Gerhard P. Hancke
Ernesto Damiani (Eds.)

LNCS 10741

Information Security Theory and Practice

11th IFIP WG 11.2 International Conference, WISTP 2017
Heraklion, Crete, Greece, September 28–29, 2017
Proceedings

Springer

Lecture Notes in Computer Science 10741

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

Editorial Board

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

Gerhard P. Hancke · Ernesto Damiani (Eds.)

Information Security Theory and Practice

11th IFIP WG 11.2 International Conference, WISTP 2017
Heraklion, Crete, Greece, September 28–29, 2017
Proceedings

Springer

Editors
Gerhard P. Hancke
City University of Hong Kong
Hong Kong
China

Ernesto Damiani
University of Milan
Milan
Italy

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-93523-2 ISBN 978-3-319-93524-9 (eBook)
https://doi.org/10.1007/978-3-319-93524-9

Library of Congress Control Number: 2018947334

LNCS Sublibrary: SL4 – Security and Cryptology

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The 11th WISTP International Conference on Information Security Theory and Practice attracted research contributions covering theoretical and practical aspects of security and privacy, especially for future ICT technologies. Technical concepts such as ambient intelligence, cyber-physical systems, and Internet of Things provide a vision of an information society in which: (a) people and physical systems are surrounded with intelligent interactive interfaces and objects, and (b) environments are capable of recognizing and reacting to the presence of different individuals or events in a seamless, unobtrusive, and invisible manner. The success of future ICT technologies will depend on how secure these systems are and to what extent they protect the privacy of individuals and individuals trust them.

In response to the call for papers, 35 papers were submitted to the conference from 20 different countries. Each paper was reviewed by at least three members of the Program Committee, and evaluated on the basis of its significance, novelty, and technical quality. The reviewing was double-blind, with the identities of the authors not revealed to the reviewers of the papers and the identities of the reviewers not revealed to the authors. The Program Committee's work was carried out electronically; each paper received at least three reviews followed by a Program Committee discussion to finalize decisions. Of the submitted papers, the Program Committee accepted eight full papers and four short papers. The technical contributions were presented in five technical sessions, and the program also included three invited talks by Prof. George Spanoudakis (City University London, UK), Prof. Fabio Martinelli (National Research Council of Italy, Italy), and Dr. Louis Marinos (ENISA, Greece).

We thank all authors and participants who contributed to make this event a great success, the Technical Program Committee members and additional reviewers who worked on the program, and the volunteers who handled aspects of the organization behind the scenes. We greatly appreciate the input from members of the WISTP Steering Committee, whose help and advice was invaluable, and the support of IFIP WG 11.2: Pervasive Systems Security. We would also like to thank the general chair, Ioannis Askoxylakis, and the other local organizers at FORTH-ICS for supporting for this event and providing assistance with general arrangements.

September 2017

Gerhard P. Hancke
Ernesto Damiani

Organization

General Chair

Ioannis Askoxylakis	FORTH-ICS, Greece

Program Chairs

Ernesto Damiani	Università degli Studi di Milano, Italy
Gerhard Hancke	City University of Hong Kong, Hong Kong, SAR China

Local Organizers

Nikolaos Petroulakis	FORTH-ICS, Greece
Andreas Miaoudakis	FORTH-ICS, Greece
Panos Chatziadam	FORTH-ICS, Greece

Steering Committee

Angelos Bilas	FORTH-ICS and University of Crete, Greece
Sara Foresti	Università degli Studi di Milano, Italy
Javier Lopez	University of Malaga, Spain
Konstantinos Markantonakis	ISG-SCC, Royal Holloway University of London, UK
Joachim Posegga	Institute of IT-Security and Security Law at the University of Passau, Germany
Jean-Jacques Quisquater	ICTEAM, Catholic University of Louvain, Belgium
Damien Sauveron	XLIM, University of Limoges, France

Program Committee

Mohamed Ahmed Abdelraheem	SICS, Swedish ICT, Sweden
Raja Naeem Akram	Royal Holloway, University of London, UK
Fahad Alharby	Naif Arab University for Security Sciences, Saudi Arabia
Claudio A. Ardagna	Università degli Studi di Milano, Italy
Ioannis Askoxylakis	FORTH-ICS, Greece
Hervé Chabanne	Morpho, France
Serge Chaumette	LaBRI, University of Bordeaux, France

Mauro Conti	University of Padua, Italy
José María De Fuentes	Universidad Carlos III de Madrid, Spain
Kurt Dietrich	NXP Semiconductors, Netherlands
Ruggero Donida Labati	Università degli Studi di Milano, Italy
Sara Foresti	Università degli Studi di Milano, Italy
Flavio Garcia	University of Birmingham, UK
Yong Guan	Iowa State University, USA
Julio Hernandez-Castro	University of Kent, UK
Michael Hutter	Cryptography Research, USA
Sushil Jajodia	George Mason University, USA
Süleyman Kardas	Batman University, Turkey
Mehmet Sabir Kiraz	TUBITAK Bilgem, Turkey
Andrea Lanzi	Università degli studi di Milano, Italy
Maryline Laurent	Institut Mines-Telecom, France
Albert Levi	Sabanci University, Italy
Tieyan Li	Huawei, Singapore
Javier Lopez	University of Malaga, Spain
Vashek Matyas	Masaryk University, Czech Republic
Sjouke Mauw	University of Luxembourg, Luxembourg
Nele Mentens	KU Leuven, Belgium
Alessio Merlo	University of Genoa, Italy
David Naccache	Ecole Normale Suprieure, France
Vladimir A. Oleshchuk	University of Agder, Norway
Joachim Posegga	University of Passau, Germany
Kai Rannenberg	Goethe University Frankfurt, Germany
Kouichi Sakurai	Kyushu University, Japan
Pierangela Samarati	Università degli Studi di Milano, Italy
Siraj Ahmed Shaikh	Coventry University, UK
Nils Tippenhauer	Singapore University of Technology and Design, Singapore
Denis Trcek	University of Ljubljana, Slovenia
Michael Tunstall	Cryptography Research, USA
Umut Uludag	TUBITAK Bilgem, Turkey
Anjia Yang	Jinan University, China
Stefano Zanero	Politecnico di Milano, Italy
Vincent Zhuang	City University of Hong Kong, SAR China

Additional Reviewers

Duygu Karaoglan Altop
Martin Gunnarsson
Chhagan Lal
Jiasi Weng
Christophe Petit
Davide Quarta
Madeline Cheah
Warren Connell
Yuto Nakano
Ming Li
Yasir Khan
Yanjiang Yang
Wenjie Yang
Partha Sarathi Roy
Sridhar Venkatesan
Giuseppe Cascavilla
Chunhua Su
Andreea-Ina Radu
Fei Xie

Sponsoring Institutions

CyberSure Project (http://www.cybersure.eu/)

Contents

Security in Emerging Systems

A Secure and Trusted Channel Protocol for UAVs Fleets

Raja Naeem Akram[1], Konstantinos Markantonakis[1], Keith Mayes[1], Pierre-François Bonnefoi[2], Amina Cherif[2,4], Damien Sauveron[2,3(✉)], and Serge Chaumette[3]

[1] Information Security Group Smart Card Centre, Royal Holloway, University of London, Egham, UK
{r.n.akram,k.markantonakis,keith.mayes}@rhul.ac.uk
[2] XLIM (UMR CNRS 7252/Université de Limoges), MathIS, Limoges, France
{pierre-francois.bonnefoi,damien.sauveron}@unilim.fr, amina.cherif@etu.unilim.fr
[3] LaBRI (UMR CNRS 5800/Université de Bordeaux), Talence, France
serge.chaumette@labri.fr
[4] LARI (Université Mouloud Mammeri de Tizi-Ouzou), Tizi-Ouzou, Algeria

Abstract. Fleets of UAVs will be deployed in near future in reliability and safety critical applications (*e.g.* for smart cities). To satisfy the stringent level of criticality, each UAV in the fleet must trust the other UAVs with which it communicates to get assurance of the trustworthiness in information received and to be sure not to disclose information to an unauthorized party. In addition, to be protected against an attacker willing to eavesdrop and/or modify the exchanged data, the communication channel needs to be secured, *i.e.* it has to provide confidentiality and integrity of exchanges. The work presented here is based on our previous research which concluded that it is required that each UAV includes a Secure Element (which we called ARFSSD standing for Active Radio Frequency Smart Secure Device) to withstand an adversary with a high attack potential. In this paper, we propose a secure and trusted channel protocol that satisfies the stated security and operational requirements for a UAV-to-UAV communication protocol. This protocol supports three main objectives: (1) it provides the assurance that all communicating entities can trust each other and can trust their internal (secure) software and hardware states; (2) it establishes a fair key exchange process between all communicating entities so as to provide a secure channel; (3) it is efficient for both the initial start-up of the network and when resuming a session after a cold and/or warm restart of a UAV. The proposed protocol is formally verified using CasperFDR and AVISPA.

1 Introduction

There are increasing number of application-areas that consider the usage of Unmanned Aerial Vehicles (UAVs), and specially of fleets of UAVs. It is thus of major importance to propose security mechanisms to provide strong guarantees

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 3–24, 2018.
https://doi.org/10.1007/978-3-319-93524-9_1

in terms of reliability, safety, privacy-protection and security. Regardless of the field of applications, whether military or civil, fleets of UAVs of course have to operate as planed and they thus should resist an adversary trying to tamper its reliability and safety, for instance by conducting attacks on the communication network between the UAVs to make them crash or misbehave. In addition, if in most civilian applications privacy-protection of end-users collected data is a required property for them to be accepted or certified-for-use, in military applications security (in term of confidentiality and integrity) of on-board data (*i.e.* the collected data and also the pre-loaded data) is a mandatory requirement.

As shown in [1] which dealt with adversary models for UAVs fleets, each UAV must be equipped with a Secure Element (SE) so as to withstand an adversary with a high attack potential. In this paper, based on presence of such SEs, we propose a secure and trusted channel protocol that satisfies the stated security and operational requirements for a UAV-to-UAV communication protocol.

1.1 Contribution

In this paper, our main goals are to propose a secure and trusted channel protocol for fleets of UAVs, and to compare its security and performance with similar protocols.

