., MM starts trading with a party other than P_i), $\mathcal{F}^{\mathtt{trade}}$ allows that but also sets a special flag abort to 1. This means that if the traders query $\mathcal{F}^{\mathtt{trade}}$ with the command getTrades (to get the list of trades accepted by MM), $\mathcal{F}^{\mathtt{trade}}$ would return $\perp$ to denote that MM has misbehaved. A corrupt MM can also decide to set

$\Pi^{\mathtt{trade}}$

1) P_i**: creation of a request.** Send $(\mathtt{request}, \mathtt{pk}_i)$ to $\mathtt{MM}$.
2) $\mathtt{MM}$**: waiting for a request.** Upon receiving $(\mathtt{request}, \mathtt{pk}_i)$ from the party P_i do the following.
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{pk}_i)$ and set $h \leftarrow h'$ and $\sigma \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, h||\mathtt{pk}_i||\mathtt{e})$.
   – Send $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi \to \check{T}}, \mathtt{price}^{\check{T} \to \Xi}, \mathtt{pk}_i)$ to P_i and ignore any request that comes from any party $P_j \neq P_i$ for τ rounds.
3) P_i**: finalizing the request.** Upon receiving $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi \to \check{T}}, \mathtt{price}^{\check{T} \to \Xi}, \mathtt{pk}_i)$ from $\mathtt{MM}$, if the received prices are not satisfactory then send $\mathtt{NO\text{-}TRADE}$. Else create $\mathtt{trade}$ with all the required information with $\mathtt{trade}.p_1 = \mathtt{price}^{\Xi \to \check{T}}$ and $\mathtt{trade}.p_2 = \mathtt{price}^{\check{T} \to \Xi}$ and do the following:
   – If $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma, h||\mathtt{pk}_i||\mathtt{e}_i) = 1$ then compute $\sigma_i \leftarrow \mathtt{Sign}(\mathtt{sk}_i, h||\mathtt{trade})$ and send $(\mathtt{trade}, \sigma_i)$ to $\mathtt{MM}$, else ignore the message received from $\mathtt{MM}$
4) $\mathtt{MM}$**: reply to the request of** P_i**.** If $(\mathtt{trade}, \sigma_i)$ is received from P_i within τ rounds such that $\mathtt{Ver}(\mathtt{pk}_i, \sigma_i, h||\mathtt{trade})) = 1$ and $\mathtt{checkTrade}(\mathtt{trade}_i, \mathtt{price}^{\Xi \to \check{T}}, \mathtt{price}^{\check{T} \to \Xi}) = 1$ then do the following.
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{trade})$ and set $h \leftarrow h'$.
   – Add $(\mathtt{pk}_i, \mathtt{trade}, \sigma_i)$ to $\mathtt{requests}$.
   – Run $\mathtt{MMalgorithm}(\mathtt{trade}, \mathtt{price}^{\Xi \to \check{T}}, \mathtt{price}^{\check{T} \to \Xi})$ thus obtaining $\mathtt{price}^{\Xi \to \check{T}'}, \mathtt{price}^{\check{T} \to \Xi'}$ and set $\mathtt{price}^{\Xi \to \check{T}} \leftarrow \mathtt{price}^{\Xi \to \check{T}'}, \mathtt{price}^{\check{T} \to \Xi} \leftarrow \mathtt{price}^{\check{T} \to \Xi'}$.

   else do the following
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{NO\text{-}TRADE})$ and set $h \leftarrow h'$ and add $(\mathtt{pk}_i, \mathtt{NO\text{-}TRADE}, 0^\lambda)$ to $\mathtt{requests}$.

   Start accepting new requests from any party (i.e., goto step 2).
5) $\mathtt{MM}$**: posting trades to the BB**. If R rounds have passed, post $(h, \sigma, \mathtt{requests}, \sigma^\star, \mathtt{e})$ to the BB, where $\sigma \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, h)$ and $\sigma^\star \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, \mathtt{requests}||\mathtt{e})$. Set $R \leftarrow R + \Delta$, update the epoch number $\mathtt{e} \leftarrow \mathtt{e} + 1$ and reinitialize $\mathtt{requests}$.
6) P_i**: checking honest behavior of the** $\mathtt{MM}$. In each round P_i does the following
   - If no message $(h', \sigma', \mathtt{requests}, \sigma^\star, \mathtt{e}_i)$ has been posted on the BB within the last Δ rounds such that $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma', h') = 1$ and $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma^\star, \mathtt{requests}||\mathtt{e}_i) = 1$ then output $\perp$, else compute $\mathtt{e}_i \leftarrow \mathtt{e}_i + 1$ and continue as follows.
   - If P_i has not received a new ticket $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi \to \check{T}}, \mathtt{price}^{\check{T} \to \Xi}, \mathtt{pk}_i)$ during the epoch $\mathtt{e}_i - 1$ then continue, else if $\mathtt{verification}(h_{\mathtt{e}_i - 1}, h', h, \mathtt{pk}_i, \mathtt{requests}) = 0$ then send $(h, \sigma, \mathtt{pk}, \mathtt{e}_{i-1})$ to the BB as a proof of cheating of $\mathtt{MM}$ and set $\mathtt{output}_i \leftarrow \perp$.
   - Set $h_{\mathtt{e}_i} \leftarrow h'$.
7) P_i upon receiving $\mathtt{getTrades}$, if $\mathtt{output}_i = \perp$ then return $\perp$ else reinitialize $\mathtt{Trades}$ and do the following.
   - For each message $(h'_j, \sigma'_j, \mathtt{requests}_j, \sigma^\star_j, j)$ with $j \in \{0, \ldots, \mathtt{e}_i - 1\}$ such that $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma'_j, h'_j) = 1$ and $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma^\star_j, \mathtt{requests}_j||j) = 1$ posted on the BB, if $\mathtt{checkBB}(h_j, h'_j, \mathtt{requests}_j, \mathtt{pk}_{\mathtt{MM}}, j)$=0 then return $\perp$, else for each $(\mathtt{pk}, \mathtt{trade}, \sigma)$ in $\mathtt{requests}_j$ add $\mathtt{trade}$ to $\mathtt{Trades}$.
   - If $\mathtt{verifyPrices}(\mathtt{Trades}) = 1$ then output $(\mathtt{Trades}, \mathtt{e}_i)$ else output $\perp$.

Fig. 1. $\Pi^{\mathtt{trade}}$, the protocol that realizes $\mathcal{F}^{\mathtt{trade}}$.

the output of $\mathcal{F}^{\texttt{trade}}$ to always be $\perp$. This captures the fact that $\texttt{MM}$ can decide to stop working at his will. Moreover, $\texttt{MM}$ can add any trade of a corrupted party to the list of trades using the command $\texttt{setAdvTrade}$, but this can be done only after $\texttt{MM}$ has concluded any in-progress trades, as specified above.

$\mathcal{F}^{\texttt{trade}}$ is parametrized by Δ, which denotes the maximum number of rounds per *epoch*. In each *epoch* $\texttt{MM}$ should allow traders to see the entire list of trades. $\texttt{MM}$ can make the list of trades accessible via a special command $\texttt{setOutput}$. If $\texttt{MM}$ does not send this command at least every Δ rounds, $\mathcal{F}^{\texttt{trade}}$ will return $\perp$ to any honest party who requests trades list.

Note that $\mathcal{F}^{\texttt{trade}}$ allows the adversarial $\texttt{MM}$ to misbehave (e.g., by completely reordering the trades) but this misbehavior will be notified to the honest parties. Moreover, the MM cannot modify the trades (e.g., change the quantity that a party P_i is willing to sell/buy). Therefore, even if the adversary reorders the trades (at the cost of being detected), all the trades will be consistent with the prices that $\mathcal{F}^{\texttt{trade}}$ sent to the traders. The market maker still has the power to choose the parties he wants to trade with first, however, this choice has to be made obliviously of the trade information of the honest party. Luckily, we can also argue that for a relevant class of market-making algorithms, this does not constitute an additional useful power. We finally note that $\mathcal{F}^{\texttt{trade}}$ does not allow any real exchange of assets. However, if the output of $\mathcal{F}^{\texttt{trade}}$ is posted on a blockchain and if the trades are defined properly according to the language of the blockchain, then the $\texttt{MM}$ can use the trades to trigger events on the blockchain that move the assets according to what is described by $\mathcal{F}^{\texttt{trade}}$. We can also disincentivize any malicious behavior of the adversary by means of the compensation paradigm over the blockchain. Indeed, given that in our protocol all the honest parties can detect a malicious behavior without using any private state, the same can be done by a smart contract.

To simplify the description of our protocol, we make use of the procedures $\texttt{checkTrade}$ and $\texttt{checkPrices}$. $\texttt{checkTrade}$ takes as input $\texttt{trade}$, $\texttt{price}^{\check{T}\to\Xi}$ and $\texttt{price}^{\Xi\to\check{T}}$, and outputs 1 if the description of a trade $\texttt{trade}$ is consistent with the prices defined by $(\texttt{price}^{\check{T}\to\Xi}, \texttt{price}^{\Xi\to\check{T}})$. $\texttt{checkPrices}$ takes as input a list of trades and verifies that trade prices are consistent with $\texttt{MMalgorithm}$. These procedures, and $\mathcal{F}^{\texttt{trade}}$ are formally specified in [35].

3.1 Our Protocol: How to Realize $\mathcal{F}^{\texttt{trade}}$

Assume all parties have access to a bulletin board $\texttt{BB}$, all parties know the MM's public key, and the procedure $\texttt{MMalgorithm}$ is public. Our protocol realizes $\mathcal{F}^{\texttt{trade}}$ as follows. $\texttt{MM}$ maintains a hash chain (that starts with a value $h_{\texttt{start}}$), all parties know $h_{\texttt{start}}$. Whenever $\texttt{MM}$ receives a request from a trader P_i, he adds to the hash chain the public identity of P_i, signs the new head (say h_i), the public key of P_i and the current prices. We call this set of information a *ticket*. The MM then hands over the ticket to the trader. The trader checks that the signature is valid under the MM's public key, and if so, P_i defines the trade $\texttt{trade}_i$, signs it thus obtaining σ_i, and sends $(\texttt{trade}_i, \sigma_i)$ to $\texttt{MM}$ (σ_i guarantees that $\texttt{MM}$ cannot change $\texttt{trade}_i$). MM, upon receiving $\texttt{trade}_i$ and its signature, checks if $\texttt{trade}_i$ is well formed (i.e., the prices used to describe $\texttt{trade}_i$ are consistent with what MM sent in the previous round). If so, MM adds to the hash chain $\texttt{trade}_i$, adds $\texttt{trade}_i$ with σ_i to a list `requests`, run `MMalgorithm` on $\texttt{trade}_i$ and the current prices to get the new prices, and waits to receive next trade request.

In every epoch (at most $\varDelta$ rounds) MM publishes to the bulletin board[2] the head h of the hash chain and the list `requests`, all authenticated with his signing key. If MM that does not post such authenticated information within $\varDelta$ rounds then all the traders will understand this as an abort and output $\perp$. Each honest party that has access to the BB now: 1) checks that each trade in `requests` is either NO-TRADE or a correctly signed trade; 2) checks that all the prices are consistent with `MMalgorithm` and that the hash chain that starts at $h_{\texttt{start}}$ and finishes at h can be constructed using the trades in `requests`; and, 3) checks if the hash value h_i (received as part of the ticket) is part of the hash chain.

We observe that anyone (even traders who did not trade e.g. third parties) can check if the first and the second conditions hold. If either the first or the second condition does not hold, then all the honest traders output $\perp$. The third condition can be checked only by a trader who received a ticket. If a trader detects that the third condition does not hold, he can post his ticket on the bulletin board. At this point all the other parties who see BB can also determine that MM misbehaved and output $\perp$. Intuitively, our protocol realizes $\mathcal{F}^{\texttt{trade}}$ because once MM sends a ticket to a trader, he also commits to a set of trades. As long as MM cannot generate collisions for the hash function, he cannot include new trades in the hash chain. This protocol, $\varPi^{\texttt{trade}}$, is formally specified in Fig. 1. In the protocol, MM maintains $h \leftarrow 0^\lambda$, an initially empty list `requests` and the integers R, τ and $\varDelta$. $\varDelta$ represents the maximum number of rounds after which MM has to post the trades on the BB, τ represents the timeout (e.g. number of seconds, or rounds) before which a party has to reply to MM (to avoid DoS attack) and R is initialized to $\varDelta$. Let also $\texttt{SP}^{\check{T}\to\varXi}$ and $\texttt{SP}^{\varXi\to\check{T}}$ be the starting prices. MM also maintains an integer called *epoch index* denoted with $\texttt{e}$, MM initializes $\texttt{price}^{\varXi\to\check{T}} \leftarrow \texttt{SP}^{\varXi\to\check{T}}$ and $\texttt{price}^{\check{T}\to\varXi} \leftarrow \texttt{SP}^{\check{T}\to\varXi}$ and $\texttt{e} = 0$. Each party maintains and initially empty list `Trades`, $h_0 \leftarrow 0^\lambda$ and a view of the current epoch index which we denote by $\texttt{e}_i$.

The protocol uses utility procedures to check misbehavior. A formal description of these procedures is presented in the full version [35], a summary follows:

- `verifyPrices` takes as input a list of trades and checks each trade price is computed according to `MMalgorithm`.
- `verification` takes as input the ticket received by a trader, the head of the hash chain and the list of trades posted at the end of an epoch to the BB by MM, and checks whether the ticket appears in the hash chain and its consistency of trades list with hash chain.

[2] Publishing can be done cheaply e.g. by only posting the hash on the blockchain and providing hash-preimages on demand.

– `checkBB` checks BB for valid tickets and runs `verification` for each of them. If `verification` outputs 0, the procedure outputs 0 as well.

In the full version [35] we formally prove the following theorem:

Theorem 1. *Assuming that unforgeable signatures, and collision resistent hash functions exist, $\Pi^{\mathtt{trade}}$ realizes $\mathcal{F}^{\mathtt{trade}}$ in the $(\mathcal{F}_{RO}, BB)$-hybrid world.*

4 Combining $\mathcal{F}^{\mathtt{trade}}$ with Σ-Exchange Protocols

We observe that if, in the realization of $\mathcal{F}^{\mathtt{trade}}$, we replace the BB with a blockchain that supports smart contracts, then a smart contract can act as a party registered to $\mathcal{F}^{\mathtt{trade}}$ that can query $\mathcal{F}^{\mathtt{trade}}$ with the command `getTrades`. We can program this smart contract in such a way that if the output of $\mathcal{F}^{\mathtt{trade}}$ is $\perp$ then the MM is penalized. In our final protocol the traders and MM run a Σ-trade protocol Π, and in parallel, invoke $\mathcal{F}^{\mathtt{trade}}$ with the same information as input i.e. the prices, quantity and the type of the trades used in the execution of Π. Once that the output of $\mathcal{F}^{\mathtt{trade}}$ is generated, we can rely on a smart contract to check that the trades are consistent with the transactions generated by Π. If this is not the case then MM can be penalized. More precisely, to punish a misbehaving $\mathtt{MM}$ we require $\mathtt{MM}$ to create a smart contract $\mathtt{SC}_{\mathtt{penalize}}$ which locks a collateral z. $\mathtt{SC}_{\mathtt{penalize}}$, if queried by any party, inspects the output of $\mathcal{F}^{\mathtt{trade}}$ and if it is $\perp$ then $\mathtt{SC}_{\mathtt{penalize}}$ burns the collateral of $\mathtt{MM}$. Otherwise $\mathtt{SC}_{\mathtt{penalize}}$ checks whether the trades from $\mathcal{F}^{\mathtt{trade}}$ are consistent with the transactions generated by $\mathtt{MM}$ on the Ethereum blockchain with respect to the wallet addresses $(\mathtt{pk}^{\Xi}_{\mathtt{MM}}, \mathtt{pk}^{\check{T}}_{\mathtt{MM}})$. If they are not, $\mathtt{SC}_{\mathtt{penalize}}$ burns the collateral. We note that this contract is expensive to execute (in terms of gas cost). However, if $\mathtt{MM}$ and the traders follow the protocol nobody will ever invoke it. On the other hand, if $\mathtt{MM}$ misbehaves then a trader will detect it (from the output of $\mathcal{F}^{\mathtt{trade}}$) and will invoke $\mathtt{SC}_{\mathtt{penalize}}$. We incentivize the honest invocations of $\mathtt{SC}_{\mathtt{penalize}}$ by transferring a small portion of the locked collateral to the calling party before burning the rest of it.

To finish exposition, we need to introduce yet another smart contract, $\mathtt{SC}_{\mathtt{account}}$. This contract, too, is created by $\mathtt{MM}$. It checks if the transactions that pay the $\mathtt{MM}$'s account exceed a certain value Y. If this is the case, then the contract blocks additional payment towards $\mathtt{MM}$. Hence it bounds the amount of commodities that $\mathtt{MM}$ can trade, We do it to prevent a situation where profit of the $\mathtt{MM}$ exceeds the collateral and thus makes it rational to misbehave (and get penalized). Observe that no malicious (even irrational) $\mathtt{MM}$ can steal money from the traders. The worst that $\mathtt{MM}$ can do is to frontrun the traders (by letting $\mathcal{F}^{\mathtt{trade}}$ output $\perp$) or avoid posting transactions that allow the settling of the trades. Both these types of misbehavior is caught by $\mathtt{SC}_{\mathtt{penalize}}$ and $\mathtt{MM}$ loses collateral. Thus, if we set Y to be smaller than the collateral of $\mathtt{SC}_{\mathtt{penalize}}$, then it is not a viable strategy for a rational $\mathtt{MM}$ to be penalized by means of $\mathtt{SC}_{\mathtt{penalize}}$.

The formal description of our final protocol $\Pi^{\mathtt{full}}$ is in the full version [35]. We describe the case when traders only want to buy $\check{T}$ for Ξ. $\Pi^{\mathtt{full}}$ combines the functionality $\mathcal{F}^{\mathtt{trade}}$ and the Σ-trade protocol. We specify $\mathtt{SC}_{\mathtt{penalize}}$

in [35]. $\texttt{SC}_{\texttt{penalize}}$ acts like a party registered to $\mathcal{F}^{\texttt{trade}}$ who, when queried sends `getTrades` to $\mathcal{F}^{\texttt{trade}}$ and decides whether the MM misbehaved. Let T be the number of rounds for which $\texttt{SC}_{\texttt{penalize}}$ has locked the collateral, we can claim the following:

Theorem 2. *If there is at least one honest party P_i then, within the first T rounds one of the following occurs with overwhelming probability:*

1. *the $\mathcal{F}^{\texttt{trade}}$ outputs $\perp$ and the collateral locked in $\texttt{SC}_{\texttt{penalize}}$ by $\texttt{MM}$ is burned;*
2. *the $\mathcal{F}^{\texttt{trade}}$ is not $\perp$ but there is not a perfect correspondence between the trades contained in the output of $\mathcal{F}^{\texttt{trade}}$ and the transactions that appear on the blockchain E with respect to $\texttt{MM}$'s public keys. Moreover, the collateral locked in $\texttt{SC}_{\texttt{penalize}}$ is burned;*
3. *the $\mathcal{F}^{\texttt{trade}}$ is not $\perp$, there is a perfect correspondence between the trades contained in the output of $\mathcal{F}^{\texttt{trade}}$ and the transactions that appear on the blockchain E with respect to $\texttt{MM}$'s public keys and all the collateral remains locked in $\texttt{SC}_{\texttt{penalize}}$ for T rounds.*

For appropriate parameters in the smart contracts, and assuming the market-maker maximizes his amount of Ξ, we can argue that the first two cases in Theorem 2 happen with negligible probability. Indeed, let α be the gas cost to run $\texttt{SC}_{\texttt{penalize}}$ with the input `detected`, let `reward` be reward that could be given to a party calling $\texttt{SC}_{\texttt{penalize}}$, let z be the locked collateral in $\texttt{SC}_{\texttt{penalize}}$ and let Y be the maximum amount of Ξ that MM can earn at $\texttt{pk}^{\Xi}_{\texttt{MM}}$. If there is at least one honest party P_i, $\texttt{reward} > \alpha$, and $z > Y$ then for every rational market-maker the probability of occurrence of the first two cases of Theorem 2 is negligible.

5 Incentive Compatibility of Market Maker (MM)

We use myopic-greedy market maker from [11] in our construction. Here we provide an overview, see [35] for details and proofs. Let market maker's distribution $p_t(v)$ quantify market maker's information on the true value V after t trades, our market maker has the following properties:

– The market discovers the originally unknown true value of the commodity based on trades with traders who arrive with imperfect information. Empirically, the speed of this convergence is illustrated in [11] and follows the standard $1/t$ convergence for Bayesian updates.
– The market maker uncertainty converges to 0. The market maker recovers the true value in expectation, and also becomes more certain of it. Again, this convergence is standard for Bayesian updates.
– In equilibrium, the market maker spread that produces maximum single step profit monotonically increases with the variance of its distribution, which converges to zero. Hence the bid-ask spread converges to a minimum possible for a profit maximizing market maker. A market maker who knows V can always make more expected profit than a market maker who does not.

The last bullet above is essentially the intuition behind why an optimal market maker has no incentive to manipulate prices. The maximum profit is made when the market maker knows the true value V. Hence the market maker is incentivized to discover the true value V as quickly as possible. The only information available on the V is through the un-manipulated trader signals x_t.

We now present the main theorem (proved in [35]):

Theorem 3 (Incentive compatibility). *A rational profit-seeking market maker has no incentive to manipulate the price given knowledge that some trader wishes to place a trade and the direction (buy/sell) of the trade being known.*

The following lemma states that it is suboptimal for the market maker in our setting to ignore trades without knowledge of other trades.

Lemma 1. *A rational profit-seeking market maker which receives sequential trades, has no incentive to disregard completed trades, even when the direction of the following trade is known.*

6 Evaluation

We implemented $\Pi^{\texttt{trade}}$ to trade Ether and ERC20 tokens on Ethereum (see [35] for implementation details). Table 1 lists the gas costs. Note that the cost of executing one trade is the sum of the costs of *execute* methods of the *SellerContract* and the *BuyerContract*.

6.1 Experiment Setup

To measure throughput, we ran several experiments on a consumer laptop equipped with Core i7-10510U 1.80 GHz CPU and 8 GB of RAM running Ubuntu 20.04. Recall that in our fair trade protocol $\Pi^{\texttt{trade}}$ (see Fig. 1), the buyer P_i first sends its public keys to the seller `MM`. Then the seller responds by sending a ticket and the current prices. Both of these messages can be computed very cheaply. Concretely creating the first message takes less than 50 ms (for each party) in our setup. Then the buyer either responds with `NO-TRADE` or `trade`. This is still cheap and can be done in less than 50 ms. Now the seller must respond to the trade offer. If this offer is `NO-TRADE`, the buyer needs to perform very little work (concretely less than 50 ms). However, if the offer is `trade`, the seller must verify and create signatures, perform balance checks on appropriate assets and create/broadcast a transaction for the trade. These operations are slow (especially the ones that involve communicating with an Ethereum node). Concretely, it takes ≈350 ms to prepare this message. Lastly, we observed that the typical round trip time from buyer → seller → buyer is less than 100 ms.