The salient contributions of this paper are the following:

1. proposed a Secure and Trusted Channel Protocol (STCP) to establish a secure channel between the communicating UAVs and to provide security assurance that each UAV is in the secure and trusted state;
2. defined comparison criteria for secure channel protocols along with the related security analysis;
3. validated the proposed protocol with a mechanical formal tools: CasperFDR and AVISPA.

1.2 Structure of the Paper

Section 2 briefly presents the domain of UAVs fleets, the associated security issues, how by providing the assurance to communication partners that the nodes are secure and trustworthy SEs can help to secure the fleet and the rational for a STCP. Section 3 discusses the existing work carried out in terms of UAV applications and secure channel protocols from a traditional computer security perspective. Section 4 discusses the proposed security comparison criteria and the proposed protocol for a SE-equipped UAVs fleet. In Sect. 5, before to formally analyze the proposed protocol using CasperFDR and AVISPA, we first compare it with different secure channel protocols of the related work based on the security comparison criteria previously defined. Finally in Sect. 6 we present future research directions and conclude the paper.

2 UAVs Fleet and Rationale for a STCP

A fleet of UAVs is composed of a set of small and light UAVs flying in swarm formation and collaborating together to achieve the entrusted mission. Each UAV is equipped with sensors which might be different of those of the other UAVs of the fleet. Additionally, for reliability reasons, there might be some redundant sensors. Since UAVs fleets can cover large geographic areas they are a possible replacement for regular big and expensive drones used in the past both in military and civilian applications. For instance, in the civilian applications such fleets of UAVs can be used for monitoring forest fires, searching missing people in avalanches, etc. As illustrated Fig. 1, to collaborate together, UAVs have to communicate. However if the recipient, application (3) running on UAV C, is not in the scope of the sender, the application (1) running on UAV A, the message must be routed like in a Mobile Ad hoc Network (MANet) by intermediary nodes (UAVs), here UAV B.

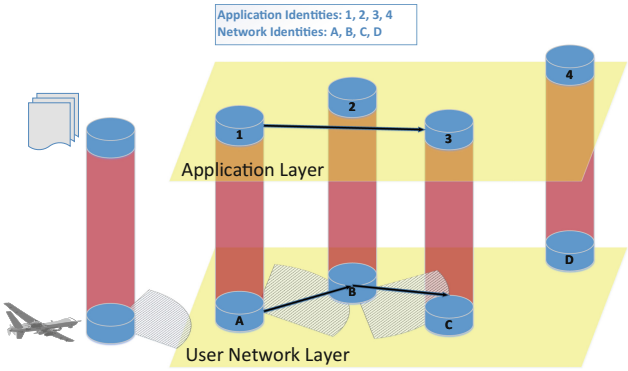

Fig. 1. Example of communication in UAVs fleet

2.1 Assets to Protect, Adversary Model and SE

Depending on the applications in which such a fleet is deployed and of the chosen adversary model, the assets to protect differ. In terms of information security, assets are the valuable data (here, data also includes the software application: intellectual property) of the owner of the UAVs fleet but are also valuable for attackers. Figure 2 depicts the general targets of an attacker on an individual UAV. There are different kinds of assets that might interest them, among which:

- pre-loaded data: *i.e.* flight-plan of the mission, cryptographic keys used to secure the communications, code of the applications running on the UAV, etc.
- collected data: *i.e.* photos, coordinates of points of interest (enemies or allies in case of a military application), etc.
- communication-related data: *i.e.* routing tables, session keys, etc.

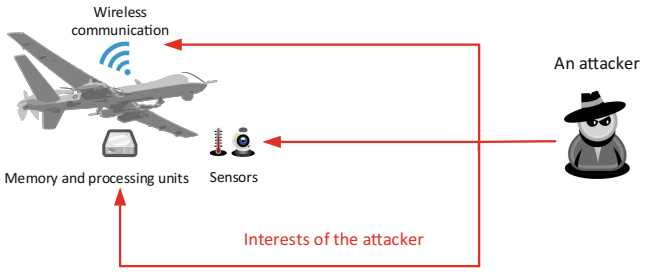

Fig. 2. Attacker interests

In our previous work [1], we have considered a strong adversary model with a high attack potential, *e.g.* the adversary has the capabilities and knowledge to capture a UAV in a functional state, to perform side-channel attacks or fault injections or other physical, software or combined attacks in order to gain access to (or to modify for his/her profit) some secret data (*e.g.* cryptographic keys), software or hardware. We have proposed a rationale which concluded that such a strong adversary model made sense, especially in the context of military usage of UAVs fleets, since the opponent can be a government-controlled organization capable of performing forensic analysis or attacks of the UAVs. Based on these conclusions we have derived the security and functional requirements and we have analyzed which one among several existing Secure Elements (*e.g.* Trusted Platform Module, active RFID, smart card) might be added to individual UAVs of the fleet to enhance the security up to the target assurance level.

Unfortunately none of them fulfilled all the criteria and we have proposed to use the one satisfying most of the requirements, i.e. the UCOM smart card (a smart card based on the User-Centric Ownership Model [2]), and to supplement it with the only missing feature which was the long range RF communication capability. This new SE, that we have named ARFSSD (Active Radio Frequency Smart Secure Device), is depicted Fig. 3.

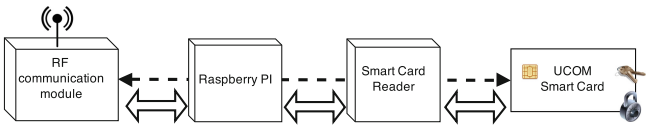

Fig. 3. Our implementation of the ARFSSD SE

We do not intend to detail this SE, still we have to explain how it is used to equip each UAV and what are its security features since we use them to propose in this paper the missing secure and trusted channel protocol required to secure UAVs fleets.

2.2 SE Usage and Its Security Features

As illustrated Fig. 4, a fleet of SE-equipped UAVs enables to build a control network layer between the SEs to provide high level of security for any exchanges in the upper network layers. This control network layer makes it possible to ensure that intermediary UAVs of the same fleet will not have access to the routed information (not even to the destination address if this is required for some privacy reasons). To ensure this kind of security properties, the destination address can be ciphered and the deciphering process can be done in the secure element which will decide if the message is for its own UAV or if it must be forwarded to another UAV of the fleet. Obviously the payload is also ciphered.

In addition, the SEs can also offer security services (like cryptography, secure storage and processing capabilities) to the application layer. In a UAVs fleet composed of UAVs belonging to several distinct owners or even in a UAVs fleet shared by several owners and running different applications. It might be possible to consider that some intermediary UAVs can be selfish. The presence of SEs enables to solve such an issue since the UAV itself is not aware of the routing decisions and some collaborative mechanisms can be also added (for instance based on reputation, or on retribution).

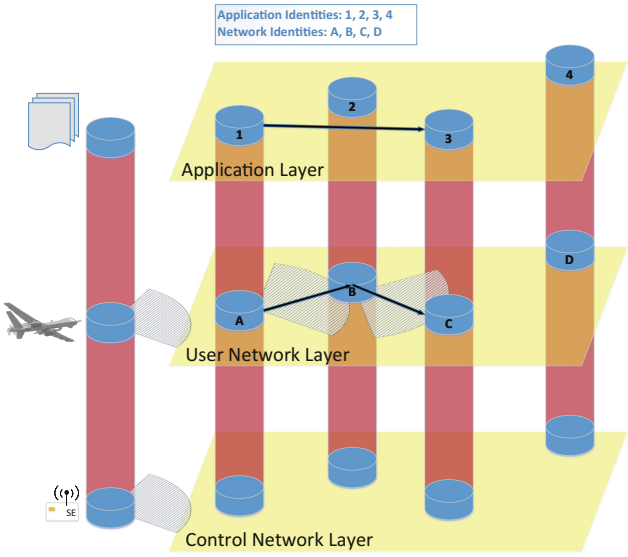

Fig. 4. Fleet of SE-equipped UAVs

In short, the SE depicted Fig. 4 can be defined as a long range RF-enabled UCOM smart card. One of the interests to have an SE equipped with long range communication capability is that the SEs altogether form an overlay network for specific control operations. This control network is parallel to the communication

network, called User Network, that already exists between UAVs. In this paper we will not detail the characteristics of the RF interface of the SEs (which should be different of those of UAV: *e.g.* the interface can use a different RF spectrum; the bandwidth can be smaller but the radio coverage can be larger). Our objective is to propose a protocol to establish a secure and trusted channel between these SEs which are the roots of trust and security for a UAVs fleet architecture withstanding an adversary with a high attack potential.

The overall architecture of a UCOM based smart card [2] is illustrated in Fig. 5. Basically it is a multiapplication smart card supplemented with new components. The most important for our protocol is the TEM (Trusted Environment & Execution Manager) which is represented as a layer between the smart card hardware and the runtime environment. This illustration provides a schematic view of the architecture and does not imply that all communications between the runtime environment and the hardware goes through the TEM.

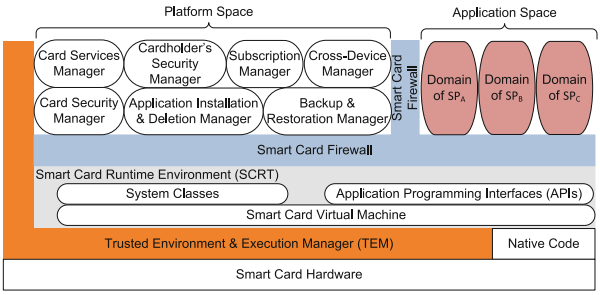

Fig. 5. UCOM smart card architecture

As depicted Fig. 6, the TEM is supporting the Trusted Computing Base (TCB) by providing several similar functionalities (usually present on TPM chip) which are useful for our protocol. The attestation handler and the self-test manager are the main building blocks that can be used to ensure a kind of trusted boot. More details on how the TEM is implemented and what are the roles of the different components are available in [3]. Basically, the self-test manager and the attestation handler can provide the assurance that the current

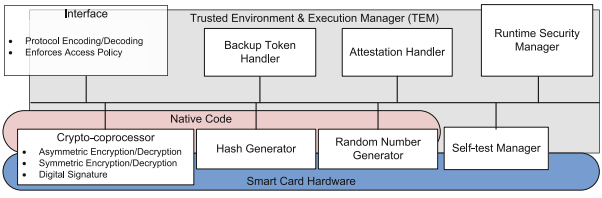

Fig. 6. Trusted platform module for smart card architecture

hardware and software state is secure as it was at the time of third party evaluation. This attestation process, called Platform State Verification/Assurance, can be requested by an internal entity on the card (*e.g.* an application) or by a remote party.

These functionalities of the TEM will be used in the secure and trusted channel protocol proposed in Sect. 4.

2.3 Rationale for a STCP

A Secure Channel Protocol (SCP) by definition provides either or both of entity authentication and key exchange between communicating parties (end points). A SCP preserves the confidentiality and integrity of the messages on the considered channel but not at the end points.

Nevertheless, there can be implicit assurance in the integrity and security of the end points as described by ETSI TS 102 412 [4] in the domain of the smart card industry. This document states that the smart card is a secure end point under the assumption that it is a tamper-resistant device. This type of assurance can be extrapolated to other devices that are implicitly trusted because of offline business relationships or because of a property of the device itself.

However, for a critical system like a fleet of UAVs it is not just implicit trust that is required but also explicit trust validation, to counter any potential threat. The explicit trust assurance should be provided by the UAVs (here the SE-equipped UAVs) that are participating in the communication of the UAVs fleet. This would build in an assurance that only secure and trusted devices (explicitly trusted devices with per-protocol run assurance) will participate in the UAVs fleet, potentially countering physically altered devices and/or re-introduction of a decommissioned device.

A trusted channel is a secure channel that is cryptographically bounded to the current state of the communicating parties [5]. This state can be a hardware and/or a software configuration, and ideally it requires a trustworthy component to validate that it is effectively as claimed. Such a component, in most instances, is a TPM [6] as demonstrated in [7–9].

Even though in a fleet of UAVs, individual devices will have prior relationships with each other (at least through the owners of the UAVs – in case of a multi-owners UAVs fleet, prior relationship must be defined as seen in Sect. 4.4), when establishing a secure channel, individual devices should still ensure that they are not only communicating with an authenticated device but also that the current state of this device is secure.

3 Related Work

In this section, we review the existing work in two different areas: UAVs fleet and Secure Channel Protocols (SCPs).

3.1 Related Work on Security Concerns of UAVs Fleets

This section describes different work related to UAV secure communications.

In [10], the authors proposed a secure channel protocol between individual UAVs and a ground station (GS). When a data communication with GS is possible, the UAVs send their collected data. To avoid that an attacker can retrieve plaintext data in case a UAV is captured, each block of data is ciphered with a one-time key generated by a key stream. This protocol is efficient to protect the confidentiality of sensed data. To protect against the forgery of messages, a tamper-resistant element (i.e. an SE) is required. In [11], the authors also proposed a protocol to secure communication between individual UAVs and a GS along with ensuring that an illegitimate access to sensed data is not easily available to an attacker. However the proposal is not as efficient as [10] due to the use Off-the-Record messaging to provide strong properties (*e.g.* deniability that does not able a GS to prove to other parties that a specific message was received from a specific UAV – this property is useful to protect journalism sourcing) which are useless in the context of UAVs fleet.

In [12], the authors presented their HAMSTER (HeAlthy, Mobility and Security based data communication archiTEctuRe) solution for unmanned vehicles. However the paper rather describes a security framework and cryptographic schemes than secure channel protocols. In the paper, they proposed some benchmark of ECC-based schemes (instead of RSA since ECC is more suitable for constrained devices) the performance of which was measured on a PC. However, the security of the private keys are not addressed.

In [13], the authors proposed SUAP, a secure reactive routing protocol the main cons of which is that it does not consider an adversary with a high attack potential. However, it is efficient to detect and prevent routing (*e.g.* wormhole, blackhole) attacks.

In [14], the authors proposed a secure communication protocol between UAVs and smart objects. If this is not exactly the same objective as that of our proposal, this paper was interesting since it took into account the capture of a UAV. The proposed protocol was based on efficient Certificateless Signcryption Tag Key Encapsulation Mechanism using ECC. However the solution does neither address smart objects capture nor the peer-to-peer communication mode of the UAVs fleet.

3.2 Related Work on Secure Channel Protocols

In this section, we restrict the discussion to the protocols that are proposed for general-purpose computing environments or to those that are used as references for comparison in the discussions to come.

The concept of trusted channel protocol was proposed by Gasmi et al. [5] along with the adaptation of the TLS protocol [15]. Later Armknecht et al. [8] proposed another adaptation of OpenSSL to accommodate the concept of trusted channels; similarly, Zhou and Zhang [7] also proposed a SSL-based trusted channel protocol.

In Sect. 5.2, we will compare the proposed STCP with the existing protocols. These protocols include the Station-to-Station (STS) protocol [16], the Aziz-Diffie (AD) protocol [17], the ASPeCT protocol [18], Just-Fast-Keying (JFK) [19], trusted TLS (T2LS) [5], GlobalPlatform SCP81 [20], the Markantonakis-Mayes (MM) protocol [21], and the Sirett-Mayes (SM) protocol [22].

This selection of protocols is intentionally broad so as to include well-established protocols like STS, AD and JFK. We also include the ASPeCT protocol, which is designed specifically for value-added services of mobile networks. Similar to our proposal where we require trust assurance during the protocol run, T2LS meets this as it provides trust assurance, whereas other protocols like SCP81, SM, and MM are specific to smart cards and are representative for embedded low-power devices. In addition, we have included the secure and trusted channel protocol, P-STCP [9], which is designed for resource-restricted and security-sensitive environments, and has some similar design requirements to those of the proposed protocol.

4 Secure and Trusted Channel Protocol

In this section, we begin the discussion with the security comparison criteria, followed by the protocol notation, the pre-setup and then the actual protocol proposal. This section concludes with a discussion of how the secure channel is re-established if one of the devices is restarted or resets the protocol.

4.1 Security Comparison Criteria

For a protocol to support the UAVs fleet, it should meet, at minimum, the security and operational requirements listed below:

(G1) ***Mutual Entity Authentication:*** All nodes in the network should be able to authenticate to each other to avoid masquerading by a malicious entity.
(G2) Asymmetric Architecture: Exchange of certified public keys between the entities to facilitate the key generation and entity authentication process must be provided.
(G3) Mutual Key Agreement: Communicating parties will agree on the generation of a key during the protocol run.
(G4) Joint *Key Control:* Communicating parties will mutually control the generation of new keys to prevent one party from choosing weak keys or predetermining any portion of the session key.
(G5) ***Key Freshness:*** The generated key will be fresh to the protocol session to protect against replay attacks.
(G6) Mutual *Key Confirmation:* Communicating parties will provide implicit or explicit confirmation that they have generated the same keys during a protocol run.
(G7) Known-Key Security: If a malicious user is able to obtain the session key of a particular protocol run, it should not enable him/her to retrieve long-term secrets (*private keys*) or *session keys* (future and past).

(G8) Unknown *Key* Share Resilience: In the event of an unknown key share attack, an entity $\mathcal{X}$ believes that it has shared a key with $\mathcal{Y}$, where the entity $\mathcal{Y}$ mistakenly believes that it has shared the key with entity $\mathcal{Z} \neq \mathcal{X}$. The proposed protocols should adequately protect against this attack.

(G9) *K*ey Compromise Impersonation (KCI) Resilience: If a malicious user retrieves the long-term key of an entity $\mathcal{Y}$, it will enable him/her to impersonate $\mathcal{Y}$. Nevertheless, key compromise should not enable him/her to impersonate other entities to $\mathcal{Y}$ [23].

(G10) *Perfect Forward Secrecy:* If the long-term keys of the communicating entities are compromised, this will not enable a malicious user to compromise previously generated session keys.

(G11) Mutual *Non-Repudiation:* Communicating entities will not be able to deny that they have executed a protocol run with each other.

(G12) Partial Chosen Key (PCK) Attack Resilience: Protocols that claim to provide joint key control are susceptible to this type of attack [24]. In this type of attack, if two entities provide separate values to the key generation function then one entity has to communicate its contribution value to the other. The second entity can then compute the value of its contribution in such a way that it can dictate its strength (*i.e.* it is able to generate a partially weak key). However, this attack depends upon the computational capabilities of the second entity. Therefore, the proposed protocols should adequately prevent PCK attack.

(G13) Trust Assurance (Trustworthiness): The communicating parties not only provide security and operation assurance but also validation proofs that are dynamically generated during the protocol execution.

(G14) Denial-of-Service (DoS) Prevention: The protocol should not require the individual nodes to allocate a large set of resources to the extent that it might contribute to a DoS attack.

(G15) Privacy: A third party should not be able to know the identities of the SE-equipped UAVs.

For a formal definition of the terms (italicized) used in the above list, the reader is referred to [25]. The requirements listed above are later used as a point of reference to compare the selected protocols in Table 3.

4.2 Protocol Notation

The notations used in the protocol description are listed in Table 1.

4.3 Pre-protocol Setup

The proposed protocol requires certain pre-protocol setup operations as listed below:

1. Each UAV that is part of the fleet has an ARFSSD, so called SE for concision reasons.

Table 1. Notations used in protocol description.

$SE1$	: Denotes an ARFSSD '1'.
$SE2$	: Denotes an ARFSSD '2'.
$A \rightarrow B$	: Message sent by an entity A to an entity B.
TEM_X	: Denotes the TEM of an entity X
X_i	: Represents the identity of an entity X.
g^{rx}	: Diffie-Hellman exponential generated by an entity X.
C_X	: Signature key certificate of an entity X.
N_X	: Random number generated by an entity X.
$X \| Y$	: Represents the concatenation of the data items X, Y in the given order.
$[M]_{K_a}^{K_e}$	: Message M is encrypted by the session encryption key K_e and then MAC is computed using the session MAC key K_a. Both keys K_e and K_a are generated during the protocol run.
$Sign_X(Z)$	: Signature generated on data Z by the entity X using a signature algorithm [26].
$H(Z)$	: Is the result of generating a hash of data Z.
$H_k(Z)$	: Result of generating a keyed hash of data Z using key k.
S_{Cookie}	: Session cookie generated by one of the communication entities. It indicates the session information and facilitates protection against DoS attacks along with (possibly) providing the protocol session resumption facility.
VR_{A-B}	: Validation request sent by entity A to entity B. In response entity B provides a security and reliability assurance to entity A.
SAS_{A-B}	: Security assurance generation by entity A that provides trust validation to the requesting entity B.