Our goal was to observe the system's throughput in the following adversarial scenarios. The first is the *Standard DoS attack*. Here, a malicious buyer floods the system with ticket requests and then stops responding, slowing the system down. To this end, we performed the following experiment: n buyers connect to

Table 1. Gas costs of seller and buyer contracts.

Methods		Gas	USD[b]
Contract	Method	47gwei/gas[a]	3,284.20 usd/eth[a]
BuyerContract	claimExpiry	31,619	4.88
	execute	67,984	10.50
SellerContract	execute	33,456	5.16
Deployments			
BuyerContract		1,082,529	167.09
SellerContract		836,341	129.09

[a]Prices taken from https://coinmarketcap.com/ on 2021-09-10.
[b]USD cost is a bad measure of contract complexity. We list it to be consistent with other work.

the seller. The seller responds (with ticket and prices) to them in the order they connect. Upon receiving the ticket (and prices) from the seller, an honest buyer will execute a trade (i.e., the `trade` scenario). On the other hand, a corrupt buyer will stop responding. After a timeout τ, the seller will assume a `NO-TRADE` response, execute the `NO-TRADE` scenario, and move to the next buyer. We ran experiments with $n = 300$ buyers, repeating 5 times and reporting the average measurement. Note that relatively few repetitions of the experiments are not a concern because of low variance of the measured values.

The other attack scenario is a *Worst-case Throughput attack*. The setting remains the same as above with one difference: the malicious buyer now waits until just before the timeout and then responds with a `trade` response. This strategy is more effective at slowing down the system than the standard DoS attack. The reason why it is the case is discussed in the next section.

6.2 Analysis of Results

The results of the experiments for *Standard DoS attack* are summarized in Fig. 2, and for *Worst-case Throughput attack*, in Fig. 3. We observe that in Standard DoS attack (cf. Fig. 2) with no corruptions, throughput is over 200 trades/min. Recall that the gas cost of a trade is $101K$, Assuming *block gas limit* is $12M$ (i.e. the current limit) and block generation delay of 15 s, Upper bound on throughput is 475 trades/min. This upper bound assumes no other application (except ours) competes for block space. Keeping this in mind, achieving over 200 trades/min is an excellent throughput. This number is higher than Uniswap's [23] average throughput/min on its highest daily volume (cf. Sect. 6.3). Recall also that this throughput is achieved on a consumer laptop. A high-end server (typical machine for such use-cases) will yield higher throughput.

Interestingly, at low values of τ, the throughput of the system goes up with the number of corruptions. This is not an anomaly. If a malicious trader does not respond within the timeout τ, the seller assumes a `NO-TRADE` response which takes about one-seventh of the time it takes to execute `trade` response. This means at low values of τ ($\tau <$ execution-timet of `trade`) and some corruptions, some (i.e. the corrupt) trades are cheaper to execute compared to when there are no corruptions (because all honest players trigger the `trade` scenario). This effect disappears as soon as the value of the timeout τ goes near and above the execution-time of the `trade` scenario. While setting a low timeout τ may seem a good idea to defend against malicious parties, it should not be less than the typical round-trip time (100 ms in our trials), otherwise it will cause timeouts for honest players. Note also that a trader needs to setup a smart contract and register with the market maker before commencing trading. Therefore, Sybil attack is not trivial and repeat offenders may be blacklisted.

A better attack strategy would be for a malicious buyer to wait until just before the timeout (for maximum slowdown) and then respond with `trade` response; to trigger the more expensive (in running time) scenario for seller. This strategy removes the above mentioned advantage. Concretely, a malicious seller would wait until he has just enough time left for one round-trip (100 ms in our setup). Thus the amount of time he should wait, `delayBudget`, can be computed as $\texttt{delayBudget} = \tau - \texttt{RoundTripTime}$. The negative effect of such attack is seen in Fig. 3. The throughput has gone down for all values of τ. Importantly though, observe that the x-axis in Fig. 3 starts at $\tau = 200$. This is because at $\tau = 100$, the `delayBudget` of the adversary is 0 i.e., he has to respond immediately and there is no longer a difference between an honest buyer and a malicious buyer.

In conclusion, the choice for value of τ should be the typical round-trip time (with some noise). This prevents throughput-loss even against a determined adversary who wants to pay (via `trade` responses) to slow down the system. Finally, consider that in real life some honest sellers may also respond with `NO-TRADE` e.g., if the prices are not favorable. Hence, the value of 205 trades per minute at $\tau = 100$ should be considered the lower bound.

6.3 Comparison with Uniswap

A summary of FairMM and Uniswap is presented in Table 2. See [35] for our analysis of Uniswap gas costs. At the time of this writing, the throughput values in v2 are higher than v3 e.g. the highest daily volume 3 times more for v2 (251K txns) than v3 (71K txns). So, we compare against v2.

Second, our trade execution time is bounded by the round trip time of the network, about 350 ms. In contrast, Uniswap trades are executed by the miners as part of mining a block. At the time of this writing, etherscan shows high fees transactions (ones that get picked up the soonest) take about 30 s. One can safely say that a trade in Uniswap takes at least 15 s (half of the value on etherscan). This is much larger than the approx. 0.3 s in our system.

First, Uniswap (or any existing market maker, centralized or decentralized) has no defense against front-running attacks without additional trust/hardware assumptions. Our construction resolves this long-standing problem by ensuring that the market maker cannot reorder trades without getting caught.

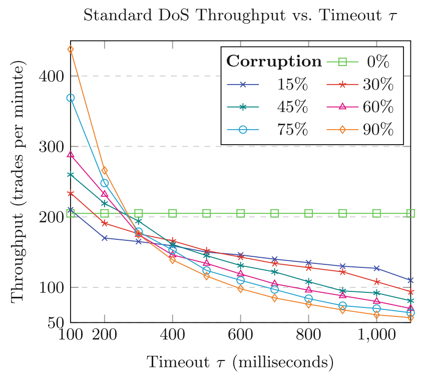


Fig. 2. Standard DoS Attack Throughput (at 0% to 90% corruption). Values average of 5 runs. Note that x-axis starts at ms, this is the typical RTT, and any $\tau < 100$ may cause timeouts for honest players.

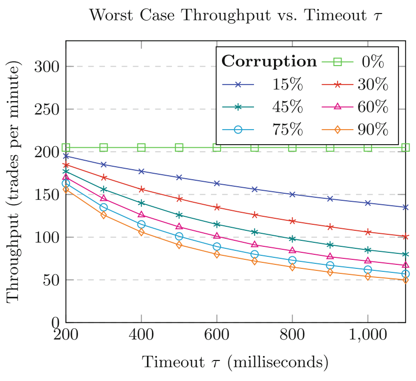


Fig. 3. Worst-case Throughput (at 0% to 90% corruptions). Note: x-axis starts at 200 ms because malicious player needs a budget of at least RTT (100 ms here) to respond without risking timeout.

Third, in Uniswap and similar systems, miners are free to reorder trades. This gives them a profit opportunity e.g. including favorable trades first. On the contrary, in our system the trade order is fixed before the corresponding transactions are broadcast to the blockchain, nullifying miners' influence. Moreover—

Table 2. Comparison Summary

Feature	FairMM	Uniswap
Front running resilience	**Yes**	No
Gas price auctions	**No**	Yes
Miner influence	**No**	Yes
Trade execution (seconds)	$\approx$**0.30**	$\geq$15
Average trade cost $\text{l}(K)$	$\approx$**101**	$\approx$141[a]
Max trade cost (K)	$\approx$**101**	$\approx$1,316[b]
Max throughput[c]	$\approx$**475**	$\approx$340

[a]Based on average cost of 1M trade transactions (block 12,162,664 to 12,231,464). Trades are calls to swap methods of V2Router02.

[b]Txn: https://etherscan.io/tx/0xa87b492f2945d2a99ca1f8e2d9530599c040f00c3257f989f9c2822e20b2ed5e). There may be more expensive transactions outside our dataset.

[c]in trades/minute. Theoretical upper bound on throughput based on average trade cost, assuming 12M block gas limit on Ethereum network.

because of the above mentioned miners' influence—traders on Uniswap have an incentive to pay high *gas price* to get their trade included sooner. In fact, since the traders can see other traders' activity, they can actively compete with one another. Such trading behavior induces the, so called, *gas price auction*s attack. Gas price auctions needlessly raise transaction cost for everyone (not just the traders). Transactions in our system are merely moving the funds and may be mined in any order. There is no incentive to pay higher than usual gas price.

Fourth, gas cost in Uniswap is variable. We observed an average gas cost of $141K$. It can be much higher depending on the trade e.g. over $1,316K$ for txn 0xa87b492f2945d2a99ca1f8e2d9530599c040f00c3257f989f9c2822e20b2ed5e. Recall that Uniswap is specifically designed and optimized for Ethereum. On the other hand, our system design is general and lacks aggressive optimizations. Yet, the gas cost of our system is constant at $101K$. Notwithstanding, even if the gas cost of Uniswap transactions were much lower than ours, Uniswap's transactions would still be more costly in Ethers because of the gas price auctions mentioned above.

Finally, based on the average trade gas cost and assuming a block gas limit of $12M$, the maximum throughput of Uniswap is $\approx$ 340 trades per minute. This is less than our upper bound of 475. Concretely, highest daily volume[3] on Uniswap has been $\approx 251K$ transactions. On average, this means about 174 trades per minute. Importantly, this throughput is achieved in a scenario where all trade data is locally available. Our construction on the other hand, communicates with the traders in real time. The fact that this communication happens sequentially—on first come first served basis—negatively affects our throughput. Despite this, we achieve at least 200 trades per minute (higher than the highest volume Uniswap). We stress that this throughput was achieved on a mid-range consumer machine. A computationally powerful server will increase throughput further. Therefore, we do not see it as a major problem in practice.

References

1. Glosten, L.R., Milgrom, P.R.: Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. J. Financ. Econ. **14**(1), 71–100 (1985)
2. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. IEEE Computer Society Press, pp. 162–167, October 1986
3. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987
4. Glosten, L.R.: Insider trading, liquidity, and the role of the monopolist specialist. J. Bus. **62**(2), 211–235 (1989)
5. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054156
6. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 93–111. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_6

[3] https://etherscan.io/address/0x7a250d5630b4cf539739df2c5dacb4c659f2488d#analytics.

7. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
8. Wolfers, J., Zitzewitz, E.: Prediction markets. J. Econ. Perspect. **18**(2), 107–126 (2004)
9. Das, S.: A learning market-maker in the Glosten-Milgrom model. Quant. Fin. **5**(2), 169–180 (2005)
10. Pennock, D., Sami, R.: Computational aspects of prediction markets. In: Algorithmic Game Theory. Cambridge University Press (2007)
11. Das, S., Magdon-Ismail, M.: Adapting to a market shock: optimal sequential market-making. In: Proceedings of the Advances in Neural Information Processing Systems (NIPS), pp. 361–368 (2008)
12. Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 252–267. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11925-5_18
13. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
14. Jutla, C.S.: Upending stock market structure using secure multi-party computation. Cryptology ePrint Archive, Report 2015/550 (2015). https://eprint.iacr.org/2015/550
15. 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, Part II. LNCS, vol. 9879, pp. 261–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_14
16. Kiayias, A., Zhou, H.-S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
17. Campanelli, M., et al.: Zero-knowledge contingent payments revisited: attacks and payments for services. In: Thuraisingham, B.M., et al. (eds.) ACM CCS 2017. ACM Press, pp. 229–243 (2017)
18. Warren, W., Bandeali, A.: Ox: an open protocol for decentralized exchange on the Ethereum blockchain (2017)
19. AirSwap: AirSwap (2018)
20. Ether Delta: EtherDelta (2018)
21. IDEX: IDEX (2018)
22. Kyber: Kyber (2018)
23. Uniswap: Uniswap Exchange Protocol (2018)
24. Bitcoin Wiki: Zero Knowledge Contingent Payment (2018)
25. Bentov, I., et al.: Tesseract: real-time cryptocurrency exchange using trusted hardware. In: Cavallaro, L., et al. (eds.) ACM CCS 2019, pp. 1521–1538. ACM Press, November 2019
26. Fuchsbauer, G.: WI is not enough: zero-knowledge contingent (service) payments revisited. Cryptology ePrint Archive, Report 2019/964 (2019). https://eprint.iacr.org/2019/964
27. Khalil, R., Gervais, A., Felley, G.: TEX - a securely scalable trustless exchange. Cryptology ePrint Archive, Report 2019/265 (2019). https://eprint.iacr.org/2019/265
28. Curve: Curve (2020)

29. Daian, P., et al.: Flash Boys 2.0: frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy, pp. 910–927. IEEE Computer Society Press, May 2020
30. Sobol, A.: Frontrunning on automated decentralized exchange in proof of stake environment. Cryptology ePrint Archive, Report 2020/1206 (2020). https://eprint.iacr.org/2020/1206
31. Almashaqbeh, G., et al.: Gage MPC: bypassing residual function leakage for non-interactive MPC. Cryptology ePrint Archive, Report 2021/256 (2021). https://eprint.iacr.org/2021/256
32. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: Maximizing extractable value from automated market makers. In: CoRR abs/2106.01870 (2021)
33. Baum, C., David, B., Frederiksen, T.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. Cryptology ePrint Archive, Report 2021/283 (2021). https://eprint.iacr.org/2021/283
34. Breidenbach, L., et al.: Chainlink 2.0: next steps in the evolution of decentralized oracle networks (2021)
35. Ciampi, M., et al.: FairMM: a fast and frontrunning-resistant crypto market-maker. Cryptology ePrint Archive, Report 2021/609 (2021). https://ia.cr/2021/609
36. Flashbots: Flashbots (2021)
37. Gnosis: Introducing Gnosis Protocol V2 and Balancer-Gnosis-Protocol (2021)
38. Stathakopoulou, C., et al.: Adding fairness to order: preventing front-running attacks in BFT protocols using TEEs. In: 40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, 20–23 September 2021, pp. 34–45. IEEE (2021)
39. Zhou, L., Qin, K., Gervais, A.: A2MM: mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. In: CoRR abs/2106.07371 (2021)
40. Zhou, L., et al.: High-frequency trading on decentralized on-chain exchanges. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 428–445 (2021)
41. Bancor: Bancor Network

In-App Cryptographically-Enforced Selective Access Control for Microsoft Office and Similar Platforms

Karim Eldefrawy[1(✉)], Tancrede Lepoint[2], and Laura Tam[1]

[1] SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, USA
{karim.eldefrawy,laura.tam}@sri.com
[2] New York, USA
crypto@tancre.de

Abstract. The interplay between cryptography and access control has been widely investigated in the literature. For example, attribute-based encryption (ABE) is a leading candidate of a cryptographic tool going beyond the all-or-nothing approach of public-key encryption by supporting fine-grained access control for encrypted data. Unfortunately, the deployment and adoption of ABE have been slow, and (to the best of our knowledge) few commercial widely-used products use it to date. In particular, selective and fine-grained control over what is shared, and with whom, is absent from common data products and formats, such as those generated by commercial authoring products, e.g., Microsoft Word documents, Excel spreadsheets, PowerPoint slides. This lack of selective and fine-grained control results in users simply *not sharing*. This major usability shortcoming impacts defense and military coalition operations, as well as commercial settings, such as life sciences, healthcare, and the financial sectors.

This paper addresses the above usability problem head-on by proposing a crypto- graphically enforced selective access control in Microsoft Office products and similar platforms. We focus on Excel as an illustrative use-case, but note that our work is applicable to (and is already implemented for) other Microsoft products such as Word, PowerPoint, and Outlook. Using the JavaScript API for Microsoft Office, we designed and developed simple add-ins that enable cell encryption according to a policy, and requires a key that embeds attributes satisfying the policy in order to decrypt. Our performance evaluation not only shows that cryptographic-based selective sharing of information in widely-deployed and widely-used commercial authoring and collaboration platforms is possible, but also practical.

1 Introduction

Private data sharing remains as of today a critical challenge for individuals, enterprises, and (inter)national organizations, and governments. While sharing data is essential, sharing sensitive data with the wrong entity can have devastating consequences or even be prohibited [2, Sect. 1201]. Fortunately, the literature is rich with

K. Eldefrawy and T. Lepoint—Contact authors.
T. Lepoint—Work performed while at SRI International.

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 447–467, 2022.
https://doi.org/10.1007/978-3-031-07689-3_32

cryptographically-enforced access control solutions. A cryptographic tool that naturally lands itself to fine-grained access control is that of attribute-based encryption (ABE) [25]. Here, ciphertexts and keys are associated with attributes which determine when decryption is possible. In a ciphertext-policy ABE (CP-ABE) [13], keys are associated with attributes like '`(continent = Europe) (trust = 2) (org = NATO)`', while ciphertexts are associated with access policies such as '`((continent == Europe) AND (org == NATO)) OR (trust > 3)`'. Decryption is only possible when the key attributes satisfy the policy.

Using the previous example, note that the encryptor does not need to know the exact identities of all other entities who should be able to access the data, but rather determine them in term of descriptive attributes. Would they be issued keys, countries like France, Italy, or Belgium would be able to decrypt, while Sweden or Finland would not (as they are not members of NATO) unless their key embeds an attribute `trust` larger or equal to 4.

Over the past decade, ABE has led to a bounty of applications, from network privacy to health record access-control and secure messaging. While companies like Zeutro [8] started investigating the use of ABE in Cloud applications, to date the deployment of ABE has been slow. This can be explained by a variety of reasons. Amongst them, (1) it became clear that ABE schemes need to *readily* accommodate new roles, attributes, and access policies to be used; and (2) real-world applications of ABE require strong security guarantees under realistic and natural attack models. Thankfully, these initial concerns are no more. The first requirement has been achieved in 2011 by Lewko and Waters [21] in what is called *unbounded ABE* (that is, an ABE that is not bounded in the number of attributes it can handle). Since then, unbounded ABE constructions have been widely improved and made efficient [9,11,12,15,18–20,23,24]. As for the security concern, recent ABE schemes are based on well-understood security assumptions against active adversaries [9,14,15], and rely on asymmetric prime-order (Type-III) pairings, the recommended choice by cryptography experts [17].

In most commercially deployed settings today, fine-grained access control is not achieved by cryptographic means. When selective access control is available (e.g., the privacy controls proposed by Facebook, or even in online Excel workbooks [7]), it typically relies on a (replicated) trusted centralized system that shares with a recipient the data she is authorized to see. To selectively share information that is contained in the most used document formats without a centralized system (e.g., `docx` for text, `jpeg` for images, `xlsx` for spreadsheets, or `pptx` for presentations), the commonly used process is to *manually* remove sensitive information and produce multiple versions of the same file while selecting content in each version that depends on the recipient of that version. In our extensive discussion with government and defense agencies, we learned that Microsoft Office has become a de facto means of sharing information between countries or agencies, and that the lack of selective access control within results in people *simply not sharing*. This major usability shortcoming impacts DoD coalition operations in the U.S. This problem is not restricted to the defense setting: analogous problems can easily be identified in commercial settings in other sectors such as the finance, healthcare, and pharmaceuticals. Additionally, both military computers and company-owned computers are often subject to strict conditions or restrictions regarding the softwares that can be installed and run.

Motivated by this state of affairs, this paper addresses the usability problem described above head-on, by showing that fine-grained access control solutions can be made available and naturally used in widely-deployed products. This paper introduces a Microsoft Excel add-in that enables cell[1] encryption according to policies. The add-in is a single-page web application written in JavaScript that interacts with the object models in Excel using the JavaScript API for Office [1]. A minimal locally hosted service (dockerized and running on a user's device), accessible through a REST API, enables to encrypt and decrypt using a CP-ABE scheme. We note that such a dockerized service is just to simplify our development, deployment, and experimentation. *We stress that in a production deployment, the entire encryption and decryption could/should be performed on the client-side in the add-in webpage itself via JavaScript. We verified that this is possible and already implemented this in later versions of the add-ins after submitting this paper.* After encryption, the document remains a *valid Excel document* and can be opened and read (without the encrypted cells) as *any* other `xlsx` document by *any* software. More precisely, the encrypted cells are stored in an XML custom part of the `xlsx` document through the JavaScript API. When our add-in is loaded from the Microsoft Excel software (or, e.g., from the Online Excel of Office 365) by a user in possession of a CP-ABE key, all the cells with a policy satisfiable by the key attributes will be decrypted and displayed. Henceforth, the same `xlsx` document can be shared with a wide audience while enabling selective access control at a cell level.

Organization: The outline of the paper is as follows. Section 2 introduces some necessary background on Microsoft Office add-ins and CP-ABE. Section 3 presents our Excel add-in, and a performance evaluation is presented in Sect. 4. Section 5 discusses potential alternative options for short-term adoption of similar selective sharing functionality based on standardized encryptions schemes but with some limitations. Finally, we conclude the paper and discuss future work in Sect. 6.

2 Preliminaries

This section provides some common background and notation on Office add-ins, ciphertext-policy attribute-based encryption (CP-ABE), and the Charm framework.

2.1 Microsoft Office Add-In

An Office add-in is a web application that is loaded from a browser inside of an Office application (desktop and online). An add-in is not installed on the host, but has its implementation hosted on a web server. Add-ins can be added in an Office application either by providing a XML manifest file (with the URL of the web application), or through the Office store.

[1] We extended the work while under review to enable row, and/or column, or full document encryption.

As with any web services, add-ins can access any web-based resources. Accessing and modifying information in the Office document is made possible by referencing the `office.js` file containing the JavaScript API for Office [1]. The add-in logic is developed in JavaScript:

```
Office.initialize = function () {
  // Office is ready
  $(document).ready(function () {
    // Implementation of add-in logic
  });
};
```

Once initialized, an add-in can access data in the underlying application. For example, the code in Fig. 1 recovers the content (the formulas) of the current selected cells in Excel.

```
Excel.run(function(ctx) {
  var range = ctx.workbook.getSelectedRange();
  range.load('formulas');
  return ctx.sync().then(function() {
    var content = range.formulas;
    // Process the content
  });
});
```

Fig. 1. Snippet of code to recover the content of a range of cells.

We refer to the official documentation for further detail on the Office add-in platform [3].

2.2 Access Structures

An access structure specifies the set of attributes required to gain access to some secrets.

Definition 21 (Access structure). *Let $\mathcal{U}$ be a universe of attributes. An access structure $\mathbb{A}$ is a collection of non-empty subsets of $\mathcal{U}$. An access structure $\mathbb{A}$ is called monotone if, for every $B \subseteq C \subseteq \mathcal{U}$, $B \in \mathbb{A} \Rightarrow C \in \mathbb{A}$.*

In this paper, we will define access control in terms of *policies* over attributes with AND and OR gates (cf. Sect. 3.6), that are then converted into access structures to be used by the CP-ABE scheme.

2.3 Ciphertext-Policy ABE

Let λ denote the target bit-security of the cryptographic scheme (a.k.a, the security parameter). A ciphertext-policy ABE scheme $\mathsf{CP\text{-}ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is a tuple of probabilistic algorithms together with a message space $\mathcal{M}$ that behave at follows:

– Setup takes as input the security parameter λ and outputs a public key pk and a master secret key msk.
– Enc takes as input the public key pk, a message m and an access structure $\mathbb{A}$, and outputs a ciphertext c.
– KeyGen takes as input the master secret key msk and a set of attributes S, and outputs a secret key sk.
– Dec takes as input the public key pk, a ciphertext c and a secret key sk, and outputs m^* or $\perp$.

A CP-ABE scheme must satisfy the following *correctness* condition: for all $m \in \mathcal{M}$, access structure $\mathbb{A}$, and set of attributes $S \in \mathbb{A}$, it holds that

$$\Pr\left[\mathsf{Dec}(\mathsf{pk}, c, \mathsf{sk}) \neq m \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)\\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{A}, m)\\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, S)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $a \leftarrow A(b)$ denotes the output of the algorithm A when run on input b and $\mathsf{negl}(\lambda)$ denotes a negligible function, i.e., a function which is smaller than the inverse of any polynomial for large enough values of λ.

In this paper, we only consider fully-secure CP-ABE schemes. We recall here the intuition, and refer to [9, Sect. 2.3] for a formal definition. A CP-ABE scheme is fully-secure against chosen plaintext attacks if, at any time after the deployment of the ABE scheme, no group of colluding users can distinguish between encryption of two messages of their choice, under an access structure (a.k.a., a policy) of their choice, as long as no member of the group can decrypt on their own.

2.4 Charm

Charm [10] is (mostly) a Python-based framework for prototyping advanced cryptosystems. It uses a hybrid design: performance intensive mathematical operations are implemented in native C modules, while cryptosystems themselves are written in Python. In particular, Charm uses the Pairing-Based Cryptography Library [4] for elliptic-curve generation, operations, and cryptographic pairing implementations.

The scheme we use in our system is FAME, a CP-ABE scheme proposed at the 2017 ACM CCS by Agrawal and Chase [9]. We use without modification the authors' implementation of FAME (as incorporated into Charm).

3 The Excel Add-In

This section presents our Excel add-in and its workflows.

3.1 Setting

Our add-in assumes the existence of the following entities/services:

- An entity O, who will create CP-ABE parameters

$$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda).$$

  O is the only entity knowing the master secret key msk, hence the only party that will be able to create secret keys sk's associated to sets of attributes.
- A web service W, which will be hosting the add-in web application (cf. Sect. 2.1). This service may *or may not* be controlled by O.
- A service E (e.g., a web service accessible through a REST API), which will enable to encrypt with the CP-ABE scheme. This service may be completely independent of O and W; it only implements the Enc operation of the CP-ABE scheme.
- A service D (e.g., a web service accessible through a REST API), which will enable to decrypt with the CP-ABE scheme. This service may be completely independent of O, W and E; it only implements the Dec operation of the CP-ABE scheme.
- n entities P_i's that will be issued secret keys by O over time; these will receive a protected spreadsheet and use the add-in to recover the information they are entitled to see.

Our add-in enables everyone (i.e., the above entities *or anybody else*) to create selectively protected spreadsheets. In particular, it will enable to select cells and encrypt them according to policies. Therefore, our add-in only requires to know which universe of attributes it should use to allow policy creation.

Additionally, in order to run the Enc and Dec algorithms, the services E and D need to know the public key pk. In our evaluation (Sect. 4), we assume the public key to be embedded in the services E and D. An alternative option could be for the add-in to be configured at load time with the public key and to send this public key along with the message and policy (resp., with the ciphertext and the secret key) every time it asks for encryption (resp., decryption).

3.2 (Offline) Key Distribution

This key distribution is *decoupled* from the Excel add-in, and may be performed offline and out-of-band.

The entity O is the only entity that can issue secret keys sk's. In the following, we assume that O issued and shared one or more secret keys sk_{ij}'s to each party P_i. Note that a key sk will be used to decrypt *several spreadsheets over time* (as long as its attributes can decrypt cell policies).

In the following, we assume the sk_{ij}'s are stored in (say) JSON files.

3.3 Spreadsheet Creation

As recalled above, anyone using the add-in may activate the privacy protection in an Excel spreadsheet.

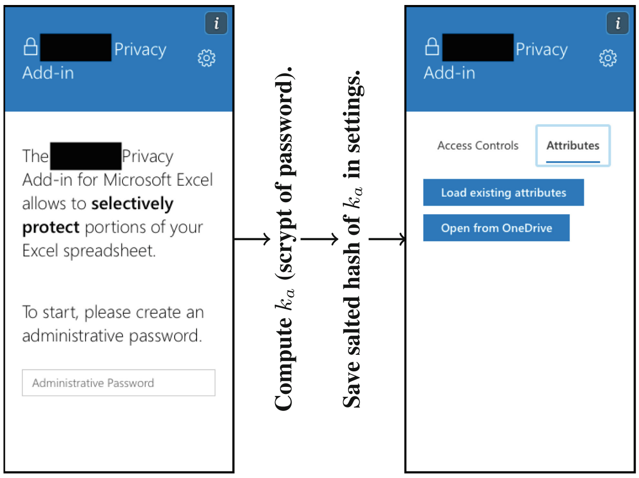
The activation of the add-in follows the following workflow:

1. Display a page to enable the activation of the privacy protection. This page asks to create an administrative password (Fig. 2).

2. Upon submission of a password p_a, apply the scrypt password-based key derivation function thereon to obtain a key k_a (our add-in uses the scrypt-async library [5]). (This administrative key will encrypt any administrative-related data.)
3. Store a salted hash $H(k_a; n_a)$ of the key k_a in the

   ```
   Office.context.document.settings
   ```

   object. The hash is computed using the TweetNaCl.js library [6]. This object is saved in the Excel document, and will be accessible via any Excel application.
4. In order to enable cell encryption, a configuration file containing the list of all attributes needs to be loaded in the add-in. An example of such file is provided in Fig. 3.
5. Store the list of attributes encrypted under k_a in the settings.


[redacted] Privacy Add-in
Compute k_a (scrypt of password).
Save salted hash of k_a in settings.
[redacted] Privacy Add-in
Access Controls
Attributes
Load existing attributes
Open from OneDrive

The [redacted] Privacy Add-in for Microsoft Excel allows to **selectively protect** portions of your Excel spreadsheet.

To start, please create an administrative password.

Administrative Password

Fig. 2. Workflow to create a privacy-protected Excel spreadsheet. (The screenshots are anonymized for submission.)

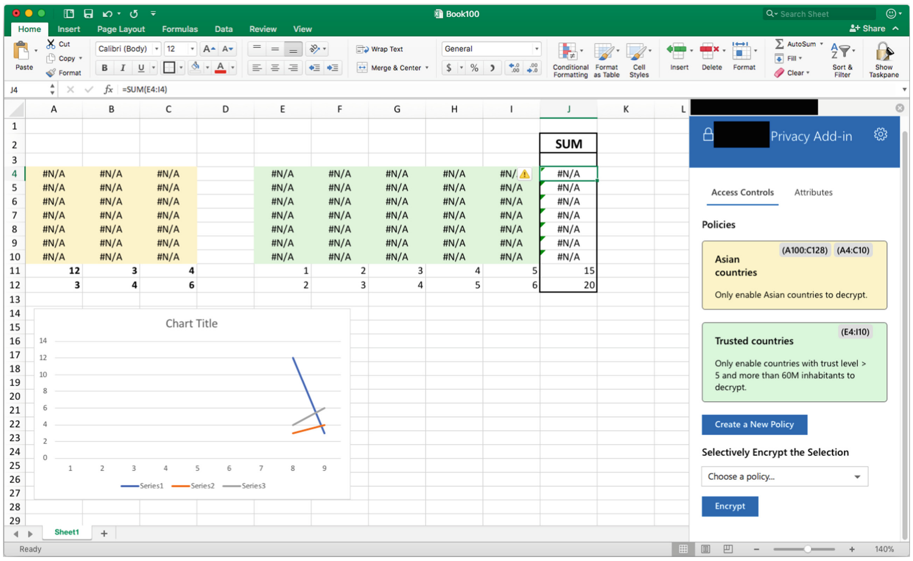
```
{
  "Continent": ["Africa", "Antarctica", "Asia", "Australia", "Europe",
       "North America", "South America"],
  "Country": ["Canada", "China", "France", "Japan", "Russia", "UK"],
  "Trust level": [1,2,3,4,5],
  "Organization": ["BRICS", "G20", "G7", "NATO", "WTO"],
}
```

Fig. 3. An example of configuration file containing the list of all attributes.

At the end of the spreadsheet creation, the settings contain a hash $(n_a, H(k_a; n_a))$ of the administrative key k_a, and the list of attributes encrypted under k_a.

3.4 Encryption

In this subsection, assume a user U wants to add cell encryption in a `xlsx` document created as in Sect. 3.3. Figure 4 shows a screenshot of the add-in after encryption of three ranges of cells with two policies.



J4 =SUM(E4:I4)

	A	B	C	D	E	F	G	H	I	J	K
1											
2										SUM	
3											
4	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
5	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
6	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
7	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
8	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
9	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
10	#N/A	#N/A	#N/A		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
11	12	3	4		1	2	3	4	5	15	
12	3	4	6		2	3	4	5	6	20	

[redacted]

[redacted] Privacy Add-in

Access Controls   Attributes

Policies

(A100:C128) (A4:C10)
Asian countries
Only enable Asian countries to decrypt.

(E4:I10)
Trusted countries
Only enable countries with trust level > 5 and more than 60M inhabitants to decrypt.

Create a New Policy

Selectively Encrypt the Selection

Choose a policy...

Encrypt

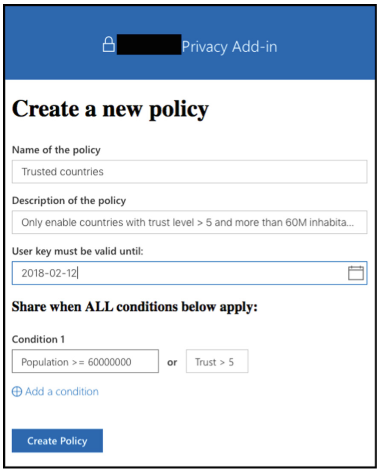
Fig. 4. Screenshot of add-in after encrypting three ranges of cells: `A4:C10` and `A100:C128` are encrypted with a policy `(continent == Asia)` (named "Asian countries" by the user), and `E4:I10` is encrypted with a policy `(trust > 5) OR (population >= 60000000)` (named "Trusted countries" by the user). Column J computes the sum of the values in the columns E to I for each row; note that it outputs `#N/A` when the cells are encrypted. The chart display the cells in the range `A4:C12`; note that only unencrypted values are visible in the chart.

Authentication. In the current version of the add-in, we only enable cell encryption when U knows the administrative password. This is not necessary and one may choose to remove this authentication step. Note that from Sect. 3.3, only the attributes are stored in the settings; future work may include additional (encrypted) content in the settings, which explains why we implemented this more general approach.

The workflow at load time is as follows:

1. Check the presence of a hash $(n_a, H(k_a; n_a))$ in the settings. If defined, display a login screen.
2. Upon submission of a password p_u by U, apply the scrypt password-based key derivation function thereon to obtain a key k_u (our add-in uses the scrypt-async library). If $H(k_u; n_a) = H(k_a; n_a)$, decrypt the attributes from the settings (if any) and populate the UI accordingly; if not, go back to Step 1.

Upon success of Step 2, U will be considered "authenticated".


[redacted] Privacy Add-in

Create a new policy

Name of the policy: Trusted countries

Description of the policy: Only enable countries with trust level > 5 and more than 60M inhabita...

User key must be valid until: 2018-02-12

Share when ALL conditions below apply:

Condition 1: Population >= 60000000 or Trust > 5

Fig. 5. Pop-up that enables creation of conjunctive normal forms policies, that is policies of the form `(expr11 OR ... OR expr1i) AND (expr21 OR ... OR expr2j) AND (expr31 OR ... OR expr3k)`.

Encryption. Assume U wants to encrypt a range of cells (say `A1:B4`) under a policy of her choice. U will select the cells in the range (`A1:B4`) in Excel, will use the policy UI to create a policy (cf. Fig. 5), and will click on the "Encrypt" button.

Upon click, the encryption workflow is as follows:

1. Get the content of the selected range (see Fig. 1 for an excerpt of our JavaScript code); without loss of generality, we assume the content is a $n \times m$ matrix of strings `C`;[2]
2. Generate a random key k_c (using TweetNaCl.js);
3. Encrypt every string `C[i][j]` with the secret key k_c and obtain `E[i][j]` (using the TweetNaCl.js symmetric encryption scheme);
4. Recover the policy as a string `P` from the UI;
5. Use the service E to encrypt k_c under `P` and obtain a ciphertext c;
6. Store $(c, \{\texttt{E[i][j]}\}_{i,j})$ in a custom XML part object in the document using the JavaScript API for Office.
7. Clear the content of the cells; e.g., our add-in replaces each of the `C[i][j]` by `#N/A`. We made this choice because each formula including an encrypted cell will then automatically display `#N/A` (cf. Fig. 4).

Note that our add-in uses hybrid encryption (i.e., data encapsulation using symmetric encryption and a public key encryption of the symmetric key), that is instead of encrypting each `C[i][j]` using the CP-ABE encryption scheme, it generates a symmetric key k_c, encrypt all the cells under k_c using a symmetric encryption scheme,

[2] Note that in our add-in, we load the *formulas* of the cells, and not the displayed text values (Fig. 1). This enables to recover cell inputs, such as `"=SUM(A1:A10)"`, that compute over cell ranges, and hence to keep the *dynamicity* of the spreadsheet.

and encrypts k_c under the CP-ABE scheme (with the public parameters). The reason is threefold: (1) encrypting/decrypting under a symmetric encryption scheme is much faster than encrypting/decrypting with the ABE scheme; (2) this enables to perform cell encryption locally rather than sending the cell content to the external service E; and (3) when encrypting with the ABE scheme, the ciphertext is significantly larger than the message (by several order of magnitudes), whereas it remains of roughly the same size when using the symmetric encryption scheme. As such, as soon as we encrypt two cells with the hybrid method, we are more efficient in time and space than encrypting both cells with the ABE scheme. We provide concrete numbers in Table 2.

As a side remark, note that most implementations using public-key cryptography today use a hybrid system. Examples include the TLS protocol, which uses a public-key mechanism for key exchange (such as Diffie-Hellman) and a symmetric-key mechanism for data encapsulation (such as AES), OpenPGP and PKCS #7 (see discussion about alternative approaches for achieving a subset of the functionality but would be viable in the short-term in Sect. 5).

3.5 Decryption

In this subsection, assume a user P_u, who has been issued one or more secret keys sk_{uv}'s for attribute sets S_{uv}'s by O, gets access to a spreadsheet with several encrypted cells as in Sect. 3.4. At load time, the add-in checks the presence of encrypted cells; if present, it displays a screen to drag and drop secret keys.

Upon drag of a key file corresponding to a CP-ABE secret key $\mathsf{sk} \in \{\mathsf{sk}_{uv}\}_v$, the decryption workflow is as follows:

1. For every group of encrypted cells as generated by Sect. 3.4, recover $(c_\ell, \{\texttt{E}_\ell\texttt{[i][j]}\}_{i,j})$ from the custom XML part object.
2. For every c_ℓ, use the service D to decrypt c_ℓ with sk, and obtain m_ℓ or $\perp$.
3. When it decrypts correctly, define $k_c = m_\ell$ and decrypt the cells $\texttt{E}_\ell\texttt{[i][j]}$ to recover $\texttt{C}_\ell\texttt{[i][j]}$.
4. Replace the content of the cell range by $\texttt{C}_\ell\texttt{[i][j]}$.

The CP-ABE scheme ensures that, if the attributes embedded in sk do not satisfy the policy associated to the ciphertext c_ℓ, P_i cannot recover the corresponding symmetric key. The symmetric encryption scheme ensures that the content of the cells remains secret to anyone that would not know the symmetric key. An important benefit of the hybrid approach is that the service D never gets to know the *content* of the cells either; instead decryption is done locally within the application itself.

Finally, note that the ABE scheme is secure against collusions. For example, assume a cell is encrypted under the policy

```
(continent == Asia) AND (continent == Europe).
```

Even if participant P_1 (resp. P_2) has the attribute `continent == Asia` (resp. `continent == Europe`), P_1 and P_2 together cannot combine their key to decrypt the ciphertext associated to the cell encryption, and therefore do not learn the cleartext content the cell.[3]

[3] Note that this policy makes sense; e.g., Russia or Turkey could be potential intended recipients of such a policy.

3.6 Expressiveness of Policies

To increase usability of our add-in, we developed a policy creation UI (Fig. 5) that allows a user to easily create policies, eventually expressible as Boolean expressions[4] with operators `AND` and `OR` of predicates of the form

```
name == value
```

for string values, and

$$\begin{array}{l} \texttt{name} == \texttt{value} \\ \texttt{name} >= \texttt{value} \\ \texttt{name} > \ \texttt{value} \\ \texttt{name} <= \texttt{value} \\ \texttt{name} < \ \texttt{value} \end{array} \qquad (1)$$

for numerical values.

For example, this allows the creation of policies of the form:

```
((continent == Europe) OR (trust >= 3)) AND (org == NATO)
            AND (key_valid_until > 1518523199)
```
,

to share data with a trusted country or a European country, part of the NATO organization, with a valid key. Indeed, the last predicate of the above policy allows for key revocation by including a numerical attribute `key_value_until` in the keys, as proposed in [13, Sect. 4.3].

Attributes in the Keys. Recall that at key generation time, CP-ABE schemes take as input a set of attributes S. In our add-in, attributes in the keys are specified by name/-value:

```
name = value.
```

When `value` is a string, we add to the set S the string `"name:value"`. When `value` is a k-bit number, we use a simple trick (already mentioned in [13, Sect. 4.3]) that decomposes the number into its bits, adding the k (string) attributes to the set S:

$$\begin{array}{c} \texttt{"name:}v_{k-1}{*}{*}\cdots{*}{*}{*}\texttt{"} \\ \vdots \\ \texttt{"name:}{*}{*}{*}\cdots{*}v_1{*}\texttt{"} \\ \texttt{"name:}{*}{*}{*}\cdots{*}{*}v_0\texttt{"} \end{array}$$

where `value` $= \sum_{i=0}^{k} v_i \cdot 2^i, v_i \in \{0, 1\}$.

[4] More precisely, it allows the creation of conjunctive normal forms (CNF).

Policies in Ciphertexts. Recall that at encryption time, CP-ABE schemes take as input access structures rather than a policy string; we therefore use the Charm policy parser [10] to convert our policies. Unfortunately, while the current policy parser of Charm explicitly parses[5] the predicates for numerical values of Eq. (1), any such predicate is replaced by the string `name` and disregards the value altogether (see the culprit function[6] on Fig. 6).

```
# convert 'attr < value' to a binary tree based on 'or' and 'and'
def parseNumConditional(s, loc, toks):
    print("print: %s" % toks)
    return BinNode(toks[0])
```

Fig. 6. Extract from the `charm/charm/toolbox/policytree.py` file in Charm that does not handle correctly numerical predicates. `toks` is a list containing three strings: the name, the operator, and the value.

In our add-in, we modified the Charm policy parser to handle the predicates of Eq. (1). Using again the bit decomposition of $\texttt{value} = \sum_{i=0}^{k} v_i \cdot 2^i, v_i \in \{0,1\}$, we use a simple tree implementing the operator (see Fig. 7 or [13, Fig. 1]) using the AND and OR operators.

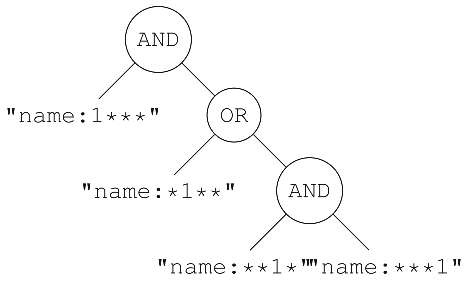


Fig. 7. Tree implementing the attribute `name >= 11`. The Boolean expression derived from the tree evaluates to true when the key contains either (a) `"name:1***"` and `"name:*1**"`; or (b) `"name:1***"`, `"name:**1*"` and `"name:***1"`; case (a) captures $\texttt{name} \geq 12$ and case (b) captures $\texttt{name} \in \{11, 15\}$.

Number of Bits. An important shortcoming of the approach described in Sect. 3.6 is that one has to be careful with the expected length of the numerical values. Indeed, assume that `name = 16`; the transformation of Sect. 3.6 yields that the key attributes set contains

[5] https://github.com/JHUISI/charm/blob/dev/charm/toolbox/policytree.py#L52.
[6] https://github.com/JHUISI/charm/blob/dev/charm/toolbox/policytree.py#L20.

```
name:1****
name:*0***
name:**0**
name:***0*
name:****0
```

The key would therefore not decrypt a ciphertext encrypted under the policy of Fig. 7 (while it should).

In our implementation, we enable specifying the number of bits of numerical attributes, defaulting to 32-bit numbers for usability. An important caveat of defaulting to 32 bits is each tree policies may contain up to 32 attributes, which impacts the performance of the online encryption with the CP-ABE scheme (see Table 1).

Table 1. Average performances of the KeyGen, Enc, and Dec operations where the ciphertext is associated to a policy `a == `k (resp., `a <=`n, resp. `a <`m) and the key is associated to an attribute `a = `k, for N-bit integers k, n, m and $k \leq n$ and $k < m$. The CP-ABE scheme is FAME instantiated in the Charm framework on a Intel Pentium CPU G4400 at 3.30 GHz.

Number of bits of the numerical values of the attributes	4	8	12	16	20	24	28	32
KeyGen (attributes: `a = `k)	31 ms	54 ms	77 ms	100 ms	123 ms	146 ms	168 ms	191 ms
Enc (policy: `a == `k)	27 ms	50 ms	74 ms	97 ms	121 ms	144 ms	168 ms	191 ms
Dec	26 ms	26 ms	26 ms	27 ms	27 ms	27 ms	27 ms	27 ms
Enc (policy: `a <=`n)	28 ms	51 ms	75 ms	98 ms	119 ms	142 ms	165 ms	186 ms
Dec	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms
Enc (policy: `a <`m)	23 ms	45 ms	70 ms	82 ms	99 ms	134 ms	160 ms	184 ms
Dec	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms

4 Evaluation and Performances

Choice of CP-ABE. As mentioned in the introduction, efficient and unbounded CP-ABE schemes based on well-established security assumptions have been proposed recently. In our add-in, we use the FAME CP-ABE scheme over the MNT224 curve introduced at CCS'2017 by Agrawal and Chase [9]. As far as we know, FAME is the most efficient CP-ABE scheme today (at the time of developing the add-ins and writing of this paper) for the encryption and decryption operations [9, Sect. 5].