2. Each SE in the fleet is pre-configured with the signature verification keys of the owners of its partners (*i.e.* public keys of the owners of SE-equipped UAVs – note that all UAVs can have the same owner) to be able to verify the signature verification key of each SE contained in C_{SE}. Each owner willing his/her UAVs to take part of the fleet has to certify the SEs public key with his/her signature key and he/she has to provide his/her public key to the other owners willing to use his/her UAVs.
3. Each SE is also pre-configured with the signature verification keys of the certification body which has assessed the security of the SE and its TEM to be able to verify the signature verification key of each SE contained in $C_{TEM_{SE}}$. This certification of TEM public key is part of the UCOM smart card manufacturing process [27].
4. Each SE is also pre-configured with the security assurance values for the trusted and secure state of its communication partners.

One additional interest of our proposal is to support a multi-owner fleet of UAVs (UAVs can be owned by different parties). It is also possible to filter the UAVs authorized from "less" trusted owners by adding the SE identities attached to trusted UAVs.

4.4 Proposed Protocol

The messages of the protocol are listed in Table 2 and are described below.

Message 1. The SE1 generates a random number N_{SE1} and computes the Diffie-Hellman exponential $g^{r_{SE1}}$. The "$H(g^{r_{SE1}} \| N_{SE1} \| SE1_i \| SE2_i)$" serves as a session cookie "S_{Cookie}", and it is appended to each subsequent message sent by both devices. It indicates the session information, facilitates protection against DoS attacks and provides the protocol session resumption facility, which is required if a protocol run is interrupted before it successfully concludes. Finally, SE1 will request SE2 to provide assurance of its current state.

Table 2. Secure and trusted channel protocol (STCP).

1. $SE1 \rightarrow SE2 : SE1_i \| SE2_i \| N_{SE1} \| g^{r_{SE1}} \| VR_{SE1-SE2} \| S_{Cookie}$
2. $SE2 \rightarrow SE1 : SE2_i \| SE1_i \| N_{SE2} \| g^{r_{SE2}} \| [Sign_{SE2}(SE2 - Data) \| Sign_{TEM_{SE2}}(SE2 - Validation) \|$
   $: C_{SE2} \| C_{TEM_{SE2}}]_{K_a}^{K_e} \| VR_{SE2-SE1} \| S_{Cookie}$
   $: SE2 - Data = H(SE2_i \| SE1_i \| g^{r_{SE1}} \| g^{r_{SE2}} \| N_{SE1} \| N_{SE2})$
   $: SE2 - Validation = SAS_{SE2-SE1} \| N_{SE1} \| N_{SE2}$
3. $SE1 \rightarrow SE2 : [Sign_{SE1}(SE1 - Data) \| Sign_{TEM_{SE1}}(SE1 - Validation) \| C_{SE1} \| C_{TEM_{SE1}}]_{K_a}^{K_e} \| S_{Cookie}$
   $: SE1 - Data = H(SE1_i \| SE2_i \| g^{r_{SE2}} \| g^{r_{SE1}} \| N_{SE2} \| N_{SE1})$
   $: SE1 - Validation = SAS_{SE1-SE2} \| N_{SE2} \| N_{SE1}$

Message 2. In response, SE2 generates a random number, and a Diffie-Hellman exponential $g^{r_{SE2}}$. It can then calculate the $k_{DH} = (g^{r_{SE1}})^{r_{SE2}} \ (mod\ n)$ which will be the a shared secret from which the rest of the keys will be generated. The encryption key is generated as $K_e = H_{k_{DH}}(N_{SE1} \| N_{SE2} \| \text{“1”})$ and a MAC key as $K_a = H_{k_{DH}}(N_{SE1} \| N_{SE2} \| \text{“2”})$.

Subsequently, the TEM generates a state validation message signed by the TEM private key represented in the protocol as "$Sign_{TEM_{SE2}}(SE2 - Validation)$". SE2 will also request SE1 to provide assurance of its current state.

On receipt of this message, SE1 will first generate the session keys. It will then verify SE2's signature and validation proof generated by the TEM of SE2 after having verified that C_{SE2} and $C_{TEM_{SE2}}$ are genuine. As the signature key belongs to the TEM of SE2, an attacker cannot masquerade this signature. By verifying the signature, SE1 can ascertain the current state is measured by the TEM of SE2. Now SE1 can verify whether the security assurance value represents a trusted and secure state (or not) because since our pre-protocol setup, SE1 would have the security assurance value of a trusted and secure state of SE2.

Furthermore, SE1 will check the values of the Diffie-Hellman exponentials (*i.e.* $g^{r_{SE1}}$ and $g^{r_{SE2}}$) and of the generated random numbers to avoid man-in-the-middle and replay attacks.

Message 3. SE1 will then generate a message similar to message 2, a signature by SE1 and trust validation proof generated by its TEM.

On receipt of the message, SE2 will verify the two certificates, the trust validation proof and generate keys. It will also check the values of the Diffie-Hellman exponentials and of the generated random numbers to avoid man-in-the-middle and replay attacks.

4.5 Post-protocol Process

The shared material generated from the Diffie-Hellman exponential can be used to generate more keys than just the session encryption and MAC keys of the protocol. If this is not desirable then the session encryption and MAC keys can be saved as master session keys.

4.6 Protocol Resumption

The protocol is run the first time that two SE-equipped UAVs of the fleet have to communicate together. The session cookie, S_{Cookie} is used to facilitate the session resumption subsequent exchanges. However, based on a predefined policy (*e.g.* based on the elapsed time since the first protocol run), the SE might require the protocol to be executed again to refresh the master session keys.

5 Protocol Evaluation

In this section, we first discuss the information analysis of the protocols, and then compare different protocols with our proposal based on the comparison criteria defined above. Finally, we provide a formal analysis using CasperFDR and AVISPA.

5.1 Brief Informal Analysis

Throughout this section, we refer to the protocol comparison criteria of Sect. 4.1 by their respective numbers as listed in the same section.

During the proposed protocol, in messages 2 and 3 the communicating entities authenticate each other, which satisfies G1. Similarly, for G2, all communicating entities have exchanged cryptographic certificates to facilitate an authentication and trust validation proof (generated and signed by the TEM) before the SE-equipped UAVs are deployed (pre-deployment configuration).

The proposed protocol satisfies requirements G3, G4, G5 and G12 by first requiring SE1 and SE2 to generate the Diffie-Hellman exponential; computational cost is thus equal on both sides. Similarly, exponential generation also assures that both devices will have equal input to the key generation process. Messages 2 and 3 are encrypted using the keys generated during the protocol execution, thus providing mutual key confirmation (satisfying G6).

In the proposed protocol, session keys generated in one session have no link with the session keys generated in the other sessions, even when the session is established between the same devices. This enables the protocol to provide

resilience against known-key security (G7). This unlinkability of session keys is based on the fact that each entity not only generates a new Diffie-Hellman exponential but also a random number, both of which are used during the protocol for key generation. Therefore, even if an adversary $\mathcal{A}$ finds out about the exponential and random numbers of a particular session, it will not enable him/her to generate past or future session keys.

Furthermore, to provide unknown key share resilience (G8), the proposed protocol includes the Diffie-Hellman exponentials along with generated random numbers and each communicating entity then signs them. Therefore, the receiving entity can then ascertain the identity of the entity with which it has shared the key.

The protocol can be considered to be a KCI-resilient (G9) protocol, as protection against the KCI is based on the digital signatures. In addition, the cryptographic certificates of each signature key also include its association with a particular device. Therefore, if $\mathcal{A}$ has knowledge of the signature key of a device, it can only masquerade this particular device to other devices but not others to it.

Table 3. Protocol comparison on the basis of the stated goals (see Sect. 4.1)

Goals	Protocols											
	STS	AD	ASPeCT	JFK	T2LS	SCP81	MM	SM	P-STCP	SSH	SSL	Proposed protocol
G1	∗	∗	∗	∗	∗	∗	−∗	−∗	∗	(∗)	∗	∗
G2	∗	∗	∗	∗	∗	∗	∗	−∗	∗	∗	∗	∗
G3	∗	∗	∗	∗	∗	∗	∗	−∗	∗	∗	∗	∗
G4	∗	∗	∗	∗	(∗)	∗			∗	(∗)	(∗)	∗
G5	∗	∗	∗	∗	∗	∗	∗	−∗	∗	∗	∗	∗
G6	∗		∗	∗			∗	−∗	∗	∗	∗	∗
G7	∗	∗	∗	∗	∗	∗	∗		∗	∗	∗	∗
G8	∗	∗	∗	∗	∗	∗	∗	−∗	∗	∗	∗	∗
G9	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗
G10	∗		∗	∗	∗	∗			∗	∗	∗	∗
G11	∗			∗	∗	∗	∗	∗	∗	∗	∗	∗
G12	(∗)	(∗)	(∗)	(∗)	(∗)	(∗)			∗	∗	∗	∗
G13			(∗)	(∗)	∗	−∗			∗	(∗)	(∗)	∗
G14				∗	(∗)				∗	(∗)	(∗)	∗
G15	(∗)		∗	∗	(∗)				∗	(∗)	(∗)	∗

Note: ∗ means that the protocol meets the stated goal, (∗) shows that the protocol can be modified to satisfy the requirement, and – ∗ means that the protocol (implicitly) meets the requirement not because of the protocol messages but because of the prior relationship between the communicating entities.

The proposed protocol also meets the requirement for perfect forward secrecy (G10) by making the key generation process independent of any long-term keys.

The session keys are generated using fresh values of Diffie-Hellman exponentials and random numbers, regardless of the long term keys: they are signature keys. Therefore, even if eventually $\mathcal{A}$ finds out the signature key of any entity it will not enable him/her to determine past session keys. This independence of long term secrets from the session key generation process also enables the protocol to satisfy G7.

Communicating entities in the STCP share signed messages with each other that include the session information, thus providing mutual non-repudiation (G11). G14 is ensured by the inclusion in the protocol of the session cookie, which provides a limited protection against DoS, and by the fact that individual devices have pre-configurations of communication partners which enable them to drop a connection if an entity trying to connect with them is not able to authenticate.