Docker-Compose. Our test environment runs three Docker containers: a `nginx:latest` container that serves the add-in web page (the web service W), a `python:latest` container accessible through a REST API to access the services E and D, and finally a `nginx:latest` proxy container that listens on port 443 and redirects either to the add-in or to the backend. The Python container uses the FAME implementation of the Charm framework [10] for CP-ABE encryption and decryption.

Environment. The host is a MacBook Air (Late 2014) running macOS High Sierra 10.13.3 with a 1.7 GHz Intel Core i7. The version of the Docker engine is 17.12.0-ce and the version of Excel is 16.9 (180116).

Easy-to-Use. Our add-in is very easy to use; it only requires a user to install the add-in (e.g., via the integrated add-in store) and to specify the attributes that will be used to construct the encryption policies (e.g., using a configuration file). In particular, it *does not modify Excel in any way* and *does not require additional software* to be installed on the machine.

4.1 Encryption

Setting. We start from 5 `xlsx` documents, containing respectively 1, 10, 100, 1 000, and 10 000 cells with value `#N/A`. We report the time to encrypt those cells against four policies (see below), and the size of the resulting documents. Note that our baseline documents contains `#N/A` as text because, in Step 7 of our encryption workflow, we clear the cells by replacing their content by `#N/A`: keeping the same content displayed in the cells enables us to measure as accurately as possible the size overhead due to the encryption.[7]

Policies. We measure the performances of our encryption workflow with four policies.

P-I: `(name == value);`
The first policy is a simple policy that checks the presence of *one* attribute `name = value`, where `value` is a string, in the secret key. This is the simplest policy that can be defined.

P-II: `(name ==` n`)` where n is a 32-bit number;
The second policy is a policy that checks that the key has been created for `(name =` n`)`. Recall from Sect. 3.6 that the key will contain 32 attributes of the form `name:`$\star\star\star$`b`$\star\star\star$ where $b \in \{0, 1\}$ and a varying number of $\star$. The policy checks equality, i.e., checks that the key contains all the aforementioned attributes.

P-III: `((name1 == value1) OR (name2 == value2)) AND (name3 >`n`)` where n is a 32-bit number;
The third policy has the form of a policy created by our UI (Fig. 5). To enable decryption, a key needs to contain at least 33 attributes (the 32 attributes for the numeral values, and at least one attribute of `name1 == value1` or `name2 == value2`). Recall from Sect. 3.6 that the policy is a tree with up to 33 leafs.

P-IV: `((name1 ==` n_1`) OR (name2 ==` n_2`) OR (name3 ==` n_3`) OR (name4 ==`n_4`))`
`AND (name5 >` n_5`)`, where $n_i, i \in \{1, \ldots, 5\}$ are 32-bit numbers;
The fourth policy is a "bad" policy, in the sense that it yields a Boolean formula with up to $5 \times 32 = 160$ predicates. As we will see, this yields a large ciphertext and impacts the encryption time.

[7] Obviously, the longer the text in the cells, the larger the documents will be. We use the default secret-key authenticated encryption of TweetNaCl.js (XSalsa20-Poly1305); hence the size of each ciphertext is 16 bytes longer than the original message.

Table 2. Benchmark of the encryption workflow (Sect. 3.4) according to different policies, on 1 to $10,000$ cells.

# of cells	1	10	100	$1,000$	$10,000$
Encryption with policy **P-I**	150 ms	150 ms	163 ms	181 ms	580 ms
Encryption with policy **P-II**	397 ms	407 ms	416 ms	421 ms	837 ms
Encryption with policy **P-III**	386 ms	394 ms	410 ms	418 ms	848 ms
Encryption with policy **P-IV**	$1,410$ ms	$1,417$ ms	$1,423$ ms	$1,434$ ms	$1,614$ ms

Timings. Table 2 reports benchmarks for the encryption workflow (Sect. 3.4), that is the time it takes from the moment a user clicks 'Encrypt' and the moment the content of the cells is cleared (Step 7).

These timings illustrate the interest of hybrid encryption (Sect. 3.4): encrypting 1 or 1 000 cells takes approximately the same time. These timings also show that such an add-in is usable: encrypting $10,000$ cells with a complex policy (i.e., that involves a lot of attributes) takes about 1.5s when using Python in a Docker container on a standard laptop. Significant gains are to be expected by running an efficient implementation of the CP-ABE scheme natively on a server.

Size. Figure 8 reports the sizes of the `xlsx` documents after encrypting 1 to 10 000 cells according to the above policies. (Note that the x axis is logarithmic.)

This figure shows that, as expected from the workflow of Sect. 3.4, there is a one-time size increase corresponding to the encryption of the key under the CP-ABE scheme (difference at the leftmost of the plot between the baseline size and the sizes after encrypting one cell), and then a small overhead corresponding to the encryption of the cells. This overhead grows linearly with the number of cells encrypted. It follows that encrypting $10,000$ cells according to the complex policy **P-IV** only increases the document size by about 60 kB.

4.2 Decryption

As shown on Table 1, regardless of the policy, the decryption time is very efficient. Indeed, decrypting requires to compute 6 cryptographic *pairings* (bilinear maps) over elliptic curves, 6 multiplications in the target group, and $6I + 3$ multiplications in the input group, where I is the number of attributes used in decryption. Since multiplying in the input group is three order of magnitude faster than computing a pairing (cf. [9, Table 5.1]), the decryption time is nearly independent of the number of attributes involved. Therefore, the execution time of the decryption workflow (Sect. 3.5) amounts to the asynchronous execution of the JavaScript in the browser within the Excel software (plus network communication). In Table 3 we report average time (over 10 runs) to decrypt 100 to $1,000$ cells, encrypted as 10 sets of 100 cells according to random policies of the form **P-I**, **P-II**, **P-III**, and **P-IV**. These timings show that our (unoptimized) implementation already achieves good performance.

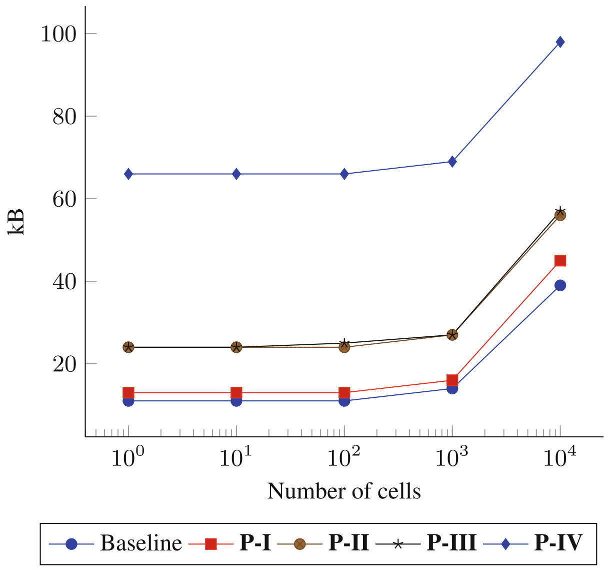


Fig. 8. Plot of the size of the `xlsx` documents after encrypting 1 to 10,000 cells according to policies **P-I** to **P-IV**.

Table 3. Benchmark of the decryption workflow (Sect. 3.5) on 10 sets of 100 encrypted cells according to a random policy of the form **P-I**, **P-II**, **P-III**, and **P-IV**.

Number of cells that can be decrypted	Average time
$1 \cdot 100$	1, 082 ms
$3 \cdot 100$	1, 295 ms
$6 \cdot 100$	1, 668 ms
$10 \cdot 100$	1, 851 ms

5 Short-Term Adoption: Policy-Based Encryption Without Collusion-Resistance via Multi-key Hybrid Encryption (Using Standardized Schemes)

The deployment of ABE in production systems, e.g., in government and commercial applications, remains limited. Currently, to the best of our knowledge, no widely deployed commercial authoring software platforms and products use ABE. The root cause of this (in the USA) may be because ABE has not been standardized yet by well known standardization bodies that develop, endorse, and maintain such national and international standards, e.g., the National Institute for Standards and Technology (NIST) in the USA. While ABE has not (yet) been standardized in the USA, there are recent efforts in that direction by the European Telecommunications Standards Institute (ETSI)[8].

Developing new cryptographic standards is a process that takes several years (as it should) due to its complexity and importance as illustrated by the ongoing[9] NIST effort to standardize post-quantum cryptography (PQC). While we acknowledge that standardizing PQC is a much larger and challenging effort compared to standardizing ABE, nevertheless, we do not expect any long-term standard to be initiated, completed, and then ratified in the next two to three years, especially if one considers a timelines similar to standardizing PQC.

A natural question then becomes *"is there a way to only utilize standard public-key/asymmetric and symmetric schemes and emulate most of the functionality and guarantees provided by ABE in some settings?"*. We sketch here a potential approach that we argue works in many enterprise settings. We stress that this is an informal treatment to argue that short-term secure selective sharing solutions may be designed and deployed, building upon the in-app cryptographically-enforced framework developed in this paper, until ABE is standardized and ready for commercial wide-scale adoption. Specifically, we focus on settings where one is not concerned about a built-in technical solution for collusion-resistance from users and insiders in the enterprise. For example, if the policy is encrypting to multiple parties, where each party by itself should be able to decrypt (i.e., an OR clause), then there is no potential (nor reason) for collusion between parties. There are a lot of settings and application where an encrypted object should be restricted to a group of employees in the enterprise, and each of them alone can, and should be able to, decrypt.

Representing Encryption Policies in Disjunctive Normal Form (DNF). While in the ABE case, policies were expressed in Conjunctive Normal Form (CNF) form (see Sect. 3.6), one can easily convert a policy into a DNF form. Whether CNF of DNF representations is preferable will depend on the application. Some functions can be succinctly represented in DNF whereas others are represented more succinctly in CNF; switching between these representations can involve an exponential increase in size [22]. We outline below techniques to use (standardized) public-key/asymmetric encryption schemes in a blackbox manner to realize AND and OR clauses. It will be up to the application to decide how to combine these into encryptions that represent DNF or CNF. It is important to stress that this encryption is only used to wrap a random short symmetric key (e.g., an AES key) as commonly used in hybrid encryption.

Encrypting to OR Clauses. The approach to encrypt an OR clause is to encrypt the symmetric key k used to encrypt the data object (m) with different public-keys, where each public-key corresponds to an attribute in the clause. For example, if the clause is a_1 OR a_2 OR a_3, where a_i corresponds to pk_i, then an encryption of data m and symmetric key k for such a clause would be $\{E^{a_1}_{pk_1}(k)||E^{a_2}_{pk_2}(k)||E^{a_3}_{pk_3}(k)||E^{s}_{k}(m)\}$, where $||$ denotes concatenation and $E^{a_i}_{pk_i}(.)$ denotes public-key/asymmetric encryption with key pk_i for attribute a_i, and $E^{s}_{k}(.)$ denotes symmetric key encryption with key s.

[8] https://www.etsi.org/newsroom/press-releases/1328-2018-08-press-etsi-releases-cryptographic-standards-for-secure-access-control.

[9] https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline.

Encrypting to AND Clauses. There are two approaches to encrypt an AND clause.

The first approach uses nested re-encryption, it performs sequential re-encryption of the symmetric key k and ciphertexts resulting from encrypting it under the different public-keys corresponding attributes in the AND clause. For example, if the AND clause is a_1 AND a_2 AND a_3, where a_i corresponds to pk_i, then encryption of data m with symmetric key k for such a clause would be $\{E^{a_3}_{pk_3}(E^{a_2}_{pk_2}(E^{a_1}_{pk_1}(k)))||E^s_k(m)\}$.

The second approach is to use additive (or another forms if t-out-of-n decryption is required) secret sharing of the symmetric key k to be encrypted, and then encrypt each share under different public keys. For example, if the AND clause is a_1 AND a_2 AND a_3, where a_i corresponds to pk_i, then encryption of data m with symmetric key k for such a clause would be $\{E^a_{pk_1}([k]_1)||E^a_{pk_2}([k]_2)||E^a_{pk_3}([k]_3)||E^s_k(m)\}$, where $[k]_i$ is (additive) share i of the key k.

The time vs space trade-off offered by the two approaches above is as follows: the first approach requires less space to store the encryption but both encryption and decryption cannot be parallelized, while in the second approach encryption and decryption can be parallelized, but would require more space.

Security. Given that the actual data is encrypted using a standard authenticated symmetric encryption scheme with a random key k (e.g., the AES-GCM authenticated encryption mode), the confidentiality of the data is ensured when k remains confidential. We argue below security of the multi-receiver key encapsulation mechanism (KEM) described above and which can be used to encrypt k for both an AND and an OR clauses.

Security of an AND Clause: The key k can be secret shared into l shares depending on the number of l literals/attributes in the AND clause. Due to the properties of secret sharing, each share of k ($[k]_i$) by itself will be a random string. Each $[k]_i$ will then be encrypted independently via the (asymmetric) public-key encryption scheme ($E^{a_i}_{pk_i}(.)$) and a different public-key pk_i. It is easy to argue by contradiction that, if such a construction is insecure, then a single application of the underlying $E^{a_i}_{pk_i}(.)$ is insecure because one could always concatenate a single such encryption with other encryptions of random messages for random public-keys and pass them to an adversary that breaks such a concatenation produced from an AND clause, thus resulting in a break of the underlying encryption scheme.

Security of an OR Clause: We note that the encryption of an OR clause is essentially a multi-receiver KEM encrypting a random symmetric key used in a data encapsulation mechanism (DEM) approach. This is the approach utilized in encrypting email in well used protocols such as S/MIME[10]. A formal security treatment of this approach is outside the scope of this paper, but we report here informally the essence of why this approach is secure. If one can break the multi-receiver use of an appropriately chosen CCA-secure public-key encryption used as a KEM mechanism (with different public-keys), then one can devise a reduction from the multi-receiver KEM used above to a single receiver KEM and thus break the security of the underlying. The reduction would generate several ciphertexts of 0 and 1 and pair them with the two given

[10] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-49.pdf.

challenge encryptions, and pass them to the multi-receiver KEM adversary to break the ones it could and then use this break to distinguish the two challenge encryptions.

Performance Overhead. We give below a high-level estimate of the encryption/decryption delay and computational overhead involved therein. We also assess the space overhead in the proposed approach.

Computational Overhead and Delay from Decryption: Assuming policies in DNF form with less than 10 OR clauses, each containing less than 10 attributes combined via an AND clause, one would have to do at most 100 public-key encryptions. As a rule of thumb, a typical public-key encryption is on the order of (or less than) several $msec$ so such encryptions and decryptions will require less than a second. We note that while opening a large MS Office document is fast, it still is a bit perceptible to the user, i.e., not instant and may take a fraction of a second or even a full second. We argue that extending this by several hundred $msec$ will be almost imperceptible to users. Finally, note that the encryptions and decryptions corresponding to the OR clauses are independent and can be easily performed in parallel. Encryptions and decryptions corresponding to AND clauses can also be parallelized if the secret sharing based technique described above is utilized.

Increase in File Size: The space overhead due to the encryption of the actual data object is minimal as it is encrypted only once using a symmetric encryption scheme (e.g., AES) and a randomly generated key. The random symmetric key is then encrypted via a public-key/asymmetric encryption several times to satisfy a policy that will contain at least two OR clauses, one for the originator of the encryption and one for the recipient of that encrypted data field. We note though that it is likely that in enterprise settings, an additional OR clause may be added to the policy so that central IT (or similar organizations) can recover encrypted content belonging to the enterprise if employees thereof leave. This clause may be such that the symmetric key is secret shared and each share is encrypted with a different key belonging to different entities in the enterprise' IT or security departments.

Limitations: One obvious limitation of the approach outlined above is that it only works for small policies, e.g., with a small number of clauses each with a few attributes. This approach also provides no collision resistance for AND clauses. We argue that if each policy only has one AND clause corresponding to the recovery term described above, then it may be acceptable because if individuals high up, and with significant privilege, are acting malicious, they could override policies and/or recover sensitive data through other means. The approach exhibits a linear overhead in the encryption size in the number of OR clauses and will require multiple public-key operations for encryption and decryption; such computationally expensive operations can be easily parallelized when both encrypting and decrypting.

6 Conclusion and Future Work

This paper investigates and addresses a major usability hurdle: the lack of selective fine-grained access control in widely deployed enterprise products, and in particular in

Microsoft Office products which are the de facto authoring and collaboration tools and often used to share information in government, commercial, and private settings.

This paper is the first step to bring ABE to widely used enterprise software products. An immediate next step will be to extend the current functionality to other products of the Office suite, such as PowerPoint. While we have developed similar add-ins and extensions to Word and Outlook, the current API for PowerPoint seems more limited. This paper motivates extensions to the JavaScript API to enable fine-grained modifications in *all* Office applications. Furthermore, it is likely that a similar approach is possible to implement for Google Workplace/Suite applications (e.g., Google Docs, Google Spreadsheet, Gmail), for which we have preliminary implementations proving the viability of developing similar add-ins. Contrary to the JavaScript API for Office, the Google add-ins framework only allows execution of server-side JavaScript code, which is a significant technical hurdle if data being encrypted (and keys) cannot (and should not) be exposed to cloud providers. We also envision developing add-ins for different web-browsers (e.g., to perform selective sharing/revealing of the content of a webpage and/or web-based applications). Future work could also investigate developing extension of the add-in(s) to support differential privacy [16].

Acknowledgments. The authors thank Tim Ellis, Ron Moore, and Karen Myers for helpful discussions and suggestions. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-15-C-4071. 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 DARPA or SSC Pacific. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. Javascript API for Office. https://dev.office.com/reference/add-ins/javascript-api-for-office
2. National Defense Authorization Act for the fiscal year 2000. https://www.congress.gov/106/plaws/publ65/PLAW-106publ65.pdf
3. Office add-ins platform overview. https://docs.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins
4. PBC library. https://crypto.stanford.edu/pbc/
5. scrypt-async. https://github.com/dchest/scrypt-async-js
6. TweetNaCl.js. https://tweetnacl.js.org/
7. Using Excel services to share pieces and parts of Excel workbooks. https://support.office.com/en-us/article/using-excel-services-to-share-pieces-and-parts-of-excel-workbooks-c9630a25-4c0a-43aa-8a93-510adb17b550
8. Zeutro LLC. http://www.zeutro.com
9. Agrawal, S., Chase, M.: FAME: fast attribute-based message encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 665–682. ACM Press, October 2017
10. Akinyele, J.A.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptographic Eng. **3**(2), 111–128 (2013). https://github.com/JHUISI/charm

11. Attrapadung, N.: Dual system encryption via doubly selective security: framework, fully secure functional encryption for regular languages, and more. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 557–577. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_31
12. Attrapadung, N.: Dual system encryption framework in prime-order groups via computational pair encodings. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 591–623. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_20
13. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society Press, May 2007
14. Chen, J., Gay, R., Wee, H.: Improved dual system ABE in prime-order groups via predicate encodings. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 595–624. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_20
15. Chen, J., Gong, J., Kowalczyk, L., Wee, H.: Unbounded ABE via bilinear entropy expansion, revisited. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 503–534. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_19
16. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_1
17. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
18. Goyal, R., Koppula, V., Waters, B.: Semi-adaptive security and bundling functionalities made generic and easy. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 361–388. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_14
19. Kowalczyk, L., Lewko, A.B.: Bilinear entropy expansion from the decisional linear assumption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 524–541. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_26
20. Lewko, A.: Tools for simulating features of composite order bilinear groups in the prime order setting. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 318–335. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_20
21. Lewko, A., Waters, B.: Unbounded HIBE and attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 547–567. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_30
22. Miltersen, P.B., Radhakrishnan, J., Wegener, I.: On converting CNF to DNF. Theor. Comput. Sci. **347**(1), 325–335 (2005)
23. Okamoto, T., Takashima, K.: Fully secure unbounded inner-product and attribute-based encryption. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 349–366. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_22
24. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 463–474. ACM Press, November 2013
25. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27

Differentially-Private "Draw and Discard" Machine Learning: Training Distributed Model from Enormous Crowds

Vasyl Pihur[1], Aleksandra Korolova[2], Frederick Liu[1], Subhash Sankuratripati[1], Moti Yung[3](✉), Dachuan Huang[1], and Ruogu Zeng[1]

[1] Snap Inc., Santa Monica, CA 90405, USA
[2] USC, Los Angeles, CA 90089, USA
[3] Columbia University, New York, NY 10027, USA
motiyung@gmail.com

Abstract. The setting of our problem is a distributed architecture facing an enormous user set, where events are repeating and evolving over time, and we want to absorb the stream of events into the model: first local model, then absorb it in the global one, and also care about user privacy. Naturally, we learn a phenomenon which happens distributedly in many places (like malware spread over smartphones, user behavior to operation and UX of an app, or other such events). To this end, we considered a configuration where the learning server is built to deal with the possibly high frequency high-volume environment in a natural distributed fashion, while taking care of statistical convergence and privacy properties of the setting as well. We propose a novel framework for privacy-preserving client-distributed machine learning. It is based on the desire to allow differential privacy guarantees in the *local* model of privacy in a way that satisfies systems constraints using high number of *asynchronous* client-server communication (i.e., not much coordination among separate clients, which is a simple communication model to implement, which in some settings already exist, i.e., in apps facing users), and provides attractive model learning properties.

We develop a generic randomized learning algorithm "Draw and Discard" because it relies on random sampling and discarding of models for load distribution and scalability, which also provides additional server-side privacy protections and improved model quality through averaging. The model is general and we show its applicability to Generalized Linear models. We analyze the statistical stability and privacy guarantees provided by our approach against faults and against several types of adversaries. We then showcase experimental results. Our framework (first reported in [28]) has been experimentally deployed in a real industrial setting. We view the result as an initial combination of ML and of distributed systems, and we believe it poses numerous directions for further developments.

Keywords: Distributed machine learning · Differential privacy · Local privacy model · High-volume distributed computing

1 Introduction

We were facing a high-frequency high-volume distributed user environment, contributing events to a learning algorithm at a server. The challenge was to absorb the volume, and assure that a distributed system is statistically stable and provides adequate level of user privacy, while tolerating faults and adversaries trying to violate user privacy.

To the above end, we propose a Machine Learning (ML) framework that is unique in that it touches on several different aspects of *practical* distributed deployment of high-frequency high-volume locally differentially private ML, all of which are equally important. These aspects include feasibility, scalability, efficiency, fault tolerance and spam protection (non manipulatability), ease of implementation in a natural distributed setting, and privacy. Ideally, they all must interplay together in a manner that enhances each other, and will not require added computational resources. From that perspective, this work is a comprehensive (already deployed!) systems-oriented one, as it is privacy, performance, and machine learning focused.

Sharing personal data that contributes to a global ML model and benefits everyone on the network—in many cases, the data collector the most– can be viewed as undesirable by many privacy sensitive users, due to distrust in the data collector or risks of subpoenas, data breaches and internal threats [20,21]. Following the deployment of RAPPOR [12], there has been an increased interest in finding ways for users to contribute data to improve services they receive, but to do so in a provably private manner, even with respect to the data collector itself [29]. This desire is often expressed by companies [17,32], presumably in part to minimize liability associated with risks and exposures associated with retention of private data of many users.

To address the privacy-utility trade-off in improving products while preserving privacy of user data even from the data collector itself (the collector itself is trusted but may be eavesdropped on by a non trusted agent at time of collection), we propose a novel client-centric distributed "Draw and Discard" machine learning framework (DDML). It provides differential privacy guarantees in the *local* model of privacy in a way that satisfies the systems constraints using *asynchronous* client-server communication (minimizing scheduling and coordination of transfer events among clients and servers). We call the framework "Draw and Discard" because it relies on randomly sampling and discarding of models.

Specifically, at the center of DDML is maintenance of k versions (or instances) of the machine learning model on a server, from which an instance is randomly selected to be sent to and updated by a client, and, subsequently, the updated instance randomly replaces one of the k instances on a server (and the replaced instance is discarded). The update is made in a differentially private manner with users' private data stored locally on their mobile devices.

We focus our analyses and experiments with DDML on the Generalized Linear Models (GLM) [25], which include regular linear and logistic regressions. GLMs provide widely-deployed and very effective solutions for many applications of non-media rich data, such as event prediction and ranking. The convex nature of GLMs makes them perfect candidates to explore client-side machine learning without having to worry about convergence issues of the more complex ML models (and, pragmatically, when it is good enough for the task, it also eases deployment). Extension of DDML to Neural Networks and other models optimized through iterative gradient updates seems doable, but are left and suggested as open issues by this initial work on the subject.

We have demonstrated through modeling, analyses, experiments, and ultimately by a practical experimental deployment that DDML provides an attractive combination of privacy, performance, model learning, and distributed systems properties (deployment feasibility, scalability, efficiency, communication overhead, and spam protection). Specifically, it offers

1. *Asynchronous and distributed training*: DDML offers continuous, lock-free, distributed and scalable training without pausing the process for averaging and updating on a server side (i.e., a type of streaming events learning as events update the learning process in an online fashion). Further, it is easy to build and maintain.
2. *Local differential privacy*: Through carefully calibrated noise in the model update step, the DDML design ensures local differential privacy [10].
3. *Privacy amplification*: Furthermore, in DDML the full model update is performed on a client and only the updated model rather than raw gradients are sent to the server. This forces an attacker to perform inference on the pre- and post-model images, while also limits the number of models on a server at any time to k.
4. *Efficient model training*: Due to the variance stabilizing property of DDML, its final model averaging and frequent model updating, DDML has superior finite sample performance relative to batching update strategies with practical batch sizes.
5. *Spam protection*: having k different instances of the same model allows, once models stabilize somewhat, to assess whether any incoming update is fraudulent or not without knowledge of the users' private data.

These properties will become clearer as we define them more precisely in the following sections. We note that our work is related to but different from what is known as "Federated Learning." We are not aware of any other, currently deployed distributed ML approaches which effectively operate in the local privacy model and use an asynchronous and distributed communication protocol in such an ongoing event streaming fashion. The novel Federated Learning [22,23] (mentioned above), another modern privacy oriented methodology invented and adopted by Google, is perhaps the closest methodology to ours which relies on server-side collecting data from clients, essentially gradient batching and averaging with relatively more infrequent model updates (on the order of a few minutes). In any event, it is good to have numerous variants with different server and client operations, and different system requirement (communication and server operation) to support a wide range of learning needs with privacy constraints in various system environments.

DDML can be viewed as offering two major contributions. First, it performs direct, *noisy* updates of model weights *on clients*, as opposed to sending *raw and exact* gradients back to the server. This change offers direct embedding of *local* differential privacy guarantees, and more importantly, requires an attacker to know the pre-image of the model (a model sent for an update to the client) that was updated to make any inference about private user data. Separation of the two critical pieces of knowledge, pre- and post-update models necessary to make any inference, *in time*, especially in a high-throughput environment with k instances being continuously updated, poses significant practical challenges for an adversary observing a stream of updates on the server side. We discuss this (preventing attackers from getting the needed data) in detail in Sect. 4.

Second, its radically different server-side architecture with the "Draw and Discard" update strategy, provides a natural way of deploying such a service in a high-performance cloud. Replication of models and treating them as equal and independent is significantly more natural for cloud deployment than a (more sequential) batch update with a locking mechanism. It is analogous to the way, for example, one would compute the total number of requests. Instead of putting each incoming request in memcache and then having a batching job, say every second, to add to the single total counter, the recommended way is to have K counters and increment each one at random with the final result being the total sum of the K counters. Asynchronous updates are almost always preferred to synchronous ones when scalability is paramount. This crucial property is exactly what DDML achieves with its server-side architecture.

Beyond these two major considerations, DDML offers a completely lock-free asynchronous, and thus, more efficient (relatively coordination-free), communication between the server and clients, which is an absolute must if one is developing in a massively distributed environment [7], as well as a straightforward distributed way to prevent model spamming by malicious actors, without sacrificing user privacy. (Note: an asynchronous communication may also allow future extension of anonymous routing hiding clients from server, effectively shuffling responses to the server).

Deployment: We have implemented DDML and deployed it experimentally (with billions of daily users accumulated over one month), and successfully trained numerous ML models. Our applications focus on ranking items, from a few dozen to several thousands, as well as security oriented services, such as predicting how likely it is that a received URL is "phishy." Our largest models contain $\approx 50,000$ weights in size, and we find $k = 20$ models to be the right trade-off between efficiency and scale, to avoid the "hotspotting" issue. Currently, at peak times for several different applications, we receive 200 model updates per second.

Organization: The paper is organized as follows: Sect. 2 reviews differential privacy and related work. Section 3 presents a detailed overview of our framework and its features, including the variance stabilizing property in Sect. 3.4. Section 4 introduces our modeling of possible adversaries and discusses DDML's privacy properties with respect to them. We conclude with a discussion of limitations, alternatives and avenues for future work in Sect. 6.

2 Preliminaries and Related Work

Differential privacy (DP) [10] has become the de-facto standard for privacy-preserving data analysis [9, 11, 13].

A randomized algorithm A is (ϵ, δ) differentially private if for all databases D and D' differing in one user's data, the following inequality is satisfied for all possible sets of outputs $Y \subseteq Range(A)$:

$$Pr[A(D) \in Y] \leq \exp(\epsilon) Pr[A(D') \in Y] + \delta$$

The parameter ϵ is called the privacy loss or privacy budget of the algorithm [26], and measures the increase in risk due to choosing to participate in the DP data collection. The variant of DP when $\delta = 0$ is the strongest possible differential privacy variant called *pure* differential privacy; whereas $\delta > 0$ allows differential privacy to fail with small probability and is called *approximate* differential privacy.

ML in the Trusted-Curator Model: Most prior work for differentially private machine learning assumes a trusted-curator model, where the data is first collected by the company and only then a privacy-preserving computation is run on it [1,6,27,31]. The trusted-curator model is less than ideal from the user privacy perspective, as it does not provide privacy from the company collecting the data, and, in particular, leaves user data fully vulnerable to security breaches, subpoenas and malicious employees. Furthermore, even in the case of the trusted curator model, differentially private deep learning that achieves good utility with reasonable privacy parameters has been an elusive goal [1,24,30]. For example, the work of [1] performs well on the MNIST data but struggles utility-wise on CIFAR for reasonable privacy parameters.

ML in the Local Model: The pioneering work of RAPPOR [12] for industry deployment, has been followed by several recent efforts to deploy DP in the local model, i.e., guarantee DP to the user before the data reaches the collector. Privacy in the local model is more desirable from the user's perspective [17,20,29,32], as in that case the user does not have to trust the data collector or the data being subject to internal or external threats to the data collector.

Since the focus on differentially private computations in the local model is recent, most, if not all, efforts to date have been limited to learning aggregate statistics, rather than training more complex machine learning models [2–5,14]. There are also numerous results on the so-called sample complexity for the local model, showing that the number of data points needed to achieve comparable accuracy is significantly higher in the local model than in the trusted curator model [18].

DDML can be considered an extension of the existing literature on locally private learning. In particular, it supplements private histogram collection of RAPPOR [12] and learning simple associations [14] by allowing estimation of arbitrary conditional expectations. While RAPPOR allows estimating marginal and joint distributions of categorical variables, DDML provides a principled framework for estimating conditional distributions in a privacy-preserving manner. For example, one can estimate the average value of Y given p features $X_1, \ldots, X_p$ by fitting a regular linear model described by

$$E(Y) = b_0 + b_1 * X_1 + \ldots + b_p * X_p.$$

3 Draw and Discard Machine Learning

In this section, we present our "Draw and Discard" machine learning framework characterized by its two major components: client-side noise addition and "Draw and Discard" server architecture. Together, these contribute to strong differential privacy guarantees for client data while supporting efficient, in terms of model training, client-server data consumption. At the heart of DDML is the server-side idea of maintaining and randomly updating one of the k model instances. This architecture presents a number of interesting properties and contributes to many aspects of the framework's scalability, privacy, and spam and abuse protections. We give a brief overview of GLMs and fully describe DDML client and server architectures next.

3.1 GLMs

In GLMs [25], the outcome or response variable Y is assumed to be generated from a particular distribution in the exponential family that includes normal (regular linear model), binomial (logistic regression) and Poisson (Poisson regression) distributions, among many others. Mathematically, GLMs model the relationship between response Y and features $X_1, \ldots, X_p$ through a link function g, whose exact form depends on the assumed distribution:

$$E(Y) = g^{-1}(b_0 + b_1 * X_1 + \ldots + b_p * X_p) \tag{1}$$

To train GLM models on clients, we use Gradient Descent (GD) for maximum likelihood estimation, as discussed below.

3.2 DDML Client-Side Update

GD is a widely used iterative procedure for minimizing an objective function

$$Q(B) = \frac{1}{N} \sum_{s=1}^{N} Q_s(B), \tag{2}$$

where $B = \{b_1, \ldots, b_p\}$ is the vector of weights to be estimated and Q_s is a functional component associated with the sth observation. Traditional optimization techniques require differentiating $Q(B)$, which, in turn, requires access to all N data points at once. GD approximates the gradient $\Delta Q(B)$ with $\Delta Q_s(B)$, computed on a small batch of N_c observations available on a single client

$$\Delta Q_{N_c}(B) = \frac{1}{N_c} \sum_{s=1}^{N_c} \Delta Q_s(B). \tag{3}$$

To provide local privacy by adding random Laplace noise, a differentially-private GD (DP-GD) update step is performed on a client using the N_c observations stored locally

$$B^{t+1} = B^t - \gamma \Delta Q_{N_c}(B^t) + L\left(0, \frac{\Delta f}{\epsilon}\right), \tag{4}$$

where γ is a learning rate and $L\left(0, \frac{\Delta f}{\epsilon}\right)$ denotes the Laplace distribution with mean 0 and scale $\frac{\Delta f}{\epsilon}$. Δf is called sensitivity in the differential privacy literature and ϵ is the privacy budget [11].

For GLMs, assuming all features X_i are normalized to the $[0, 1]$ interval and the average gradients $\frac{1}{N_c} \sum_i^{N_c} (\hat{Y}_i - Y_i) X_i$ are clipped to $[-1, 1]$ (indicated by $\|A\|_{[-1,1]}$), the differentially-private update step becomes

$$B^{t+1} = B^t - \gamma \left\| \frac{1}{N^c} \sum_{i=1}^{N_c} (\hat{Y}_i - Y_i) X_i \right\|_{[-1,1]} + L\left(0, \frac{2\gamma}{\epsilon}\right).$$

Algorithm 1. DDML Algorithm for GLMs (client side). Parameters:
Y - response value, $\hat{Y}$ - predicted value.
X - feature vector, B - a set of model weights.
γ - learning rate, ϵ - privacy budget.
$L(\mu, s)$ - Laplace distribution with mean μ and scale s.

```
Normalize response $Y$ and features $X$ to $[0, 1]$
If $N_c > 0$, request model $B^t$ from the server
Compute clipped average gradient: $\|G\|_{[-1,1]} = \|\frac{1}{N^c}\sum_{i=1}^{N_c}(\hat{Y}_i - Y_i)X_i\|_{[-1,1]}$
Update model: $B^{t+1} = B^t - \gamma\|G\|_{[-1,1]} + L\left(0, \frac{2\gamma}{\epsilon}\right)$
Return model $B^{t+1}$ to the server
```

Algorithm 2. DDML Algorithm (server side). Parameters:
k - the number of models, B - a set of model weights.

```
Initialize $k$ models
for each requested model update $t$ do
    Pick a random model instance $B^t$
    Send $B^t$ to a client
    Receive updated $B^{t+1}$ from a client
    Replace a random instance of the $k$ models with $B^{t+1}$
end for
Prediction: average $k$ model instances
```

Here, $\hat{Y}$ is the predicted value of Y given a feature vector X and the model B^t. Clipping gradients to $[-1, 1]$ ensures that the sensitivity of the update step is at most 2γ. For logistic regression, if all features are normalized to $[0, 1]$, no gradient clipping is necessary.

DDML client-side architecture is shown in Algorithm 1.

3.3 DDML Server-Side Draw and Discard

While maintaining the k model instances on a server (k versions of the same model with slightly different weights), we randomly "draw" one of the k instances, update it on a client and put it back into the queue by "discarding" an instance uniformly at random. With probability $\frac{1}{k}$, we will replace the same instance, while with probability $\frac{k-1}{k}$, we will replace a different one.[1]

This seemingly simple scheme has significant practical implications for performance, quality, privacy, and anti-spam, which we discuss in Sect. 3.4.

DDML server-side architecture is shown in Algorithm 2.

Model Initialization. We initialize our k models randomly from a normal distribution with means $b_0^0, \ldots, b_p^0$, which are usually taken to be 0 in the absence of better starting values and variance $\sigma_k^2 = \frac{k}{2}\sigma^2$, where σ^2 is the variance of the Laplace noise added on a client side.

[1] This is only approximately correct, since in a high-throughput environment, another client request could have updated the same model in the meantime.

Because of the variance-stabilizing property (to be discussed in detail in Sect. 3.4), σ_k^2 will remain the same in expectation even after a large number of updates. It is crucial for our spam detection solution that this initialization happens correctly and the right amount of initial noise is added to calibrate the update step on a client with the variance of the k instances on the server side.

Model Averaging. We average weights from all k instances for final predictions. Of course, depending on application, another way for using k versions of the same model could be preferred, such as averaging predicted values from each instance, for example.

3.4 Properties and Features of DDML

We now describe properties of DDML that distinguish it from existing solutions and make it feasible and scalable for practical deployments.

Variance-Stabilizing Property of DDML. Having introduced an additional source of variation due to having k model instances, an intra-model variance, it is important to understand its nature and magnitude, especially relative to the variance of the noise added on the client through the Laplace mechanism. It is also of interest to understand how it changes over time. One of the remarkable properties of the "Draw and Discard" algorithm with $k > 1$ is its variance-stabilizing property. We prove in Theorem 1 that the expected intra-model variance of the k instances is equal to $\frac{k}{2}\sigma^2$ and remains unchanged after an infinite number of updates when adding noise with mean 0 and variance σ^2.

Theorem 1. *Let $B = (B_1, \ldots, B_k)$ be a vector of k random variables (weights) with mean μ and variance $\frac{k}{2}\sigma^2$. Selecting one of the weights at random, adding noise with mean 0 and variance σ^2 and putting it back with replacement does not change the expected intra-model variance of B (i.e., weights remain distributed with variance $\frac{k}{2}\sigma^2$).*

The intuition behind this theorem is that with probability $\frac{1}{k}$, we replace the same model, which increases the variance of the k instances. This increase, however, is exactly offset by the decrease in variance due to the cases when we replace a different model with probability of $\frac{k-1}{k}$ because original and updated models are highly correlated. The full proof can be found in the Appendix.

DDML, due to its "Draw and Discard" update strategy, dissipates the additional intra-model variability through random model discarding which is particularly important when the model has converged and the contractive pull of the GD is either small or non-existent, at a time when we continue training the model and adding the Laplace noise on the client.

Asynchronous Learning. Maintaining k model instances allows for a *scalable*, *simple* and *asynchronous* model, updating with hundreds or thousands of update requests per second. It is trivial to implement, relies on its inherent randomness for load distribution, and requires no locking mechanism that would pause the learning process to aggregate or summarize results at any given time.

Privacy. Due to random sampling of model instances, the DDML server architecture uniquely contributes to privacy guarantees as will be discussed in Sect. 4. Specifically, by keeping *only* the last k models from clients, discarding models at random, and avoiding server-side data batching, the DDML fulfills the goal of keeping as little data as possible on the server. Through a nuanced modeling of possible adversaries (Sect. 4.1) and corresponding privacy analyses, DDML is able not only to provide privacy guarantees in the local model, but also improve these privacy guarantees against weaker but realistic adversaries.

Ability to Prevent Spam Without Sacrificing Privacy. The k instances are instrumental in spam and abuse prevention, which is a real and ubiquitous pain point in all major client-server deployments. Nothing prevents a client from sending an arbitrary model back to the server. We could keep track of which original instance was sent to each client; however, this would negate the server-side privacy benefits and pose implementation challenges due to asynchronicity. In DDML, having k replicates of each weight b_i allows us to compute their means μ_i and standard deviations σ_i and assess whether the updated model is consistent with these weight distributions (testing whether the updated value is within $[\mu - t\sigma, \mu + t\sigma]$), removing the need to make trade-offs between privacy and anti-abuse.

3.5 Parameter Tuning and Clipping

Choosing the right learning rate γ is critical for model convergence. If chosen too small, the learning process proceeds too slowly, while selecting a rate too large can lead to oscillating jumps around the true minimum. We recommend trying several values in parallel and evaluating model performance to select the best one. In the future, we plan to explore adaptive learning rate methods in which we systematically decrease γ (and, therefore, add noise) as the model converges.

By clipping gradients to a $[-1, 1]$ range, we ensure that the sensitivity of our update is 2γ. In practice, the vast majority of gradients, especially as the model becomes sufficiently accurate, are much smaller in absolute terms and, thus, could be clipped more aggressively. Clipping to a $[-0.1, 0.1]$ range would reduce sensitivity by a factor of 10 to $\gamma/5$.

4 Privacy of DDML

We now discuss differential privacy guarantees provided by DDML. Our analyses are with respect to feature-level differential privacy, as discussed in Sect. 6, but they can be easily extended to model-level privacy by scaling up the noise by the number of features or by adjusting the norm of the gradient in Algorithm 1.

4.1 Adversary Modeling

The main innovation of our work with respect to privacy analyses comes from more nuanced modeling of heterogeneous adversaries, and the demonstration that the privacy guarantees a client obtains against the strongest possible adversary operating in the local model of privacy are strengthened by DDML against weaker but realistic adversaries.

Our work introduces and considers three kinds of adversaries, differing in the power of their capabilities:

I (Channel Listener): is able to observe the communication channel between the client and the server in real time and, therefore, sees both the model instance sent to the client and the updated model instance sent from the client to the server.

II (Internal Threat): is able to observe the models on the server at a given point in time; i.e., this adversary can see model instances 1 through k but lacks the knowledge of which of the k instances was the pre-image for the latest model update due to lack of visibility into the communication channel.

III (Opportunistic Threat): can observe a model instance at a random time, but has no knowledge of what was the state of the model weights over the last Tk updates, i.e., this adversary can, for example, see models at regular time intervals Tk which is much larger than 1. Clients themselves are such threats as they periodically receive a model to update.

The first adversary is the most powerful, and the privacy guarantees we provide against this adversary (Sect. 4.2) correspond to the local model of privacy commonly considered by the differential privacy community (Sect. 2).

The second adversary is modeling ability to access the k model instances from within the entity running DDML. It is reasonable to assume that such an adversary may be able to obtain several snapshots of the models, though it will be technically unfeasible and/or prohibitively costly to obtain snapshots at the granularity that allows observation of k models before and after every single update.

The third type of adversary is the weakest and also the most common. Occasional access to models allows attackers to obtain a snapshot of k model instances (in a case of an internal threat) or just a single model instance (in a case of a client who receives a model for an update) after a reasonably large number of updates Tk. Because Tk independent noise additions have occurred in the meantime, each model instance has received an expected T updates and therefore, T independent noise additions after a particular user's update. Every user benefits from this additional noise to a different degree, depending on the order in which their data was ingested, and, in expectation and with high probability, enjoys significantly stronger differential privacy guarantees against this adversary than those of the local model, as will be shown in Sect. 4.4.

4.2 Privacy Against Channel Listener (Adversary I)

DDML guarantees ϵ-differential privacy against adversary I. The claim follows directly [11] from our choice of the scale of Laplace noise in the client-side Algorithm 1 and the observation that clipping the gradient in Algorithm 1 ensures sensitivity of at most 2γ. (It is possible to replace the Laplace noise used in the client-side Algorithm 1 with Gaussian. In that case, the variance of the Gaussian noise would need to be calibrated according to Lemma 1 from [19] or Theorem A.1 of [11].)

In practice, the channel is going to be encrypted (under TLS, say) so one can view the local differential privacy as a hedging against breaks of TLS or temporary leak at the server. The above is assurance against one-time viewer of the channel; if we expect more channel breaks, say t, then each contribution should have ϵ/t-differential privacy (handling the privacy budget). When learning from a huge crowd, we anyway expect only one contribution from each client (on the average) to be reflected in the state (where each client's input affects at most one of the k existing models at any given time. (Note that more refined analysis of other cases is certainly left open for future work).

4.3 Privacy Against Internal Threat (Adversary II)

DDML offers the same ϵ-differential privacy guarantees as against adversary I. However, there are several practical considerations that amplify client privacy in a more informal manner. As k gets larger, there are more and more plausible pre-images for any updated model. This uncertainty introduced by updating the model fully on the client, forces an adversary II to perform inference on what the potential update was. Given the variance stabilization Theorem 1 and the contractive property of the GD, the k models will be relatively similar and, therefore, potential candidates for the pre-image. The benefits of this additional privacy protection do diminish with the size of the model where the curse of dimensionality takes over. Nevertheless, even having to do actual work can be a strong barrier for a casually curious attacker.

4.4 Privacy Against Opportunistic Threat (Adversary III)

Finally, we analyze the privacy guarantees DDML provides against adversary III – the one that is able to inspect a random model instance out of the k models after a user of interest to the adversary has submitted their model instance and an expected T updates to that model instance have occurred since. Note that in practice, the adversary may have an estimate of T, but not know it precisely, as it is difficult to measure how many updates have occurred to a model instance in a distributed system serving millions of clients such as DDML.