To satisfy G15, the device identities are basically a random string that should not have any link with the function of the device. This would hinder an attacker from eavesdropping a protocol run to determine which SE is communicating on the wireless channel.

Finally, the TEMs of all SE-equipped UAVs provide trust validation proof signed by the TEM private key. This provides mutual validation of the trust between communicating devices, confirming that the other device is operating in a secure and reliable state (G13).

5.2 Revisiting the Requirements and Goals

Table 3 provides a comparison between the protocols listed in Sect. 3.2 and the proposed protocol in terms of the required goals (see Sect. 4.1).

As shown in Table 3, the STS protocol meets the first eleven goals. The main issue with the STS protocol is that it does not provide adequate protection against partial chosen key attacks (G12) and privacy protection (G15). The remaining goals are not met by the STS because of the design architecture and deployment environment, which did not require these goals. Similarly, the AD protocol does not meet G6, G10, G11 and G13–G15.

The ASPeCT and JFK protocols meet a large set of goals. Both of these protocols can easily be modified to provide trust assurance (requiring additional signatures). Both of these protocols are vulnerable to partial chosen key attacks. However, in Table 3 we opt for the possibility that the ASPeCT and JFK protocols can be modified to meet this goal because in a fleet of UAVs all communicating SE may be of the same computation power and have a strong offline pre-deployment relationship.

The T2LS protocol meets the trust assurance goal by default. However, for the remaining goals it has the same results as the SSL protocol. A point in favour of the SCP81, MM, and SM protocols is that they were designed for the smart card industry where there is a strong and centralised organisational model. Most of these protocols, to some extent, have a similar architecture, in which a server generates the key and then communicates that key to the client. There is no non-repudiation as they do not use signatures in the protocol run.

Both SSH and SSL meet a large set of requirements and also have the potential to be extended to satisfy the additional requirements. However, to provide a flexible, backward compatible and universally acceptable architecture these protocols have too many optional parameters. Such flexibility is one of the main causes of most of the issues that these protocols have been plagued with in the last couple of years, heartbleed being the most infamous vulnerability.

The only difference between the P-STCP and the proposed protocol (except for the message structure) is the number of rounds to successfully complete a protocols run. P-SCTP has four messages (2-round protocol) and the proposed protocol uses 3 messages (1.5-round protocol).

As can be seen from Table 3, the proposed protocol satisfies all goals that were described in Sect. 4.1.

5.3 Protocol Verification by CasperFDR and AVISPA

We selected the CasperFDR approach for formal analysis of the proposed protocol. The Casper compiler [28] takes as input a high-level description of the protocol, together with its security requirements along with the definition of an attacker and of its capabilities. The compiler then translates the description into the process algebra of Communicating Sequential Processes (CSP) [29]. The CSP description of the protocol can be machine-verified using the Failures-Divergence Refinement (FDR) model checker [30]. The intruder's capabilities modeled in the Casper script for the proposed protocol are as follows:

- an intruder can masquerade any entity in the network,
- an intruder can read the messages transmitted in the network, and
- an intruder cannot influence the internal process of an entity in the network.

The security specification for which CasperFDR evaluates the network is shown below. The listed specifications are defined in the #Specification section of the Casper script:

- the protocol run is fresh and both applications are alive,
- the key generated by the entity A is known only to the entity B (A and B are communication partners/devices),
- entities mutually authenticate each other and have mutual key assurance at the conclusion of the protocol,
- long-term keys of communicating entities are not compromised, and
- an intruder is unable to deduce the identities from observing the protocol messages.

The CasperFDR tool evaluated the protocol and did not find any feasible attack(s). The script is provided in Appendix A.

Similarly, we scripted the proposed protocol in to High-Level Protocol Specification Language (HLPSL), as protocol description language for Automated Validation of Internet Security Protocols and Applications (AVISPA) [31]. The

HLPSL is then translated into an intermediate language, which is an input to four different verifiers - as part of the AVISPA. These verifiers include SATMC, CL-AtSe, OFMC and TA4SP. Based on this analysis, no viable attack in the context of the operational environment of the protocol was found. The script is provided in Appendix B.

6 Conclusion and Future Research Directions

In this paper, we outlined the concept of fleets of UAVs and discussed why such an architecture requires a secure channel for communication. The data communicated over a UAVs fleet has a strong requirement for confidentiality and integrity. To satisfy this requirement, communicating devices should have some cryptographic secrets to provide confidentiality and integrity. To generate these cryptographic secrets, the SEs run a secure channel protocol. In this paper, we proposed a secure channel protocol that not only provides mutual authentications and key sharing between the communicating entities but also provides assurance that each of the devices is in a secure and trusted state. We compared our proposed protocol with a list of selected protocols. Finally, we evaluated our protocol using CasperFDR and AVISPA, showing that it is secure against a number of attacks.

In future work, we will explore the major issues of detecting and neutralising wireless jamming and DoS attackers, along with building a strong mitigating framework.

Acknowledgments. The authors from XLIM acknowledge the support of:

- the SFD (Security of Fleets of Drones) project funded by Région Limousin;
- the TRUSTED (TRUSted TEstbed for Drones) project funded by the CNRS INS2I institute through the call 2016 PEPS ("Projet Exploratoire Premier Soutien") SISC ("Sécurité Informatique et des Systèmes Cyberphysiques");
- the SUITED (Suited secUrIty TEstbed for Drones), SUITED2 and UNITED (United NetworkIng TEstbed for Drones), UNITED2 projects funded by the MIRES (Mathématiques et leurs Interactions, Images et information numérique, Réseaux et Sécurité) CNRS research federation.

The authors from LaBRI acknowledge the support of:

- the TRUSTED (TRUSted TEstbed for Drones) project funded by the CNRS INS2I institute through the call 2016 PEPS ("Projet Exploratoire Premier Soutien") SISC ("Sécurité Informatique et des Systèmes Cyberphysiques");
- the SUITED-BX, SUITED2-BX and UNITED-BX, UNITED2-BX projects funded by LaBRI and its MUSe team.

Appendix A CasperFDR Script

```
#Free variables
datatype Field = Gen | Exp(Field, Num) unwinding 2
hkSE2, hkSE1, iMsg, rMsg, EnMaKey : Field
SE1, SE2, U: Agent
```

```
gSE1, gSE2: Num
nSE1, nSE2, SE1Val, SE2Val: Nonce
VKey: Agent->PublicKey
SKey: Agent->SecretKey
InverseKeys = (VKey, SKey), (EnMaKey, EnMaKey), (Gen, Gen), (Exp, Exp)

#Protocol description
0.    -> SE2 : SE1 [SE1!=SE2] <iMsg := Exp(Gen,gSE2)>
1. SE2 -> SE1 : SE2, nSE2, iMsg%hkSE2 <EnMaKey := Exp(hkSE2, gSE1); rMsg :=
Exp(Gen,gSE1)>
2. SE1 -> SE2 : nSE1, rMsg%hkSE1 <EnMaKey := Exp(hkSE1, gSE2)>
3. SE2 -> SE1 : nSE2, nSE1
4. SE1 -> SE2 : {{rMsg, U, nSE2}{SKey(U)}}{EnMaKey} [rMsg==hkSE2]
5. SE2 -> SE1 : {{iMsg,SE2, nSE1}{SKey(SE2)}}{EnMaKey} [iMsg==hkSE1]
6. SE1 -> SE2 : {{SE1OSHash, SE1, nSE2}{SKey(SE1)}}{EnMaKey}

#Actual variables
ADev1, ADev2, ME: Agent
GSE1, GSE2, GMalicious: Num
NSE1, NSE2, SE1VAL, SE2VAL, NMalicious: Nonce

#Processes
INITIATOR(SE2, SE1, U, SE2VAL, gSE2, nSE2) knows SKey(SE2), VKey
RESPONDER(SE1, SE2, U, SE1VAL, gSC, nSC) knows SKey(U), SKey(SC), VKey

#System
INITIATOR(ADev2, ADev1, ADev2Val, GSE2, NSE2)
RESPONDER(ADev1, ADev2, ADev1Val, GSE1, NSE1)

#Functions
symbolic VKey, SKey

#Intruder Information
Intruder = ME
IntruderKnowledge = {ADev2, ADev2, ME,
GMalicious, NMalicious, SKey(ME), VKey}

#Specification
Aliveness(SE2, SE1)
Aliveness(SE1, SE2)
Agreement(SE2, SE1, [EnMaKey])
Secret(SE2, EnMaKey, [SE1])
Secret(SE1, U, [SE2])

#Equivalences
forall x, y : Num . Exp(Exp(Gen, x), y) = Exp(Exp(Gen, y), x)
```