The privacy amplification against this adversary will come from two sources – from the "discard" step, in that it contributes to the possibility that the model the user contributed to is discarded in the long-term and from the accumulation of the noise, in that with each model update, additional random noise is added to it, which contributes to further privacy protection for the user whose update has occurred in preceding steps. The analysis of the privacy amplification due to the "discard" step is presented in Lemma 1 and it can be shown that only $\frac{1}{k}$ updates will survive (be still contributing) to any of the models as T becomes large.

Lemma 1. *DDML discards* $1 - \frac{1}{k}$ *of updates in the long-term.*

Proof can be found in the Appendix.

Analysis on the amount of noise accumulation over time is presented in Theorem 2, which results in the *overall* $k\sqrt{T}$ expected privacy amplification against adversary III compared to adversary I. The full proof can be found in the Appendix.

Theorem 2. *With high probability, DDML guarantees a user (ϵ_T, δ_T)-differential privacy against adversary III, where*

$$\epsilon_T = \frac{\epsilon}{\sqrt{2T}} \sqrt{\ln\left(\frac{1}{2\delta_T}\right)}$$

and δ_T is an arbitrary small constant (typically chosen as O(1/ number of users)). T is the number of updates made to the model instance between when a user submitted his instance update and when the adversary observes the instances. The statement holds if T is sufficiently large.

5 Real World Applications

We instrumented DDML on all Android devices at SNAP, a company with hundreds of millions of monthly active users. Since we do not store or associate user IDs with model updates, it is impossible to know exactly how many unique users have contributed to each model, but it is definitely in hundreds of millions. We rely on a pull model where a device collects relevant data and, if any collected, pulls the model from a server at most every 6 h, updates it and sends it back, after which the data on the device is deleted (so learning locally is always from fresh data). We also limit the amount of records that can be stored on a given device at any point in time.

One of the most common abuse vectors is phishing user credentials. When malicious accounts manage to befriend legitimate ones or some accounts get hacked, they begin to send phishing url's to users within their social circle. Even if only a small fraction of users takes the bait, the cycle of abuse is perpetuated indefinitely. We rely on Google Safe Browsing [16] and internally maintained lists of known phishy URLs to prevent these kinds of attacks. However, the process of discovering *novel* malicious URLs depends on customer ops reports which is slow, manual and opportunistic.

We implemented a DDML logistic regression model for predicting the likelihood of a URL being phishy. Our features come from both the URL itself (different parts of the URL, such as domain, subdomain, host, path, query parameters), as well as the page content. We consider features related to character distribution, special characters $(?, \&, \ldots)$, lengths, language, particular keywords, etc. For content page features, we include features related to major page component, such as iframes, input boxes, password boxes, images, scripts, page size, readability and many others.

For the model used, we received 1,730,624,961 model updates. The number of weights is 387. We use $k = 20$, $\epsilon = \log(32)$ and learning rate $\gamma = 0.001$. We achieved 89.2% recall and 15.7% precision for this application. Overall, our recall is very good, while precision remains relatively low because of the inherent complexity of the adversarial setting in which attackers constantly change their approaches making it hard to detect new types of URLs that have not been previously seen. Our online model allowed us to increase discovery of new phishing URLs by an order of magnitude once it was launched.

6 Discussion

Having deployed DDML at scale with hundreds of million daily active clients, we realized how critical a well-designed server-side architecture was to the client-side learning process. Due to the symmetric nature of draws and discards, with the number of reads equaling the number of writes, there must be sufficient redundancy in place to scale our serving infrastructure. k model instances offer, besides increased privacy, an incredibly scalable and asynchronous solution to client-server communication.

One can easily make an argument that replacing model instances at random is "wasteful" from the model training perspective (though, we have an abundance of events in our applications). The claim, nevertheless, is partially true (and further analysis of tradeoffs is another issue for further investigations). However, so is setting the wrong learning rate, mismanaging the server-side batch size, etc. We are never perfect in utilizing our data in the absolute sense even before moving to a distributed ML setting (complicated dynamic distributed system has these properties). There, things only become more complicated from a learning perspective and it is not unreasonable to see additional performance sacrifices. If your architecture can support only 10 writes per second, for example, and your overall traffic is 100 write requests per second, you will not be able to perfectly utilize all your available data in a sequential updating scheme. Trade-offs must be made. The focus, methodologically speaking, should be not on what we are losing because we must lose something, but what we are gaining in exchange. By making a small sacrifice in performance by introducing k instances, we gain scalability, ease of implementation, spam detection, and additional privacy. The only question (yet another open issue) is whether we are trading off these properties efficiently.

We offer *feature-level* local differential privacy and, therefore, in a situation when features are correlated, the privacy loss scales with the number of features. In principle, if one would like to achieve model-level privacy, one needs to scale the noise up according to the number of features included in the model or alternatively perturb the whole feature vector as in [8]. Applications of differential privacy to very high-dimensional data, particularly, in the local model, have not yet been adopted in practice. In theoretical work, the distinction is often mentioned, but the choice is left to industry practitioners. We believe that in practice, feature-level privacy combined with limited server-side model retention is sufficient to protect the privacy of our clients against most realistic adversaries. Note, also, that we have shown that due to the discard operation, the constantly evolving model loses with some probability dependency on user's contribution which certainly adds to privacy.

We certainly hope the above discussion will generate further investigations in the model, which has demonstrated to be very useful in the large scale distributed events case. Improved more refined analysis of cases is also open, and various architectural-algorithmic tradeoff-s are worth investigation as well.

Appendix

A Variance Stabilization Proof and Other Proofs

Theorem 1 Let $B = (B_1, \ldots, B_k)$ be a vector of k random variables (weights) with mean μ and variance $\frac{k}{2}\sigma^2$. Selecting one of the weights at random, adding noise with mean 0 and variance σ^2 and putting it back with replacement does not change the expected intra-model variance of B (i.e., weights remain distributed with variance $\frac{k}{2}\sigma^2$).

Proof. We use the Law of Total Variance

$$V(B) = E(V(B|X)) + V(E(B|X))$$

three times as we think of B as a mixture of mixtures. In the first mixture, the expectation is taken over the distribution of whether the same or different model was updated ($X \in \{j \to j, j \to j'\}$). In the inner mixture, the expectation is taken over the partitioning of the k weights themselves (Z).

Total variance, therefore, is equal to

$$\begin{aligned} V(B) = &\frac{1}{k}V(B|j \to j) + \frac{k-1}{k}V(B|j \to j') \\ &+V(E(B|X)). \end{aligned}$$

Because we add noise with mean 0, in either case, the mean of B does not change, so $V(E(B|X)) = 0$.

Replacing the same weight partitions the k weights into two sets, a single weight updated and the rest of the weights. Partition means do not change, and the variance increases due to added noise in the first partition (mixture component). Thus, $V(B|j \to j)$ is given as

$$\begin{aligned} V(B|j \to j) &= \frac{1}{k}\left(\frac{k}{2}+1\right)\sigma^2 + \frac{k-1}{k}\frac{k}{2}\sigma^2 \\ &= \left(\frac{1}{2}+\frac{1}{k}+\frac{k-1}{2}\right)\sigma^2 \\ &= \left(\frac{k}{2}+\frac{1}{k}\right)\sigma^2. \end{aligned}$$

Replacing a different weight partitions the weights space into 2 and $(k-2)$ subsets. Unlike in the first case, $V(E(B|Z))$ is non-zero due to a single weight essentially replicated twice in the first partition (B_1 has a mean of 0, but B_2 has a mean of B_1). After the update, the overall mean of B under Z becomes

$$\mu_Z = \frac{2\mu_1 + (k-2)\mu}{k},$$

where μ_1 is the mean of the model selected and model replaced and has a distribution with mean μ and variance $\frac{k}{2}\sigma^2$. Note that the mean of B over sampling in X is still 0 as $E(B_1) = 0$ over X.

Then $V(B|j \to j')$

$$\begin{aligned}
&= \frac{2}{k}\frac{1}{2}\sigma^2 + \frac{k-2}{k}\frac{k}{2}\sigma^2 \\
&\quad + \frac{2E[(\mu_1 - \mu_{B_i})^2] + (k-2)E[(\mu - \mu_{B_i})^2]}{k-1} \\
&= \frac{1}{k}\sigma^2 + \frac{k-2}{2}\sigma^2 \\
&\quad + \frac{2}{k-1}\Big(\frac{k-2}{k}\Big)^2 E[(\mu_1 - \mu)^2] \\
&\quad + \frac{k-2}{k-1}\Big(\frac{2}{k}\Big)^2 E[(\mu_1 - \mu)^2] \\
&= \frac{1}{k}\sigma^2 + \frac{k-2}{2}\sigma^2 + \frac{2}{k-1}\left(\frac{k-2}{k}\right)^2 \frac{k}{2}\sigma^2 \\
&\quad + \frac{k-2}{k-1}\left(\frac{2}{k}\right)^2 \frac{k}{2}\sigma^2 \\
&= \left(\frac{1}{k} + \frac{k-2}{2} + \frac{(k-2)^2}{k(k-1)} + \frac{2(k-2)}{k(k-1)}\right)\sigma^2 \\
&= \frac{2k - 2 + k(k-2)(k-1) + 2(k-2)^2 + 4k - 8}{2k(k-1)}\sigma^2 \\
&= \frac{k^3 - k^2 - 2}{2k(k-1)}\sigma^2.
\end{aligned}$$

Note that the variance component must be computed with $k-1$ and not k because of the finite nature of k in this case.

Putting it all together,

$$\begin{aligned}
V(B) &= \frac{1}{k}V(B|j \to j) + \frac{k-1}{k}V(B|j \to j') \\
&= \frac{1}{k}\left(\frac{k}{2} + \frac{1}{k}\right)\sigma^2 + \frac{k-1}{k}\frac{k^3 - k^2 - 2}{2k(k-1)}\sigma^2 \\
&= \left(\frac{1}{2} + \frac{1}{k^2} + \frac{k^3 - k^2 - 2}{2k^2}\right)\sigma^2 \\
&= \frac{k}{2}\sigma^2.
\end{aligned}$$

Lemma 1 DDML discards $1 - \frac{1}{k}$ updates long-term.

Proof. Consider a Markov process on the states $0, 1, \ldots, k$, where each state represents the number of models in which a particular update can be found. Denote by p_i - the probability to go from i to $i-1$ or $i+1$. In DDML, $p_i = \frac{(k-i)i}{k^2}$, for $1 \le i \le k-1$, and $p_0 = 0, p_k = 1$.

Let q_i be the probability of eventually ending up at state 0 if you start in state i. By the set-up of DDML, for $1 < i < k-1$ we have:
$q_i = p_i q_{i-1} + p_i q_{i+1} + (1 - 2p_i)q_i$, or $2q_i = q_{i-1} + q_{i+1}$.

We also know that

$$q_0 = 1, p_0 = 0, q_k = 0, p_k = 1,$$
$$q_1 = p_1 q_0 + p_1 q_2 + (1 - 2p_1) q_1,$$
$$q_{k-1} = p_{k-1} q_{k-2} + p_{k-1} q_k + (1 - 2p_{k-1}) q_{k-1}$$

Summing equations for $1 \leq i \leq k-1$ we have $2q_1 + \cdots + 2q_{k-1} = q_0 + q_1 + 2(q_2 + \cdots + q_{k-2}) + q_{k-1} + q_k$ Simplifying: $q_1 + q_{k-1} = q_0 + q_k$ or

$$q_1 + q_{k-1} = 1 \tag{5}$$

On the other hand,

$$q_{k-2} = 2q_{k-1},$$
$$q_{k-3} = 2q_{k-2} - q_{k-1} = 3q_{k-1}$$
$$q_{k-4} = 2q_{k-3} - q_{k-2} = 4q_{k-1}$$
$$\ldots$$

$$q_1 = 2q_2 - q_3 = (k-1) q_{k-1} \tag{6}$$

Combining (5) and (6), we have $(k-1)q_{k-1} + q_{k-1} = 1$ or $q_{k-1} = \frac{1}{k}$ and $q_1 = 1 - \frac{1}{k}$. Since DDML is set-up that each particular contribution is initially in state 1, this completes the proof.

Theorem 2 With high probability, DDML guarantees a user (ϵ_T, δ_T)-differential privacy against adversary III, where $\epsilon_T = \frac{\epsilon}{\sqrt{2T}} \sqrt{\ln\left(\frac{1}{2\delta_T}\right)}$ and δ_T is an arbitrary small constant (typically chosen as O(1/ number of users)). T is the number of updates made to the model instance between when a user submitted his instance update and when the adversary observes the instances. The statement holds if T is sufficiently large.

Proof. We rely on the result from concurrent and independent work of [15] obtained in a different context to analyze the privacy amplification in this case. Specifically, their result states that for *any* contractive noisy process, privacy amplification is no worse than that for the *identity* contraction, which we analyze below.

The sum of T random variables drawn independently from the Laplace distribution with mean 0 will tend toward a normal distribution for sufficiently large T, by the Central Limit Theorem. In DDML's case with Laplace noise, the variance of each random variable is $\frac{8\gamma^2}{\epsilon^2}$, therefore, if we assume that the adversary observes the model instance after T updates to it, the variance of the noise added will be $T \cdot \frac{8\gamma^2}{\epsilon^2}$. This corresponds to Gaussian with scale $\sigma = \frac{2\sqrt{2T}\gamma}{\epsilon}$.

Lemma 1 from [19] states that for points in p-dimensional space that differ by at most w in ℓ_2, addition of noise drawn from $N^p(0, \sigma_T^2)$, where $\sigma_T \geq w \frac{\sqrt{2\left(\ln\left(\frac{1}{2\delta_T}\right) + \epsilon_T\right)}}{\epsilon_T}$ and $\delta_T < \frac{1}{2}$ ensures (ϵ_T, δ_T) differential privacy. We use the result of Lemma 1 from [19], rather than the more commonly referenced result from Theorem A.1 of [11], because the latter result holds only for $\epsilon_T \leq 1$, which is not the privacy loss used in most practical applications.

We now ask the question: what approximate differential privacy guarantee is achieved by DDML against adversary III? To answer this, fix a desired level of δ_T and use the approximation obtained from the Central Limit theorem to solve for the ϵ_T.

$$\frac{2\sqrt{2T}\gamma}{\epsilon} \geq w\frac{\sqrt{2\left(\ln\left(\frac{1}{2\delta_T}\right)+\epsilon_T\right)}}{\epsilon_T}$$

$$T\cdot\frac{8\gamma^2}{\epsilon^2} \geq w^2\frac{2\left(\ln\left(\frac{1}{2\delta_T}\right)+\epsilon_T\right)}{\epsilon_T^2}$$

$$T\cdot\frac{4\gamma^2}{\epsilon^2}\cdot\epsilon_T^2 - w^2\epsilon_T - w^2\ln\left(\frac{1}{2\delta_T}\right) \geq 0$$

Solving the quadratic inequality, we have:

$$D = w^4 + 4T\cdot\frac{4\gamma^2}{\epsilon^2}w^2\ln\left(\frac{1}{2\delta_T}\right)$$

$$\epsilon_T \geq \frac{w^2+\sqrt{D}}{2T\cdot\frac{4\gamma^2}{\epsilon^2}} = \frac{\epsilon^2 w^2}{8\gamma^2 T}\left[1+\sqrt{1+16T\frac{\gamma^2}{\epsilon^2 w^2}\ln\left(\frac{1}{2\delta_T}\right)}\right]$$

For large T, the additive term of 1 under the square root is negligible, so we have:

$$\epsilon_T \approx \frac{\epsilon^2 w^2}{8\gamma^2 T}4\sqrt{T}\frac{\gamma}{\epsilon w}\sqrt{\ln\left(\frac{1}{2\delta_T}\right)} = \frac{\epsilon w}{2\gamma\sqrt{T}}\sqrt{\ln\left(\frac{1}{2\delta_T}\right)}$$

In DDML, $w = \sqrt{2}\gamma$, therefore,

$$\epsilon_T \approx \frac{\epsilon}{\sqrt{2T}}\sqrt{\ln\left(\frac{1}{2\delta_T}\right)}$$

References

1. Abadi, M., et al.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016), pp. 308–318 (2016). http://doi.acm.org/10.1145/2976749.2978318
2. Apple Differential Privacy Team: Learning with Privacy at Scale, vol. 1. Apple Mach. Learn. J. (8) (2017). https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html
3. Bassily, R., Nissim, K., Stemmer, U., Thakurta, A.G.: Practical locally private heavy hitters. In: Advances in Neural Information Processing Systems, pp. 2285–2293 (2017)
4. Bassily, R., Smith, A.: Local, private, efficient protocols for succinct histograms. In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, pp. 127–135. ACM (2015)
5. Bun, M., Nelson, J., Stemmer, U.: Heavy hitters and the structure of local privacy. arXiv preprint arXiv:1711.04740 (2017)

6. Chaudhuri, K., Monteleoni, C.: Privacy-preserving logistic regression. In: Advances in Neural Information Processing Systems, pp. 289–296 (2009)
7. Delange, J.: Why using asynchronous communications? (2017). http://julien.gunnm.org/programming/linux/2017/04/15/comparison-sync-vs-async
8. Duchi, J.C., Jordan, M.I., Wainwright, M.J.: Local privacy and statistical minimax rates. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS), pp. 429–438, 0272-5428 (2014). https://doi.org/10.1109/FOCS.2013.53
9. Dwork, C.: A firm foundation for private data analysis. Commun. ACM **54**(1), 86–95 (2011)
10. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_14
11. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends® Theor. Comput. Sci. **9**(3–4), 211–407 (2014)
12. Erlingsson, Ú., Pihur, V., Korolova, A.: RAPPOR: randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014), pp. 1054–1067 (2014)
13. European Association for Theoretical Computer Science. Gödel Prize (2017). https://eatcs.org/index.php/component/content/article/1-news/2450-2017-godel-prize
14. Fanti, G., Pihur, V., Erlingsson, Ú.: Building a RAPPOR with the unknown: privacy-preserving learning of associations and data dictionaries. Proc. Privacy Enhancing Technol. **3**, 41–61 (2016)
15. Feldman, V., Mironov, I., Talwar, K., Thakurta, A.: Privacy amplification by iteration. ArXiv e-prints abs/1808.06651, August 2018
16. Google. Google Safe Browsing (2018). https://safebrowsing.google.com/
17. Greenberg, A.: Apple's Differential Privacy is About Collecting Your Data - But Not Your Data. In Wired, 13 June 2016
18. Kairouz, P., Oh, S., Viswanath, P.: Extremal mechanisms for local differential privacy. In: Advances in Neural Information Processing Systems, pp. 2879–2887 (2014)
19. Kenthapadi, K., Korolova, A., Mironov, I., Mishra, N.: Privacy via the Johnson-Lindenstrauss transform. J. Privacy Confidentiality **5**(1), 39–71 (2013)
20. Madden, M., Rainie, L.: Americans' Attitudes About Privacy, Security and Surveillance. Pew Research Center (2015). http://www.pewinternet.org/2015/05/20/americans-attitudes-about-privacy-security-and-surveillance/
21. Madden, M., Rainie, L.: Privacy and Information Sharing. Pew Research Center (2016). http://www.pewinternet.org/2016/01/14/privacy-and-information-sharing/
22. Brendan McMahan, H., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: AISTATS (2017)
23. Brendan McMahan, H., Ramage, D., Talwar, K., Zhang, L.: Learning differentially private language models without losing accuracy. CoRR abs/1710.06963 (2017). arxiv:1710.06963
24. McSherry, F.: Deep learning and differential privacy (2017). https://github.com/frankmcsherry/blog/blob/master/posts/2017-10-27.md
25. Nelder, J.A., Wedderburn, R.W.M.: Generalized linear models. J. R. Stat. Soc. Ser. A General **135**(1972), 370–384 (1972)
26. Nissim, K., et al.: Differential privacy: a primer for a non-technical audience (Preliminary Version). Vanderbilt J. Entertainment Technol. Law (2018)
27. Papernot, N., Abadi, M., Erlingsson, Ú., Goodfellow, I., Talwar, K.: Semi-supervised knowledge transfer for deep learning from private training data. In: 5th International Conference on Learning Representations (2016)
28. Pihur, V.: Differentially-Private "Draw and Discard" Machine Learning (2018). https://doi.org/10.48550/ARXIV.1807.04369

29. Portnoy, E., Gebhart, G., Grant, S.: In EFF DeepLinks Blog (2016). www.eff.org/deeplinks/2016/09/facial-recognition-differential-privacy-and-trade-offs-apples-latest-os-releases
30. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1310–1321 (2015)
31. Song, S., Chaudhuri, K., Sarwate, A.D.: Stochastic gradient descent with differentially private updates. In: Global Conference on Signal and Information Processing (GlobalSIP), pp. 245–248. IEEE (2013)
32. WWDC 2016. WWDC 2016 Keynote, June 2016. https://www.apple.com/apple-events/june-2016/

Privacy Preserving DCOP Solving by Mediation

Pablo Kogan[1,3], Tamir Tassa[1](✉), and Tal Grinshpoun[2,3]

[1] Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel
pablokogan@pm.me, tamirta@openu.ac.il

[2] Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
talgr@ariel.ac.il

[3] Ariel Cyber Innovation Center, Ariel University, Ariel, Israel

Abstract. In this study we propose a new paradigm for solving DCOPs, whereby the agents delegate the computational task to a set of external mediators who perform the computations for them in an oblivious manner, without getting access neither to the problem inputs nor to its outputs. Specifically, we propose MD-Max-Sum, a mediated implementation of the Max-Sum algorithm. MD-Max-Sum offers topology, constraint, and decision privacy, as well as partial agent privacy. Moreover, MD-Max-Sum is collusion-secure, as long as the set of mediators has an honest majority. We evaluate the performance of MD-Max-Sum on different benchmarks. In particular, we compare its performance to PC-SyncBB, the only privacy-preserving DCOP algorithm to date that is collusion-secure, and show the significant advantages of MD-Max-Sum in terms of runtime.

Keywords: DCOP · Max-Sum · Privacy · Multiparty computation · Mediated computing

1 Introduction

A Distributed Constraint Optimization Problem (DCOP) [2,5] is a commonly accepted and practical mathematical framework for solving coordination challenges in multi-agent systems. A DCOP consists of a set of variables that are controlled by several independent agents. Some subsets of variables may be dependent in the sense that when they are assigned values from their respective domains, different combinations of those values may incur costs. The goal is to assign values to all variables so that the sum of all incurred costs would be minimal. One of the main motivations for solving constraint optimization problems in a distributed manner is to preserve the privacy of the interacting agents. Hence, many *privacy-preserving* DCOP algorithms were proposed over the past two decades (see the review of related work in Sect. 2).