Appendix B AVISPA Script

```
role   se_1 (A,B: agent,
             G : text,
      PK1,PKTM1: public_key,
    CSE1,CTMSE1: message,
 H,Hk, MAC, SIGN: hash_func,
      SND, RCV : channel(dy))
played_by A
     def=
        local   NS1, NS2, Rs1, Rs2          : text,
                State                       : nat,
                VR1, VR2                    : text,
                SAS1, SAS2, Sdata1, Sdata2  : text,
                Success                     : text,
                Svalid1, Svalid2            : text.text.text,
                Kdh, Ke, Ka                 : message,
                PK2, PKTM2                  : public_key,
                Scookie                     : message,
                CSE2, CTMSE2                : message
       const    sec_kdh1, sec_ke1, sec_ka1  : protocol_id
       init     State:= 0
transition
```

```
1.   State=0 /\ RCV(start) =|>
                State':=2 /\ NS1':= new() /\ Rs1' := new()
                          /\  SND(A.B.NS1'.exp(G, Rs1')).VR1.Scookie)
2.   State=2 /\ RCV(B.A.NS2'.exp(G, Rs2').Sdata2'.Svalid2'.
            {SIGN(Sdata2')}_(inv(PK2')).{SIGN(Svalid2')}_(inv(PKTM2')).
            MAC(Ka'.{{SIGN(Sdata2')}_(inv(PK2')).
            {SIGN(Svalid2')}_(inv(PKTM2')).CSE2'.CTMSE2'}_Ke')
            .VR2.Scookie) =|>
     State':= 4   /\  SAS1':= new()
                  /\ Svalid1':= SAS1'.NS2'.NS1
                  /\  Sdata1':= H(A.B.exp(G, Rs2').exp(G, Rs1).NS2'.NS1)
                  /\ Kdh' := exp(exp(G, Rs2'), Rs1)
                  /\ Ke':= {Hk(NS1.NS2'.1)}_Kdh'
                  /\ Ka':= {Hk(NS1.NS2'.2)}_Kdh'
                  /\ SND(A.B.Sdata1'.Svalid1'.{SIGN(Sdata1')}_(inv(PK1)).
                     {SIGN(Svalid1')}_(inv(PKTM1)).MAC(Ka'.
                     {{SIGN(Sdata1')}_(inv(PK1)).
                     {SIGN(Svalid1')}_(inv(PKTM1)).CSE1.CTMSE1}_Ke').Scookie)
                  /\ waitness ( A,B, ns1, NS1)
3. State= 4 /\ RCV(Success') =|>
              State':=6       /\ request (A,B, ns2, NS2)
                              /\ secret (Kdh,sec_kdh1, {A,B})
                              /\ secret(Ke, sec_ke1, {A,B})
                              /\ secret(Ka, sec_ka1, {A,B})
end role
role   se_2 (B,A: agent,
               G : text,
       PK2,PKTM2: public_key,
     CSE2,CTMSE2: message,
     H,Hk, MAC, SIGN: hash_func,
           SND, RCV : channel(dy))
played_by B
     def=
        local   NS2, NS1, Rs2, Rs1              : text,
        State                                   : nat,
          VR2, VR1                              : text,
          SAS2, SAS1, Sdata2, Sdata1            : text,
          Success                               : text,
          Svalid2, Svalid1                      : text.text.text,
          Kdh, Ke, Ka                           : message,
          PK1, PKTM1                            : public_key,
          Scookie                               : message,
          CSE1, CTMSE1                          : message
          const     sec_kdh2, sec_ke2, sec_ka2  : protocol_id
          init      State:= 1
transition
1.          State=1 /\ RCV(A.B.NS1'.exp(G, Rs1'). VR1.Scookie) =|>
            State':= 3 /\ NS2':= new()    /\ Rs2':=new()   /\  SAS2':= new()
                       /\ Sdata2':= H(B.A.exp(G, Rs1').exp(G, Rs2'). NS1'.NS2')
                       /\  Svalid2':= SAS2'.NS1'.NS2'
                       /\   Kdh' := exp(exp(G, Rs1'), Rs2')
                       /\ Ke' := {Hk(NS1.NS2'.1)}_Kdh'
                       /\ Ka' := {Hk(NS1.NS2'.2)}_Kdh'
                       /\ SND (B.A.NS2'.exp(G, Rs2'). Sdata2'.Svalid2'.
                       {SIGN(Sdata2')}_(inv(PK2)). {SIGN(Svalid2')}_(inv(PKTM2))
                       .MAC(Ka'.{{SIGN(Sdata2')}_(inv(PK2)).
                       {SIGN(Svalid2')}_(inv(PKTM2)).CSE2.CTMSE2}_Ke').VR2.Scookie)
                       /\ waitness (B,A, ns2, NS2')
2.        State=3 /\ RCV  (A.B.Sdata1'.Svalid1'.{SIGN(Sdata1')}_(inv(PK1')).
                  {SIGN(Svalid1')}_(inv(PKTM1')).MAC(Ka.{{SIGN(Sdata1')}_(inv(PK1')).
                  {SIGN(Svalid1')}_(inv(PKTM1')).CSE1'.CTMSE1'}_Ke).Scookie)
=|>
             State':= 5 /\  request(B,A,ns1, NS1)
                        /\ SND(Success')
                        /\ secret (Kdh,sec_kdh2, {B,A})
                        /\ secret(Ke, sec_ke2, {B,A})
                        /\ secret(Ka, sec_ka2, {B,A})
end role
role session (A,B: agent,
              G: text,
           H,Hk, MAC, SIGN: hash_func)
def=
    local         SA, RA, SB, RB :     channel(dy),
       CSE1, CSE2, CTMSE1, CTMSE2: message,
           PK1, PK2, PKTM1, PKTM2:  public_key
    composition
    se_1(A,B, G, PK1, PKTM1, CSE1, CTMSE1, H,Hk, MAC, SIGN, SA, RA)
       /\    se_2(B, A, G, PK2, PKTM2, CSE2, CTMSE2, H, Hk, MAC, SIGN, SB,
RB)
end role
role environment( )  def=
```

```
        const   ns1, ns2                : protocol_id,
                a, b                    : agent,
                pk1, pk2, pki           : public_key,
                g                       : text,
                h, hk, mac, sign        : hash_func
        intruder_knowledge = {a, b, i, pk1, pk2, pki, inv(pki), g, h, hk,
mac, sign}
        composition
                session (a ,b, g, h, hk, mac, sign)
              /\  session (a ,i , g, h, hk, mac, sign)
              /\  session (i ,b , g,  h, hk, mac, sign)
end role
goal
secrecy_of    sec_kdh1,   sec_kdh2 , sec_ke1,    sec_ke2,    sec_ka1,    sec_ka2
authentication_on    ns1
authentication_on    ns2
end goal
environment ()
```

References

1. Akram, R.N., Bonnefoi, P.F., Chaumette, S., Markantonakis, K., Sauveron, D.: Secure autonomous UAVs fleets by using new specific embedded secure elements. In: 2016 IEEE TrustCom/BigDataSE/ISPA, pp. 606–614, August 2016
2. Akram, R.N., Markantonakis, K., Mayes, K.: A paradigm shift in smart card ownership model. In: 2010 International Conference on Computational Science and Its Applications, pp. 191–200, March 2010
3. Akram, R.N., Markantonakis, K., Mayes, K.: Trusted platform module for smart cards. In: 2014 6th International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5, March 2014
4. Smart Cards; Smart Card Platform Requirements Stage 1 (Release 9), European Telecommunications Standards Institute (ETSI), France, Technical Specification ETSI TS 102 412 (V9.1.0), June 2009. http://www.etsi.org/deliver/etsi_ts/102400_102499/102412/09.01.00_60/ts_102412v090100p.pdf
5. Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., Asokan, N.: Beyond secure channels. In: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, STC 2007, pp. 30–40. ACM, New York (2007)
6. Trusted Platform Module Main Specification, Trusted Computing Group, Technical report (2011)
7. Zhou, L., Zhang, Z.: Trusted channels with password-based authentication and TPM-based attestation. In: International Conference on Communications and Mobile Computing, vol. 1, pp. 223–227 (2010)
8. Armknecht, F., Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., Ramunno, G., Vernizzi, D.: An efficient implementation of trusted channels based on OpenSSL. In: Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, ser. STC 2008, pp. 41–50. ACM, New York (2008)
9. Akram, R.N., Markantonakis, K., Mayes, K.: A privacy preserving application acquisition protocol. In: Min, G., Marmol, F.G. (eds.) 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 2012), Liverpool, United Kingdom. IEEE Computer Society, June 2012
10. Blazy, O., Bonnefoi, P.-F., Conchon, E., Sauveron, D., Akram, R.N., Markantonakis, K., Mayes, K., Chaumette, S.: An efficient protocol for UAS security. In: 2017 Integrated Communications Navigation and Surveillance (ICNS) (2017)

11. Steinmann, J.A., Babiceanu, R.F., Seker, R.: UAS security: encryption key negotiation for partitioned data. In: 2016 Integrated Communications Navigation and Surveillance (ICNS), pp. 1E4-1–1E4-7, April 2016
12. Pigatto, D.F., Gonçalves, L., Roberto, G.F., Rodrigues Filho, J.F., Floro da Silva, N.B., Pinto, A.R., Lucas Jaquie Castelo Branco, K.R.: The HAMSTER data communication architecture for unmanned aerial, ground and aquatic systems. J. Intell. Robot. Syst. **84**(1), 705–723 (2016). https://doi.org/10.1007/s10846-016-0356-x
13. Maxa, J.A., Mahmoud, M.S.B., Larrieu, N.: Extended verification of secure UAANET routing protocol. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–16, September 2016
14. Won, J., Seo, S.-H., Bertino, E.: A secure communication protocol for drones and smart objects. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ser. ASIA CCS 2015, pp. 249–260. ACM, New York (2015). https://doi.org/10.1145/2714576.2714616
15. Dierks, T., Rescorla, E.: RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2, Technical report, August 2008
16. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Crypt. **2**(2), 107–125 (1992)
17. Aziz, A., Diffie, W.: Privacy and authentication for wireless local area networks. IEEE Pers. Commun. **1**, 25–31 (1994)
18. Horn, G., Preneel, B.: Authentication and payment in future mobile systems. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 277–293. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055870
19. Aiello, W., Bellovin, S.M., Blaze, M., Canetti, R., Ioannidis, J., Keromytis, A.D., Reingold, O.: Just fast keying: key agreement in a hostile internet. ACM Trans. Inf. Syst. Secur. **7**, 242–273 (2004)
20. Remote Application Management over HTTP, Card Specification v 2.2 - Amendment B. GlobalPlatform Specification, September 2006
21. Markantonakis, K., Mayes, K.: A secure channel protocol for multi-application smart cards based on public key cryptography. In: Chadwick, D., Preneel, B. (eds.) CMS 2004. ITIFIP, vol. 175, pp. 79–95. Springer, Boston, MA (2005). https://doi.org/10.1007/0-387-24486-7_6
22. Sirett, W.G., MacDonald, J.A., Mayes, K., Markantonakis, K.: Design, installation and execution of a security agent for mobile stations. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 1–15. Springer, Heidelberg (2006). https://doi.org/10.1007/11733447_1
23. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0024447
24. Mitchell, C., Ward, M., Wilson, P.: Key control in key agreement protocols. Electron. Lett. **34**(10), 980–981 (1998)
25. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton (1996)
26. Furlani, C.: FIPS 186–3: Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST) Std., June 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf
27. Akram, R.N., Markantonakis, K., Mayes, K.: A dynamic and ubiquitous smart card security assurance and validation mechanism. In: Rannenberg, K., Varadharajan, V., Weber, C. (eds.) SEC 2010. IAICT, vol. 330, pp. 161–172. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15257-3_15

28. Lowe, G.: Casper: a compiler for the analysis of security protocols. J. Comput. Secur. **6**, 53–84 (1998)
29. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
30. Ryan, P., Schneider, S.: The Modelling and Analysis of Security Protocols: The CSP Approach. Addison-Wesley Professional, Boston (2000)
31. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_27

Philanthropy on the Blockchain

Danushka Jayasinghe(✉), Sheila Cobourne, Konstantinos Markantonakis, Raja Naeem Akram, and Keith Mayes

Smart Card and IoT Security Centre, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
{Danushka.Jayasinghe.2012,Sheila.Cobourne.2008}@live.rhul.ac.uk,
{K.Markantonakis,R.N.Akram,Keith.Mayes}@rhul.ac.uk

Abstract. One of the significant innovations that came out of Bitcoin is the blockchain technology. This paper explores how the blockchain can be leveraged in the philanthropic sector, through charitable donation services in fiat currency or Bitcoin via a web-based donor platform. The philanthropic model is then used for a case study providing humanitarian aid for a community living in a challenging geographical environment with limited internet availability. An SMS based mobile payment system is proposed for provisioning the received donations using the existing GSM network, very basic mobile phones and One Time Password (OTP) security tokens. The proposed scheme is finally evaluated for security while discussing the impact it has on charities and donors.