All existing DCOP algorithms (privacy-preserving or not) are carried out by the agents themselves. However, some of those algorithms, and in particular the privacy-preserving ones, require significant computing resources. In addition, all DCOP algorithms assume that the agents are connected through a communication network. We propose here a new paradigm in DCOPs: solving them in the so-called mediated model; i.e., there are external servers to whom the agents send the problem inputs in some protected manner. The external servers, whom we call *mediators*, simulate a DCOP algorithm on those inputs. At its completion, they send to the agents messages from which the agents extract the outputs, i.e., the variable assignments. Throughout this process, the mediators remain oblivious to the problem inputs, to the content of the protected messages that they exchange, and to the outputs.

Performing the DCOP algorithm in such a mediated manner offers several significant advantages: (a) it protects the privacy of the agents and their sensitive private data; (b) the agents do not need to establish a communication network amongst them; and (c) it delegates the computational workload from the agents, who may have limited computational resources, to dedicated servers.

In this work we demonstrate the power of mediated computing by presenting MD-Max-Sum, a mediated execution of the Max-Sum algorithm [1]. In MD-Max-Sum, the agents send to the mediators secret shares [13] in their private inputs. The mediators proceed to execute the Max-Sum computations on the shares of the problem inputs, while remaining oblivious to the value of the underlying inputs. At the end, the mediators send to the agents messages from which the agents may infer the assignments to their variables. If the mediators have an honest majority (namely, the number of colluding mediators is smaller than the number of mediators outside the coalition), then MD-Max-Sum provides topology, constraint, and decision privacy, as well as partial agent privacy (see [9] for the definition of those notions). Those security guarantees hold against any collusion among the set of agents.

2 Related Work

Léauté and Faltings [9] devised three secure versions of the complete DCOP-solving algorithm DPOP [12], each with different runtimes and privacy guarantees. In their study they suggested four notions of privacy for the DCOP framework, to which we adhere in this study: agent, topology, constraint, and decision privacy. Grinshpoun and Tassa [3] presented a privacy-preserving version of another complete algorithm – SyncBB [5]. The resulting method, P-SyncBB, preserves topology, constraint, and decision privacy.

Since complete algorithms are not scalable, research efforts were invested also in designing privacy-preserving implementations of incomplete algorithms. Tassa et al. [15] developed the P-Max-Sum algorithm, which runs Max-Sum with cryptographic enhancements in order to preserve privacy. P-Max-Sum provides topology, constraint, and decision privacy, and it may be extended to provide also agent privacy. Grinshpoun et al. [4] devised another incomplete privacy-preserving algorithm, P-RODA, which is based on region optimality [6,7]. P-RODA provides constraint privacy and partial decision privacy.

All of the above mentioned privacy-preserving DCOP algorithms assume *solitary* conduct of the agents. However, if two or more agents collude and combine the information that they have, they may extract valuable information about other agents. To address the risk of collusion, Tassa et al. [14] introduced PC-SyncBB, the first privacy-preserving DCOP algorithm that is *collusion-secure*. It is secure under the assumption that the agents have an honest majority. PC-SyncBB does extensive usage of Secure Multiparty Computation (MPC) [16] in order to obliviously compare between costs of partial assignments.

In this work we propose MD-Max-Sum, the first *incomplete* privacy-preserving DCOP algorithm that is *collusion-secure*.

3 DCOP Definitions and the Max-Sum Algorithm

A Distributed Constraint Optimization Problem (DCOP) [5] is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of agents $A_1, A_2, \ldots, A_N$, $\mathcal{X}$ is a set of variables $X_1, X_2, \ldots, X_N$, $\mathcal{D}$ is a set of finite domains $D_1, D_2, \ldots, D_N$, and $\mathcal{R}$ is a set of relations (constraints). Each variable X_n, $n \in [N] := \{1, 2, \ldots, N\}$, takes values in the domain D_n, and it is held by the agent A_n.[1] Each constraint $C \in \mathcal{R}$ defines a non-negative cost for every possible value combination of some subset of variables, and is of the form $C : D_{n_1} \times \cdots \times D_{n_k} \rightarrow [0, q]$, for some $1 \leq n_1 < \cdots < n_k \leq N$, and a publicly known maximal constraint cost q.

An *assignment* is a pair including a variable and a value from that variable's domain. The goal of the agents is to find assignments to their variables so that the sum of all costs that those assignments incur would be minimal.

We consider here a binary version of DCOPs, in which every $C \in \mathcal{R}$ constraints exactly two variables and takes the form $C_{n,m} : D_n \times D_m \rightarrow [0, q]$, where $1 \leq n < m \leq N$. Such an assumption is customary in DCOP literature, see e.g. [11,12]. As the domains are finite, they me be ordered. Hence, the binary constraint $C_{n,m}$ between X_n and X_m may be described by a matrix, which we also denote by $C_{n,m}$, of dimensions $|D_n| \times |D_m|$; in that matrix, $C_{n,m}(i, j)$ equals the cost that corresponds to the assignment of the ith value in D_n to X_n and the jth value in D_m to X_m, where $1 \leq i \leq |D_n|$ and $1 \leq j \leq |D_m|$.

The constraint graph $G = (V, E)$ is an undirected graph over the set of variables $V = \mathcal{X}$, where an edge in E connects two variables if and only if they are constrained. If we define for every pair of variables $(X_n, X_m) \notin E$ a constraint matrix $C_{n,m}$ which is the zero matrix of dimensions $|D_n| \times |D_m|$, then the set of all such matrices,

$$\mathcal{C} := \{C_{n,m} : 1 \leq n < m \leq N\}, \tag{1}$$

encompasses all topology and constraint information.

Every DCOP is also associated with a so-called *factor graph*. It is a bipartite graph $G' = (V', E')$ that is defined as follows. The set V' has two types of nodes:

[1] We make the standard assumption that the number of variables equals the number of agents, and that each variable is held by a distinct single agent, see e.g. [11,12].

variable nodes, $X_1, \dots, X_N$, and *function nodes*, X_e, for each $e = (X_n, X_m) \in E$. The edge set E' has an edge connecting X_n with X_e if and only if e is an edge in G that is adjacent to X_n.

The Max-Sum algorithm [1] is an iterative message-passing algorithm that operates on the factor graph G'. In every iteration a message is sent from each node in V' to each one of its adjacent nodes. In the kth iteration, each variable node X_n sends to each adjacent function node X_e a message denoted $Q^k_{n\to e}$, while the message in the opposite direction is denoted $R^k_{e\to n}$; both messages are vectors of dimension $|D_n|$. In the $k = 0$ iteration all messages are zero. After completing the kth iteration, the messages in the next iteration will be as follows. Fixing a variable node X_n and letting V_n be the set of function nodes adjacent to X_n in G', then for each $X_e \in V_n$, X_n will send to X_e the vector

$$Q^{k+1}_{n\to e} := \sum_{X_f \in V_n \setminus \{X_e\}} R^k_{f\to n} \,. \tag{2}$$

As for messages sent from function nodes, if X_e is a function node that connects the two variable nodes X_n and X_m then the message sent from X_e to X_n is

$$R^{k+1}_{e\to n}(x) := \min_{y \in D_m} \left[C_{n,m}(x, y) + Q^k_{m\to e}(y) \right] , \quad \forall x \in D_n \,; \tag{3}$$

the message that X_e sends to X_m is constructed similarly. Finally, after completing a preset number K of iterations, each variable node X_n computes $\overline{R}_n := \sum_{X_e \in V_n} R^K_{e\to n}$ and then selects a value $x \in D_n$ for which $\overline{R}_n(x)$ is minimal.

4 Mediated Max-Sum

In order to implement Max-Sum in a manner that preserves the privacy of the agents even when some of them collude, we propose herein MD-Max-Sum – an implementation of Max-Sum in the mediated model.

Let $\mathbf{M} = \{M_1, \dots, M_L\}$ be an external committee of so-called *mediators*. The agents in $\mathcal{A}$ will share their DCOP private inputs, namely, the topology and constraint information, with the mediators using a t-out-of-L threshold secret sharing scheme [13], where $t := \lfloor (L+1)/2 \rfloor$. Specifically, the agents distribute to the mediators t-out-of-L shares in each of the entries in each of the matrices in $\mathcal{C}$, see Eq. (1); the underlying secret sharing field will be denoted henceforth by $\mathbb{Z}_p$. The agents trust the mediators to have an honest majority, in the sense that if some of the mediators decide to collude in order to reconstruct the shared private data, the number of colluding mediators would be smaller than the number of mediators outside the coalition. Under that assumption, the mediators cannot recover the private inputs that were shared with them, since at least t mediators have to collude in order to be able to reconstruct the shared secrets, and $t = \lfloor (L+1)/2 \rfloor \geq L - t$.

After the agents had completed sharing all their private inputs with the mediators, they go to rest and the mediators start emulating the performance of the entire Max-Sum algorithm by implementing secure multiparty computation (MPC) on the shared data. The main challenge in this regard is to design an implementation of Max-Sum that operates on *shared* data, namely, in a manner that is oblivious to the underlying topology and constraint values.

When the mediators complete their emulation of Max-Sum, say by running an agreed preset number of iterations, K, they send to each of the agents a message from which that agent can infer the assignment of its variable in the solution that the algorithm had found.

In order to hide the constraint graph topology from the mediators, MD-Max-Sum operates on an augmented version $G_+ = (V, E_+)$ of the constraint graph $G = (V, E)$. G_+ is a complete graph in the sense that E_+ includes all $\binom{N}{2}$ pairs of nodes/variables from $V = \mathcal{X}$, where all edges $(X_n, X_m) \in E_+ \setminus E$ are associated with a zero constraint matrix, $C_{n,m}$, as described earlier in Sect. 3. In the full version of this paper [8] we show that the addition of such so-called *phantom edges* does not change the algorithm's outputs. However, with such added phantom edges, the mediators, who only get shares in the constraint matrices, cannot distinguish between a zero matrix and a non-zero matrix and, consequently, they cannot tell which of the edges in the augmented graph are phantom ones, so the graph topology is preserved.

4.1 The MD-Max-Sum Algorithm

We assume that all agents know the total number of agents N, and the identifying index $n \in [N]$ of each agent. In addition, the sizes of all domains, $|D_n|$, $n \in [N]$, are also publicly known.

4.1.1 Distributing to the Mediators Shares in the Problem Inputs

In this preliminary stage, the agents share with the mediators the problem inputs, which, as explained in Sect. 3, are encoded through the set of matrices $\mathcal{C}$, see Eq. (1). To do so, each agent A_n, $1 \le n \le N-1$, shares with the mediators $\mathbf{M}$ the constraint matrices $C_{n,m} \in \mathcal{C}$ for all $n < m \le N$, where, as explained in Sect. 3, the matrix $C_{n,m}$ is of dimensions $|D_n| \times |D_m|$ and it either spells out the constraint values between those two agents, or, if they are not constrained, it is the zero matrix. The matrices are shared by performing an independent t-out-of-L secret sharing for each entry in each of those matrices, where $t = \lfloor (L+1)/2 \rfloor$. Letting $n < m \in [N]$ be indices of two agents, and $\ell \in [L]$ be an index of a mediator, we denote the share of the cost $C_{n,m}(i,j)$ that the mediator M_ℓ receives by $C^\ell_{n,m}(i,j)$. The entire matrix of shares that M_ℓ receives is denoted

$$C^\ell_{n,m} = \left(C^\ell_{n,m}(i,j) : 1 \le i \le |D_n|, 1 \le j \le |D_m|\right) .$$

After each mediator M_ℓ, $\ell \in [L]$, got its share matrix $C^\ell_{n,m}$ for all $1 \le n < m \le N$, they have all problem inputs and they may now begin an MPC emulation of Max-Sum over those inputs.

We assume that the mediators have an *honest majority*. Hence, as we use t-out-of-L secret sharing with $t = \lfloor (L+1)/2 \rfloor$, the mediators cannot learn any information on the content of the constraint matrices. Therefore, not only the constraints themselves are kept secret, also the topology is kept secret, since the mediators cannot tell from their shares whether $C_{n,m}$ is the zero matrix or not.

While Max-Sum, as well as P-Max-Sum, operate on the exact factor graph G', the mediated algorithm MD-Max-Sum operates on an *augmented factor graph*, denoted $G'_+ = (V'_+, E'_+)$, in which every two variable nodes are connected through a function node, even if some of those function nodes stand for a zero/phantom constraint (which was introduced only for the purpose of hiding the real topology of G from the mediators).

After the agents finish distributing to the mediators shares in the problem inputs, they go to rest and let the mediators do the work. The mediators start an emulation of each of the iterations in Max-Sum. They do so by producing proper shares in the true messages that would have been sent along each edge of the factor graph, if the agents had run the Max-Sum algorithm by themselves.

We proceed to explain in the next sections the details of the MD-Max-Sum implementation. Specifically, we need to explain how in each iteration of the algorithm, the mediators create shares in the messages that the corresponding Max-Sum algorithm would have generated. In doing so, we focus on an arbitrary pair of neighboring nodes in the augmented factor graph: a variable node X_n, $n \in [N]$, and a function node, X_e, where $e = (X_n, X_m)$ and $m \in [N] \setminus \{n\}$.

4.1.2 Producing Shares in the Messages of the Initial Iteration

In iteration 0, all of the L mediators have to emulate zero messages between X_n and X_e,

$$Q^0_{n\to e} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}, \quad R^0_{e\to n} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}. \tag{4}$$

To do so, each mediator M_ℓ, $\ell \in [L]$, creates for himself corresponding zero share vectors as follows:

$$Q^{0,\ell}_{n\to e} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}, \quad R^{0,\ell}_{e\to n} = (0, \ldots, 0) \in \mathbb{Z}_p^{|D_n|}. \tag{5}$$

Note that no interaction between the mediators is needed at this stage, and that the L vector shares of the Q-messages in Eq. (5) are t-out-of-L vector shares in the zero Q-messages in Eq. (4), and likewise for the R-messages.

4.1.3 Producing Shares in Q-messages

In iteration $k+1$, the mediators have to emulate the message $Q^{k+1}_{n\to e}$ from the variable node X_n to the adjacent function node, X_e, where $e = (X_n, X_m)$. In view of Eq. (2), and as the mediators already have t-out-of-L shares in R-messages of the kth iteration, such a computation can be done, without interaction between the mediators, by computing $Q^{k+1,\ell}_{n\to e} := \sum_{X_f} R^{k,\ell}_{f\to n}$, where the sum is over all $N-2$ function nodes, X_f, between X_n and X_i for any $i \in [N] \setminus \{n, m\}$.

4.1.4 Producing Shares in *R*-messages

Here, we concentrate on the more involved task of computing t-out-of-L shares in the R-messages, $R^{k+1}_{e\to n}$, from the function node, X_e, where $e = (X_n, X_m)$, to the variable node X_n. We rewrite Eq. (3) in the following manner,

$$R^{k+1}_{e\to n}(x) := \min_{y\in D_m} B^k_{n,m}(x,y)\,, \quad x \in D_n\,, \tag{6}$$

where $B^k_{n,m}(x,y)$ denotes the sum

$$B^k_{n,m}(x,y) := C_{n,m}(x,y) + Q^k_{m\to e}(y)\,. \tag{7}$$

The L mediators hold t-out-of-L shares in $C_{n,m}(x,y)$ for all $(x,y) \in D_n \times D_m$ (denoted $C^\ell_{n,m}(x,y)$, $\ell \in [L]$), since such shares were generated and distributed to them by the agents in the preliminary stage. Moreover, the mediators had computed in the kth iteration t-out-of-L shares in $Q^k_{m\to e}(y)$ for all $y \in D_m$, where M_ℓ's shares are denoted $Q^{k,\ell}_{m\to e}(y)$. Hence, $C^\ell_{n,m}(x,y) + Q^{k,\ell}_{m\to e}(y)$, which we denote by $B^{k,\ell}_{n,m}(x,y)$, are t-out-of-L shares in $B^k_{n,m}(x,y)$, as implied by Eq. (7) and the linearity of secret sharing. Hence, the main computational challenge is to compute t-out-of-L shares in the left-hand side of Eq. (6) from the shares that the mediators hold in each of the terms on the right-hand side of Eq. (6). This task is non-trivial because the minimum function is non-linear.

Protocol 1, which we describe below, is simultaneously executed by each of the L mediators. It is executed for each pair of a function node in the augmented factor graph, X_e, where $e = (X_n, X_m)$, and one of its two adjacent variable nodes, X_n. At the completion of that protocol, each mediator M_ℓ holds a share $R^{k+1,\ell}_{e\to n}$ in $R^{k+1}_{e\to n}$. That protocol will be executed in every iteration $N(N-1)$ times, as there are $\binom{N}{2} = N(N-1)/2$ function nodes in the augmented factor graph G'_+, and each one of them has two adjacent variable nodes.

Protocol 1: Computing shares in an R-message from the function node X_e, where $e = (X_n, X_m)$, to the variable node X_n.

```
Input: Mediator $M_\ell$, $\ell \in [L]$, holds a $t$-out-of-$L$ share, $B^{k,\ell}_{n,m}(x,y)$, in
       $B^k_{n,m}(x,y)$, for every $x \in D_n$ and $y \in D_m = \{y_1,\dots,y_{|D_m|}\}$.
1 forall $x \in D_n$ do
2     $M_\ell$ sets $\beta^\ell_{n,m}(x) \leftarrow B^{k,\ell}_{n,m}(x,y_1)$
3     forall $j = 2,\dots,|D_m|$ do
4         if COMPARE($\{B^{k,\ell}_{n,m}(x,y_j)\}_{\ell\in[L]}, \{\beta^\ell_{n,m}(x)\}_{\ell\in[L]}$) = true then
5             $M_\ell$ sets $\beta^\ell_{n,m}(x) \leftarrow B^{k,\ell}_{n,m}(x,y_j)$
6     $M_\ell$ sets $R^{k+1,\ell}_{e\to n}(x) \leftarrow \beta^\ell_{n,m}(x)$
Output: Mediator $M_\ell$, $\ell \in [L]$, gets a $t$-out-of-$L$ share $R^{k+1,\ell}_{e\to n}(x)$ in $R^{k+1}_{e\to n}(x)$.
```

The external loop in the protocol (lines 1–6) is over all values x in the domain D_n, i.e., over all entries in the vector message $R^{k+1}_{e\to n}$. For each such $x \in D_n$, the mediators have to find the minimum among $\{B^k_{n,m}(x,y) : y \in D_m\}$, where each of the values in that set is shared by a t-out-of-L scheme among them. The t-out-of-L shares of the minimum will be stored in $\beta^\ell_{n,m}(x)$, $\ell \in [L]$. First (line 2), each mediator initiates its β-shares with the shares corresponding to $B^k_{n,m}(x, y_1)$. Then (lines 3–5), for each y_j, $j = 2, \dots, |D_m|$, the mediators compare $B^k_{n,m}(x, y_j)$ to the current minimum, in which they have t-out-of-L shares in $\beta^\ell_{n,m}(x)$, $\ell \in [L]$. The comparisons are performed in a secure manner by invoking a distributed sub-protocol that all mediators jointly execute, as will be explained below. If $B^k_{n,m}(x, y_j)$ is smaller than the current minimum, then each mediator updates its share of the minimum (line 5).

In order to perform comparisons between values that are known to the mediators only through t-out-of-L shares, without recovering those values and perform the comparison over those recovered values, Protocol 1 calls upon an MPC sub-protocol called **COMPARE** (line 4), which all the mediators run together in a distributed manner. That sub-protocol assumes that the mediators have t-out-of-L shares in two values $x, y \in \mathbb{Z}_p$; it returns **true** if $x < y$ (when x and y are interpreted as integers) and **false** otherwise. This sub-protocol is perfectly secure in the sense that it reveals to the mediators nothing about the two compared values beyond the final output bit which indicates which of the two is smaller. (A full description of **COMPARE** is provided in [8].)

Finally (line 6), each mediator stores in $R^{k+1,\ell}_{e\to n}(x)$ its share in the minimum that was found above, $\beta^\ell_{n,m}(x)$.

4.1.5 Termination

After completing a preset number of K iterations, the final assignment to X_n, $n \in [N]$, is determined by the minimal entry in $\overline{R}_n = \sum_{X_e \in V_n} R^K_{e\to n}$. To find that assignment, each agent A_n, $n \in [N]$, selects a subset of t mediators and asks them for their shares in the vector $\overline{R}_n$. Using those vector shares, A_n can recover each of the $|D_n|$ entries in $\overline{R}_n$. Afterwards, A_n finds the minimal entry in $\overline{R}_n$ and then assigns the corresponding value to X_n.

4.2 Correctness and Privacy

The MD-Max-Sum algorithm is correct and privacy-preserving, as stated in Theorems 1 and 2. (The proofs are given in the full version of this paper [8]).

Theorem 1. *When* MD-Max-Sum *and* Max-Sum *are executed the same number of iterations K on the same input problem, they will issue the same assignments to all variables.*

Theorem 2. MD-Max-Sum *provides topology, constraint, and decision privacy, as long as the mediators have an honest majority.*

5 Experimental Evaluation

We implemented and executed the MD-Max-Sum algorithm on the AgentZero simulator [10], running on AWS C5a instances comprised of a 2nd generation AMD EPYC$^{\text{TM}}$7R32 processor and 64 GB memory, except for the call to the

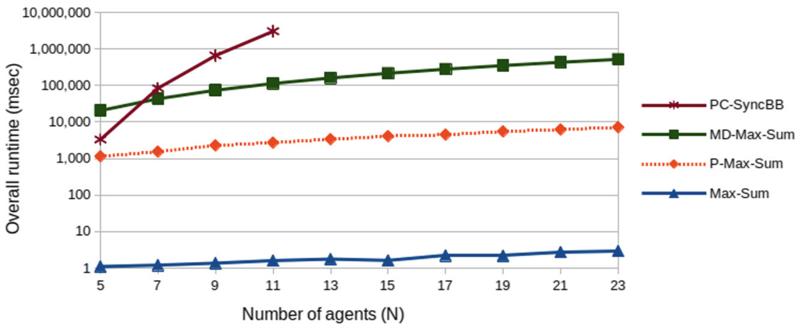


Fig. 1. Unstructured random graphs ($p_1 = 0.3$, $|D_n| = 5$), varying N

COMPARE sub-protocol that was executed over LAN with EC2 machines of type c5.large in Amazon's North Virginia data center. We compared MD-Max-Sum with the baseline algorithm Max-Sum (no privacy) and P-Max-Sum [15] (provides privacy, but not against coalitions). As shown in [15], and stated above in Theorem 1, both of those privacy-preserving implementations of Max-Sum simulate perfectly the basic Max-Sum. We used in all experiments $K = 10$ iterations in each of these algorithms, similarly to [15]. MD-Max-Sum was executed with $L = 5$ mediators. In addition, we included in our experiments the PC-SyncBB algorithm [14], which is the only other DCOP-solving algorithm that is privacy-preserving and collusion-secure. Recall that unlike MD-Max-Sum, PC-SyncBB is a complete algorithm; hence, it outputs the optimal solution but it is expected to be more time consuming.