Keywords: Blockchain · Bitcoin · Rootstock · Philanthropy
Smart contract · Multi-signatures · Hosted wallet · SMS · Charity
OTP · Security token

1 Introduction

Bitcoin is a decentralised cryptocurrency system which works on a peer-to-peer network, using blockchain technology [8,18]. Blockchain technology has gained rapid interest due to its decentralised nature and strong security properties [22,23]. However, blockchains are not limited to decentralised cryptocurrencies, but can also be applied to other innovative ideas such as smart contracts, recording asset ownerships, cross-border payment solutions, trade finance, etc. [21,22].

A report by the UK Charities Aid Foundation [10] identifies that for charities, blockchains can increase transparency, openness and trust whilst reducing transaction costs and providing new opportunities for fundraising. We explore this by introducing a generic blockchain based philanthropic model that uses a web-based donation platform where donors can choose which charity projects to support, through donations in Bitcoin or fiat currency [17]. In the proposed scheme, the charity will maintain hosted Bitcoin Wallets for registered users: back-end payments are done via multi-signature Bitcoin transactions to enhance security: more advanced services can be offered using Smart Contracts via the

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 25–38, 2018.
https://doi.org/10.1007/978-3-319-93524-9_2

Rootstock platform [2]. This allows the charity to provide feedback on how each individual donation was used (donation transparency), along with secure, cheap and speedy transactions and an infrastructure that can be used for donation provisioning. This generic philanthropic model is then applied to financial aid distribution in offline geographical environments such as warzones, disaster areas and economically deprived areas of the world where the basic technological infrastructure necessary for a blockchain solution is not available. In these challenging conditions, conventional internet-based money transfer may not be possible and physical cash handling may be fraught with danger. Using our solution, Bitcoin payments can be done using basic mobile phones, an SMS mobile payment system utilising an existing GSM network and low-cost security tokens.

The main contributions of the paper are: (1) a new philanthropic model that leverages the Bitcoin blockchain/Smart Contract Platform for charitable donations/provisioning and (2) an SMS based Bitcoin mobile payment system that can be used in an offline environment. The paper is structured as follows. Benefits of Blockchains are identified in Sect. 2. Our philanthropic model is introduced in Sect. 3, and in Sect. 4, this is applied to a use case of humanitarian aid in a disconnected environment. The proposed scheme is evaluated in Sect. 5, and the discussion is concluded and future research directions outlined in Sect. 6.

2 Benefits of Blockchain Solutions for Charities/Donors

Blockchain solutions can provide advantages when used in conjunction with charitable giving [10,11,15]. For example, the Royal National Lifeboat Institution (RNLI) in the UK has accepted Bitcoin since August 2015 [7]. BitGive Foundation's (GiveTrack) allows donors to trace Bitcoin transactions (donations) in real time [5] Bitcoin solutions can benefit both charities and donors as follows:

1. Donation transparency: by using Bitcoin addresses for donations there is a publicly available audit trail detailing exactly where a particular donation went.

2. Reducing transaction costs: Low International transaction fees are a feature of blockchain payments as seen in Table 1. Bitcoin Currency (BTC), United States Dollar (USD), Great British Pound (GBP).

Table 1. Comparison of transaction fees

Transaction method	Fee BTC	Fee USD	Fee GBP	Speed
Bitcoin (average 645 bytes)[a]	0.0001	$0.25	£0.20	Roughly 50 min [9]
Western Union[b]	-	$14.83	£8.90	Less then 1 h
Western Union[b]	-	$11.50	£6.90	Next day
MoneyGram[b]	-	$16.50	£9.90	Less then 1 h
Ria[b]	-	$10.00	£6.00	Same day

[a]Bitcoin transaction fees are calculated on transaction size, not monetary value [8].
[b]Based on remittance transfer of 120 GBP from the United Kingdom to Uganda [20].

3. Donation speed: All Bitcoin transactions are broadcast immediately. Each transaction that is included in a valid mined block and added to the blockchain is called a confirmation which takes just over seven minutes [8,9]. With each subsequent block mined, the number of confirmations for that particular transaction increases by one. It is common practice to wait until at least six confirmations [18], taking roughly fifty minutes [9]. Some transactions could be considered to be complete after only one or two confirmations. This is fast compared to existing methods which could take several days [23]: see Table 1.

4. Donation provisioning: Provisioning the donations to beneficiaries can be challenging. For example, humanitarian financial aid distribution in warzones can be hindered if the country's banking system is subject to sanctions. Bitcoin payments bypass the banking system and donations can reach their intended target, without requiring the charity to transport large amounts of cash [10].

3 The Blockchain Philanthropic Model

We propose a system where a donor can make their donation in Bitcoin via a Donor Platform. Each charity/project on the donor platform has a Bitcoin address, with the 'granularity' ranging from one Bitcoin address per project through to a central Bitcoin address for the charity. These standalone Bitcoin addresses can be funded by donors using a standard pay to address (Pay To Public Key Hash) Bitcoin transaction. Bitcoin donors can use any Bitcoin wallet/client to donate, or use fiat currency that gets converted to Bitcoin automatically by using an online exchange. Once a donation is made, the donor can query the blockchain to see whether the donated funds have been used or not. The charity then uses the donations to allocate financial aid to individual beneficiaries. Beneficiaries can then perform Bitcoin transactions for day-to-day activities.

3.1 Bitcoin Transaction Methods

We propose that donations can be used by the charity for donation provisioning and subsequent SMS payment processing via one of two Bitcoin payment methods: Multi-Signature Addresses and Smart Contracts.

Option 1: Multi-Signature Addresses are derived using a multi-signature process, where more than one private key is needed to authorise a transaction. For example, a 2-of-3 multi-signature is when a Bitcoin address is associated with three private keys and at least two out of the three private keys are needed to authorise a Bitcoin transaction. In our proposal, we use 'Pay To Script Hash' (P2SH) transactions to process multi-signatures. To generate a multi-signature, a Full Redeem Script which includes details of the three public keys is hashed to generate a hashed Redeem Script which becomes the P2SH multi-signature. The Full Redeem Script is shared between all key-holding entities. The Redeem Script can be used to verify the transferred amount and whether its being sent

to the correct multi-signature address. It also gives details about how many signatures are needed to make a payment. The recipient needs to provide the full redeem script to spend the received Bitcoins.

Option 2: Smart Contracts can be defined as a set of instructions represented in computer code published on a distributed network, that receives inputs, executes instructions and provides outputs. It can enable a charity to offer additional features such as: routine provisioning of donations when beneficiaries are low in cash, issuance of small micro-finance loans, record keeping, donation requests to donors and automatic audit reports of a charity activity. Running advanced smart contracts on the Bitcoin network is not possible, however, a suitable platform would be Rootstock (RSK) [2], which is a sidechain that is based on a 2-Way peg mechanism. The 2-Way peg is a method to convert BTC into Smart Bitcoin Currency (SBTC) and vice-versa. When a user intends to convert BTC to SBTC, some BTC are locked in Bitcoin and the same amount of SBTC is unlocked in RSK and vice-versa [2]. Provisioning of donations, payments between beneficiaries and transaction fees to execute instructions are paid using SBTC. The smart contract is then published in the RSK network i.e. the contract exists on every node joining the network, including miners. To execute an instruction in the smart contract, the charity broadcasts a message to the RSK network. A small transaction fee is paid for this process ("Gas"). A smart contract can be instructed to receive two or more signatures (similar to multi-signature functionality) via programmable logic before a transaction can be executed and broadcast to the peer-to-peer network.

4 The Philanthropic Model in an Offline Environment

Blockchain based schemes have constraints, such as requiring Internet and compatible devices: computers, tablets or smart phones that can perform cryptographic processes. People in a geographical area without reliable Internet facility would find it difficult to use the hosted wallet based transactions. There is more GSM network coverage than Internet access in most countries around the world [16], and the use of mobile phones within the GSM network coverage is considerably higher compared to other communication technologies [16], so this points us to consider an Short Message Service (SMS)-based solution.

4.1 SMS Payments and Bitcoin

SMS m-payment systems have been extremely successful in the developing world, most notably M-PESA in Kenya [6]. The SMS approach has been extended to perform Bitcoin transactions [1,3]. However, all these schemes require the user to have initial online access to set up and maintain their Wallet. Attempts to integrate Bitcoin directly with M-PESA in Kenya have largely been unsuccessful due to business pressures [24]. Other proposals need smartphone apps to interact with online Bitcoin wallets e.g. BTC Wallet [12]. As none of these existing solutions is suitable, we propose a novel SMS based mobile payment system which

acts as a gateway to transact with the blockchain, using Bitcoin wallets hosted on beneficiaries' behalf by the charity (Hosted Wallet). Offline beneficiaries can then make and receive Bitcoin payments using SMS messaging on basic mobile phones along with a One Time Password (OTP) security token, that provides some assurance that only a genuine user can send an SMS to make a transaction.