The first experiment, shown in Fig. 1, was conducted on unstructured random graphs with constraint density $p_1 = 0.3$ and domains of size $|D_n| = 5$, $n \in [N]$. We varied the number of agents N to observe the scalability of the algorithms. The cut-off time for a single execution was set to 30 min.

The performance gap between Max-Sum and P-Max-Sum demonstrates the price of privacy. The gap between P-Max-Sum and MD-Max-Sum demonstrates the price of collusion security. Those two gaps remain constant. Conversely, the gap between MD-Max-Sum and PC-SyncBB, which demonstrates the price of completeness, is constantly growing. For small problems with $N \leq 7$, PC-SyncBB is competitive with MD-Max-Sum and even with P-Max-Sum. However, as the number of agents increases, we can see that the performance of PC-SyncBB becomes much more time-consuming than MD-Max-Sum's. This advantage of MD-Max-Sum over PC-SyncBB is explained as follows: a significant portion of the runtime of both algorithms is in performing secure comparisons between secret values. In PC-SyncBB, that MPC sub-protocol is carried out by all agents; in MD-Max-Sum, on the other hand, it is carried out by the mediators. The runtime of this computation depends on the number of interacting parties, see [14, Table 1]. Hence, while the time spent in PC-SyncBB on secure comparisons increases with N, in MD-Max-Sum it is independent of N. This mitigation of the dependency of the runtime on N demonstrates the strength of the *mediated model*. (Of course, the runtime of MD-Max-Sum does depend on N through other computations, outside the secure comparisons in **COMPARE**, since N affects the size of the graph.)

We also evaluated the algorithms on 3-color graph coloring problems, similar to the setting described by Zivan et al. [17]. In this setting, for every $1 \leq n < m \leq N$, $C_{n,m}(x, y) = q$ if $x = y$ and $C_{n,m}(x, y) = 0$ if $x \neq y$, for some positive constant q. Figure 2 presents the runtime of the algorithms on 3-color graph problems with $p_1 = 0.4$ and shows similar scalability properties to the previous experiments. The small domain size, $|D_n| = 3$, enables us to experiment with problems of larger sizes. For this experiment, we started with $N = 5$ and moved up to $N = 75$ agents in steps of 10. While all other algorithms remain within the cut-off limit of 30 min per single execution, the runtime of PC-SyncBB exceeded the cut-off limit already for $N = 20$. Hence, we include in Fig. 2 the runtime of PC-SyncBB for $N = 19$, which was the highest number of agents that could be processed within 30 min. The trends are similar to those in the former experiment.

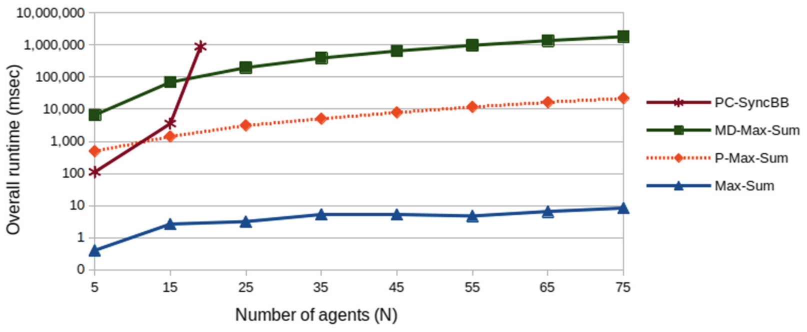


Fig. 2. 3-color graph coloring problems ($p_1 = 0.4$), varying N

6 Conclusion

In this work we introduced MD-Max-Sum, the first incomplete privacy-preserving DCOP algorithm that is also collusion-secure. It is an implementation of Max-Sum in the mediated model of computation. It preserves topology, constraint, decision, and partial agent privacy. We analyzed the security and correctness of the algorithm and, using experimentation, demonstrated its characteristics, its advantages over the only other collusion-secure DCOP algorithm, PC-SyncBB, and its viability.

Aside from the performance gains achieved by utilizing an incomplete algorithm (as opposed to PC-SyncBB that is based on a complete algorithm), the transition to the mediated model offers other significant benefits: MD-Max-Sum is privacy-preserving and is immune to any coalition among the agents, under the assumption of an honest majority within the mediators; the agents do not need to communicate with each other, a significant advantage in settings where the agents do not have an efficient way to communicate among themselves; the agents, that may run on computationally-bounded devices, can outsource costly and cryptographically-complex computations to dedicated servers; and, finally, MD-Max-Sum is more robust than all previous DCOP algorithms since if an agent goes offline (e.g., due to a technical failure) after secret sharing its private data to the mediators, the algorithm can still be executed and issue the correct outputs to all agents.

We believe that the mediated model of computation could be successfully implemented for other DCOP algorithms as well as for various problems of federated learning, in order to achieve enhanced privacy guarantees, and to reap the advantages of the mediated model of computation as we have identified herein.

Acknowledgments. This work was partially supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office.

References

1. Farinelli, A., Rogers, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS, pp. 639–646 (2008)
2. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: a survey. J. Artif. Intell. Res. **61**, 623–698 (2018)
3. Grinshpoun, T., Tassa, T.: P-SyncBB: a privacy preserving branch and bound DCOP algorithm. J. Artif. Intell. Res. **57**, 621–660 (2016)
4. Grinshpoun, T., Tassa, T., Levit, V., Zivan, R.: Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. Artif. Intell. **266**, 27–50 (2019)
5. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: CP, pp. 222–236 (1997)
6. Katagishi, H., Pearce, J.P.: Kopt: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In: DCR (2007)
7. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: AAMAS, pp. 133–140 (2010)
8. Kogan, P.: Privacy Preserving Solution of DCOPs by Mediation. Master's thesis, supervised by Tassa, T. and Grinshpoun, T., The Open University of Israel (2022). https://www.openu.ac.il/Lists/MediaServer_Documents/PersonalSites/TamirTassa/MD_Max_Sum.pdf
9. Léauté, T., Faltings, B.: Protecting privacy through distributed computation in multi-agent decision making. J. Artif. Intell. Res. **47**, 649–695 (2013)
10. Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., Grubshtein, A.: AgentZero: a framework for simulating and evaluating multi-agent algorithms. In: Agent-Oriented Software Engineering, pp. 309–327 (2014)
11. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artif. Intell. **161**, 149–180 (2005)
12. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: IJCAI, pp. 266–271 (2005)
13. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)

14. Tassa, T., Grinshpoun, T., Yanai, A.: PC-SyncBB: a privacy preserving collusion secure DCOP algorithm. Artif. Intell. **297**, 103501 (2021)
15. Tassa, T., Grinshpoun, T., Zivan, R.: Privacy preserving implementation of the Max-Sum algorithm and its variants. J. Artif. Intell. Res. **59**, 311–349, Article no. 103501 (2017)
16. Yao, A.C.: Protocols for secure computation. In: FOCS, pp. 160–164 (1982)
17. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. Artif. Intell. **212**, 1–26 (2014)

BFLUT
Bloom Filter for Private Look Up Tables

Shlomi Dolev[1](✉), Ehud Gudes[1], Erez Segev[1], Jeffrey Ullman[2], and Grisha Weintraub[1]

[1] Ben-Gurion University of the Negev, Beersheba, Israel
dolev@cs.bgu.ac.il

[2] Stanford University, Stanford, USA

Abstract. Open addressing hash tables, possibly under double hashing policy, are regarded more memory efficient than linked list hashing; as the memory used for pointers can be used for a longer table, and allow better-expected performance as the load factor is smaller and there are fewer expected collisions. We suggest further eliminating the single pointer to the memory location used in each entry of the open addressing, and using a single bit per entry, namely use a Bloom Filter to encode the memory address. Thus, obtain even a better load factor, with the same memory, and less number of wrongly mapped addresses when the load is low.

Moreover, we can prove that the content in the lookup table that is based on the bloom filter is pseudo-random (in the level of randomization implied by the hash function), thus, keeping the items *and* the addresses that the LookUp Table (LUT) encodes private.

1 Introduction

Hashing into hash tables and into Bloom filters (BF) share similar concepts [4,6,10,12], still Bloom Filters are not used to function as look up tables (LUT) that always return values inserted, as hash tables do. We propose a secure implementation and privacy preserving of LUT using BF, which we call BFLUT.

We suggest to encode an item name, say, "John Smith" with item address, say "0110", by making sure that the bit of the bloom filter, in the address $H(John\ Smith0)$ is turned on, then the bits of the bloom filter in the addresses $H(John\ Smith01)$, $H(John\ Smith011)$, $H(John\ Smith0110)$ are also turned on. Searching for the "John Smith" address will start in finding out whether the bloom filter bits in the addresses $H(John\ Smith0)$ and $H(John\ Smith1)$ are set (Fig. 1). If none of them is set then we can return that "John Smith" is not in the LUT. If only one of them are set, say, $H(John\ Smith0)$ then, addresses that start

This research was (partially) funded by a grant from the Ministry of Science and Technology, Israel & the Japan Science and Technology Agency (JST), the German Research Funding (DFG, Grant #8767581199), Genesis Consortium, the Rita Altura trust chair in computer science, and by the Lynne and William Frankel Center for Computer Science.

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 499–505, 2022.
https://doi.org/10.1007/978-3-031-07689-3_35

with 0 are plausible, and we continue checking the possibility that the address starts with 00 or 01, by hashing $H(john\ Smith00)$ and $H(John\ Smith01)$. If both the bits in addresses $H(John\ Smith0)$ and $H(John\ Smith1)$ are set in the bloom filter, then we continue to investigate extensions for both of them. Note that the load factor (say, kept to be less than 1/2) and the (pseudo) randomness of the hash function imply a small set of plausible (non relevant) addresses.

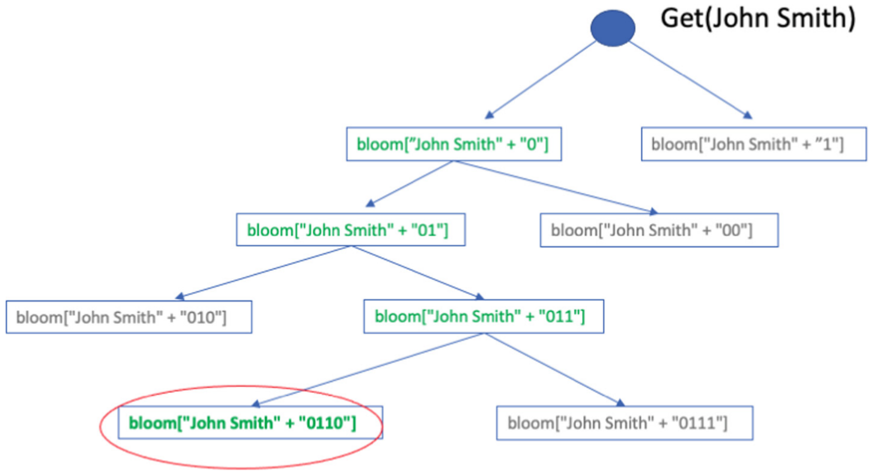

Get(John Smith)
bloom["John Smith" + "0"]
bloom["John Smith" + "1"]
bloom["John Smith" + "01"]
bloom["John Smith" + "00"]
bloom["John Smith" + "010"]
bloom["John Smith" + "011"]
bloom["John Smith" + "0110"]
bloom["John Smith" + "0111"]

Fig. 1. Get value from BFLUT example

Beyond provable privacy of the LUT items and address(es) our design of the Bloom Filter LUT (BFLUT) can fit the cases in which one has a limited RAM memory, memory that implies fast inquires on table of bits, and fast hash function executions, but expensive access to the actual item record(s), possibly stored in disk or in Data Lakes [13].

Since the addresses (or pointers) are encoded by a series of hashes, the number of bits that are founded with value 1 in the bloom filter (when the load factor is approaching 1/2) is approximately half of the address, hence allowing further saving in the size of the bloom filter.

Assume an address requires k bits, in this case the array used for an open addressing hash table, say under the policy of double hashing, and load factor of approximately 1/2, requires mk bits to handle $m/2$ keys.

Consider a BFLUT for the same k and m and the same load factor (measured in terms of the ratio of bits with value 1 over the total number of bits in the BFLUT). When inserting a new member, when the BFLUT load factor is close to 1/2, only approximately half of the m hash mapping find the value 0 that should be flipped to 1, and hence uses only approximately $m/2$ bits to encode the address.

Privacy is obvious. Given a BFLUT where every bit value is defined by a pseudo-random hash function, the entire content of the BFLUT reveals nothing on the keys nor on the addresses prior to guessing the key and using the hash function to reveal the address(es). As mentioned, the addresses maybe addresses to a disk or lake locations, phone numbers of spies hidden as part of a totally different scope BFLUT.

A particular use case can be a public publication of a secret directory (look up table), where the information publicized is not useful unless one knows a secret *salt* string (or private key) that is concatenated, say as a prefix of the inputs to the hash function (both in insertion and in search). The resulting scheme is post-quantum (as apposed to public key encryption systems such as RSA) when the hash function used is SHA256 or SHA512.

2 Problem Statement and Motivation

Given a set of keys K and a set of integers $V = [0, 2^m - 1]$ for some $m > 0$, we want to map keys from K to values from V. The mapping is a relation $R \subseteq K \times V$

Our goal is to store R in a data structure D that satisfies the following properties:

- Supported operations on D are:
  - $D.put(key, value)$ - inserts a pair $(key, value)$ into D
  - $D.get(key)$ - returns values mapped to a given key
- The memory footprint of D should be as small as possible
- D should be privacy-preserving, meaning that by having access to D an adversary cannot learn anything about actual values in R (both keys and values)
- False positives are acceptable, while false negatives are not, meaning that $\forall k \in K, D.get(k) \supseteq \{v|(k,v) \in R\}$

One motivation scenario is an implementation of an inverted index. Say we have a set of text documents $D = \{D_1, D_2, ..., D_n\}$ and we want to create an index that maps keywords from the documents in D to the documents containing them (Table 1).

Table 1. Inverted index example

Keyword	Documents
word1	Doc1, Doc23
word2	Doc7
word3	Doc6, Doc27
....	....

Both index and documents can be uploaded to the untrusted public cloud and multiple clients can query the index by a specific keyword to get ids of documents containing the keyword, and further download the documents from the cloud by using their ids. If documents contain sensitive information, they can be encrypted before uploading them to the cloud.

3 Memory Comparison with Hash Tables

The most relevant data-structure competitor is Hash Table. Assume the implementation uses a hash table to implement a look up table, where each entry of the table consists of k bits to describe an address (pointer) in a significantly slower and cheaper memory. Note that we do not store the key in the hash table, as then the memory comparison will be biased in favor of the BFLUT, but at the same time we allow the Hash Table to return several addresses as candidates for the address of a given key. Assume further that the number of entries in the Hash Table is m, so the total number of bits used to implement the hash table is $m \cdot k$ bits.

The case in which the load factor of the Hash Table is $\alpha = 1/2$ HT is better than BFLUT with one hash. Assume further that $m/2$ entries are occupied in the open addressing based Hash Table, implying a load factor, $\alpha_{ht} = 1/2$. A search in the hash table for an address of a given *key* must stop in an empty slot, as there is no clear indication that the address(es) found by the hash function in the first probed location(s) does (do, respectively) not belong to another key.

The expected number of items found until the first empty location is found, given that the address of the key has been inserted into the table, is $\Sigma_{i=1}^{m-1} 1/(2^{i-1}) = 2 - 1/2^{m-1}$. Which is close to 2 elements per search.

Using the same amount of memory $m \cdot k$ bits the BFLUT, and $m/2$ keys, α_{bf}, the number of bits that are set to 1 divided by $m \cdot k$ will be smaller than 1/2. If we consider each of the $m/2$ keys hashed k times, then by a reduction to the number of empty bins when throwing $(m \cdot k)/2$ balls into $m \cdot k$ bins, we get that the expected fraction of empty bins is: $((k \cdot m - 1)/(k \cdot m))^{k \cdot m/2} \approx 0.6$, and therefore $\alpha_{bf} \approx 0.4 < 0.5$.

Thus, the expected number of returning items when using the binary search strategy in the BFLUT. The expected number of wrong addresses prefix of length 1 is α_{bf}. The expected number of wrong prefixes of length 2 is $2\alpha_{bf}^2 + \alpha_{bf}$, as either an extension of a wrong prefix or an extension of the right prefix, similarly the expected number of wrong prefixes of length 3 is $4\alpha_{bf}^3 + 2\alpha_{bf}^2 + \alpha_{bf}$. Therefore, the expected number of wrong addresses to be returned is $r(k) = 2r(k-1)\alpha_{bf} + \alpha_{bf}$ where $r(1) = 2/5$, converges to approximately 2 and the total with the right address is 3.

The case in which the load factor of the Hash Table is $\alpha_{ht} = 1/4$ BFLUT is better with two hashes. In this case, the expected number of items found until the first empty location is found, given that the address of key has been inserted to the table, is $\Sigma_{i=1}^{m-1} 1/(4^{i-1}) \approx 4/3$.

$((k \cdot m - 1)/(k \cdot m))^{k \cdot m/4} \approx 0.76$, and therefore $\alpha_{bf} \approx 0.24 < 0.25$.

Thus, the expected number of returning items when using the binary search strategy in the BFLUT. The expected number of wrong addresses prefix of length 1 is α_{bf}. The expected number of wrong prefixes is 0.28, and the total is $1.46 > 1.33$.

If we take the optimal number of hash function, h, calculated for a given mk and given load factor of the BF $h = 1/\alpha_{ht} \ln 2$ where α_{ht} is the load factor of the equivalent hash table, in our case $\alpha_{ht} = 0.25$, we get in this case approximately 2.77 hash functions. When we use two hash functions instead of one we return less addresses than hash table with the same memory.

The obtained α_{bf} for the case of two hash functions is 0.4 as in $\alpha_{ht} = 0.5$ above. Then the expected number of returned items is replacing α_{bf} to $(\alpha_{bf})^2$ which is 0.16, plugin in into the expected number of returning items we BUFLT performs better $1.24 < 1.33$.

The above results are in favor for the Hash Table alternative as the memory used when a single address is mapped to a key is better than the size of memory required by the BFLUT. However, when the number of hash functions approach its optimal number the BFLUT performs better.

Experiment Results. An implementation of the BFLUT[1] has been used to verify the above BFLUT performance calculations, using SHA256 as the (cryptographic post quantum) hash function. Following the analysis in [11] we show that the false positive can be tuned by (slightly) enlarging the size of the Bloom filter and using fitting (greater) number of hash functions.

4 Related Work

When a standard hash table with separate chaining is used to store key-value pairs, additional memory should be used for pointers. In the case of open addressing, there is no simple way to encode multiple values per key and the values are exposed to anyone who has access to the table. In our approach, we do not use pointers, can map multiple values per key, and the data structure does not leak information about the stored values.

Invertible Bloom lookup table [10] and the Bloomier filter [3] have a similar motivation to ours but do not support multiple values per key and can return false negatives to the lookup queries which are not acceptable in our system model.

Encrypted Bloom filters are used in [9] and [2] to implement a secure inverted index, but the Bloom filter is used in a standard way to check if a particular keyword belongs to a particular document (hence the number of Bloom filters is equal to the number of documents). In our approach, we use a single data structure to get all the values assigned to a particular key.

"Searchable encryption" [1,5,7,8] is a related (yet orthogonal) topic to our research, as it is mainly focused on developing secure protocols and databases rather than basic data structures. Our data structure can be used as a building block for these schemes.

To the best of our knowledge, our approach is the first to use Bloom filters for privacy-preserving lookup table implementation without false-negative responses.

[1] Can be found in https://github.com/segevere/BFLUT.

5 Concluding Remarks

We have presented a basic data structure that implements a dynamic (supporting addition of elements) look-up table that is based on Bloom Filter. The scheme preserves (in post-quantum fashion) privacy, with no need to use (and store) encryption keys. Thus, multiple users can update the data structure even if they do not share a key.

Still, a user that would like to avoid guessing the participating keys may concatenate a fixed string (salt) to the keys prior to hashing.

Moreover, it can be used for deniability where, after the data structure is presented to the public, there is no definite proof (especially when the load factor is relatively high) that a certain element has been inserted into the BFLUT, as it returns with other possible addresses. In the case of encrypted Hash Table, the owner of the data will be requested to reveal the private key, and then the contents of the Hash Table revealed in clear the addressees of the inserted elements.

Note that the BFLUT can support mapping of several addresses to a (term or) key rather than one address for key, in such a scenario, a single search returns a set that includes all valid addresses that were inserted to the BFLUNT for the searched key.

References

1. Avni, H., Dolev, S., Gilboa, N., Li, X.: SSSDB: database with private information search. In: Karydis, I., Sioutas, S., Triantafillou, P., Tsoumakos, D. (eds.) ALGO-CLOUD 2015. LNCS, vol. 9511, pp. 49–61. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29919-8_4
2. Bellovin, S.M., Cheswick, W.R.: Privacy-enhanced searches using encrypted bloom filters. IACR Cryptol. ePrint Arch., p. 22 (2004)
3. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier filter: an efficient data structure for static support lookup tables. In: Ian Munro, J. (ed.) Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, 11–14 January 2004, pp. 30–39. SIAM (2004)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2022)
5. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
6. Dharmapurikar, S., Song, H., Turner, J.S., Lockwood, J.W.: Fast packet classification using bloom filters. In: Bhuyan, L.N., Dubois, M., Eatherton, W. (eds.) Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ANCS 2006, San Jose, California, USA, 3–5 December 2006, pp. 61–70. ACM (2006)
7. Fisch, B.A., et al.: Malicious-client security in blind seer: a scalable private DBMS. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, 17–21 May 2015, pp. 395–410. IEEE Computer Society (2015)

8. Benjamin Fuller, et al.: SoK: cryptographically protected database search. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 172–191. IEEE Computer Society (2017)
9. Goh, E.: Secure indexes. IACR Cryptol. ePrint Arch., p. 216 (2003)
10. Goodrich, M.T., Mitzenmacher, M.: Invertible bloom lookup tables. In: 49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011, Allerton Park & Retreat Center, Monticello, IL, USA, 28–30 September 2011, pp. 792–799. IEEE (2011)
11. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
12. Pontarelli, S., Reviriego, P., Mitzenmacher, M.: Improving the performance of invertible bloom lookup tables. Inf. Process. Lett. **114**(4), 185–191 (2014)
13. Weintraub, G., Gudes, E., Dolev, S.: Needle in a haystack queries in cloud data lakes. In: Costa, C., Pitoura, E. (eds.) Proceedings of the Workshops of the EDBT/ICDT 2021 Joint Conference, Nicosia, Cyprus, Volume 2841 of CEUR Workshop Proceedings, 23 March 2021. CEUR-WS.org (2021)

Author Index