4.2 Security Requirements and Adversarial Model

The proposed scheme must satisfy the following security requirements. **Confidentiality**: sensitive information should not be disclosed to unauthorised parties. **Integrity**: information must not be tampered with by unauthorised parties. **Authentication**: all participants in a transaction must be authorised and all transaction data must be genuine. **Non-repudiation**: none of the participants in a transaction can subsequently deny taking part in it. **Availability**: services should not be denied to authorised users (distributed denial of service - DDoS).

In a humanitarian aid setting, the adversarial model is as follows [14]. **State Level Attackers (SL):** high levels of skill/resources, employed by government agencies to attack commercial/government systems. State sponsored cyber attacks on humanitarian operations have been recorded. **Cyber Criminals (CC):** are organised groups who attack systems for money, who also have

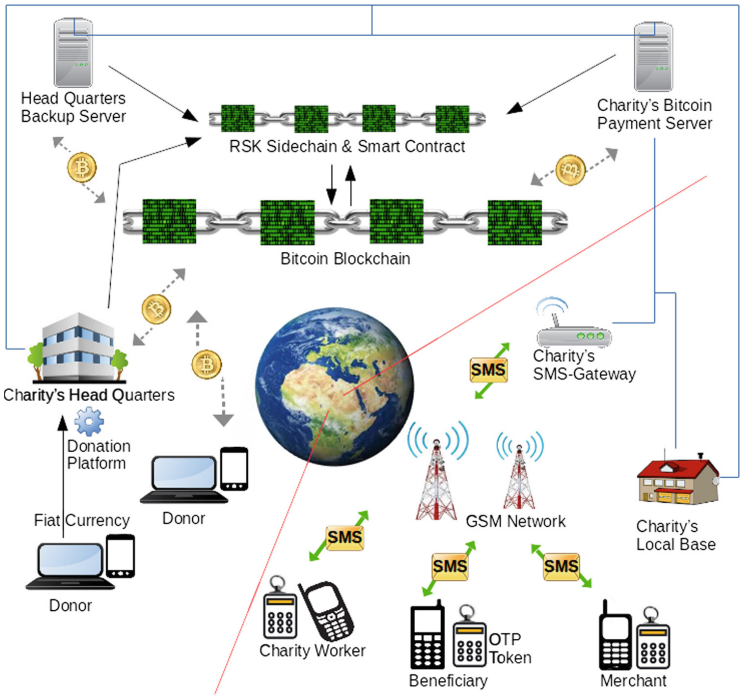

Fig. 1. Philanthropic model and SMS payment system architecture

high levels of skill and resources. **Hacktivists (Ha):** have moderate skills and resources and use digital tools to mount attacks for ideological reasons. **Insiders (In):** may have low levels of technical skill and resources, corrupt users, charity workers or merchants can be particularly dangerous if they have privileged access to data.

4.3 Proposed SMS-Based Bitcoin Payment Scheme

The charity creates Hosted Wallets for beneficiaries, and during a secure registration process at the local office, issues OTP tokens that will be used to make payment requests. Our proposal involves interactions between a number of entities, described below: the relationship between entities is illustrated in Fig. 1.

Bitcoin Payment Server (BPS): manages hosted Bitcoin Wallets on behalf of beneficiaries, securely holds Bitcoin keys for each account holder and is connected to the Bitcoin/RSK peer-to-peer network. **Charity Local Office (LO):** located at the disconnected environment, registers phone numbers of users and manages distribution of OTP tokens. **Charity Head Quarters (HQ):** geographically distant from the aid environment, and has online access/secure servers: the HQ holds relevant Bitcoin private keys for all payers. **Charity Head Quarters Backup Server (HQB):** backup server which also holds relevant Bitcoin private keys for all payers. **One Time Password (OTP) Token:** cheap Hash-based One Time Password (HOTP) security token used with every SMS transaction. **SMS-Gateway:** server that sends and receives SMS transmissions[1] to and from the telecommunication network, and is connected to the BPS. Additionally, we make the following assumptions: **Charity Head Quarters (HQ):** The charity operates on an international level while providing humanitarian aid for offline beneficiaries. It is a reputable and trusted entity, with secure premises and online access/backup servers which may be geographically distant from the aid environment. **Donors:** Potential donors must have online access to use the donor platform. **Donor Platform:** Hosted on a secure web server adhering to industrial standard security controls to prevent attacks (such as: Denial of Service, website defacing, content manipulation, etc.). **Bitcoin Payment Server (BPS):** Secure server managed under industrial standard security controls to prevent attacks. All security keys are kept encrypted and stored securely to minimise the risk of data breaches. **Phones:** All users of the system possess simple mobile phones ('feature phones') that are protected by security code/access PINs, and the local existing GSM network can be used for SMS messages. **Secure Registration:** At the LO, all users of the system must register their mobile numbers and be issued with cheap Hash-based One Time Password (HOTP) security tokens. Mobile numbers are assigned an OTP identifier and Bitcoin wallet. All registration details are sent to the BPS (encrypted using the LO's private key), in batches if the LO's internet connection is intermittent. **Security Token:** This is a cheap hardware security token that generates HMAC-Based (HOTP) passcodes when the user requests ("event-driven"). These codes remain valid

[1] All SMS messages used in the proposal are within the standard 160 character length.

until used by the authenticating application. Typical OTP lengths are 8 digits or 6 alphanumeric characters, and are generated by standardised algorithms e.g. RFC4226 [19]. The BPS can generate a user's transaction OTP using the same algorithm. **Trust:** SMS-Gateway and BPS are trusted & secure. Mobile phones are not. **Bitcoin wallet addresses:** All the Bitcoin wallet addresses and Bitcoin transactions use a 2-of-3 multi-signature process. The key holding entities are the BPS, charity HQ and HQB. So when the BPS receives a payment request, it cannot broadcast a valid Bitcoin transaction to the Bitcoin peer-to-peer network without it being authorised by one of the other keyholders.

4.4 Processing a Bitcoin Payment Request

Payments can be made from charity worker to beneficiary, beneficiary to merchant, or merchant to merchant[2], and the message flow is shown in Fig. 2. The notation used is shown in Table 2, security credentials for each entity are shown in Table 3 and the content of each SMS messages used is shown in Table 4. For simplicity of exposition, the following description shows the Head Office (HQ) providing the second Bitcoin key.

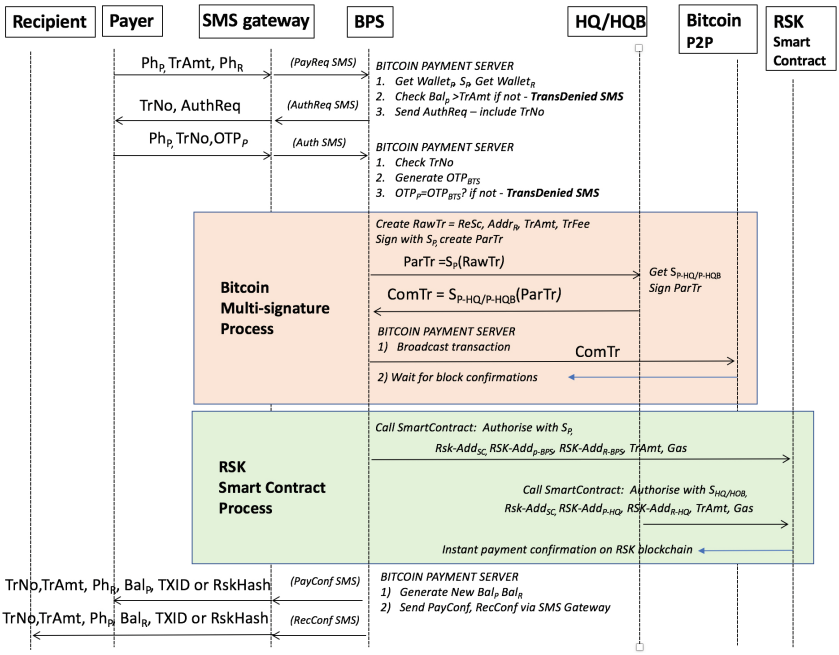

Fig. 2. SMS payment message flow

[2] Merchants could use an existing Bitcoin address, registered and associated with a short Merchant ID by the BPS, used instead of Ph_P/Ph_R in transactions.

Table 2. Notation used in proposed SMS payment scheme

Notation	Description
$Addr_X$	Bitcoin multi-signature address for entity X
BPS/Ph_X	Bitcoin payment server (entity)/Phone number of entity X
BAL_X	Bitcoin balance in Account AC_X for entity X
E_K(Z)/X→Y	Encryption of data Z with key K/Message sent from X to Y
HQ/HQB	Head quarters (entity)/Head quarters backup location (entity)
LO/P/R	Local office (entity)/Payer (entity)/Recipient (entity)
OTP_X	One time password generated by entity X
PK_X/SK_X	Public/Secret key pair of entity X
$S_X/TrHash$	Bitcoin private key of entity X (signing key)/Transaction Hash
$TrAmt/TrNo$	Transaction amount/Transaction number
TXID	Unique transaction ID of a transaction recorded in the blockchain. Also referred to as the Transaction Hash (TrHash)
(Z)$Sign_K$	Signature on data Z with signature key K
TrFee	Transaction fee paid to the Bitcoin miner
RawTr	Raw transaction created for signing
ParTr	Partial signed transaction created after signing *RawTr*
ComTr	Complete signed transaction created after signing *ParTr*
ReSc	Full redeem script used for the Bitcoin multi-signature address
RSKHash	Rootstock transaction hash
RSK-Add$_{SC}$	RSK smart contract address, unique for the contract and never changes
RSK-Add$_{X-Y}$	RSK public key (RSK address) of entity X kept securely with entity Y
$S_{RSK-X-Y}$	RSK private key of entity X kept securely with entity Y
Gas	Transaction fee paid to execute instructions on the smart contract

Stage 1: Payment Request: to make a payment, the Payer (P) types an SMS with payment instructions (*PayReq SMS*), and sends it to a local phone number provided by the charity, to be forwarded to the charity's BPS via the SMS-Gateway. The BPS retrieves Bitcoin wallets for both Payer and Recipient, checks *TrAmt* is not greater than BAL_P, pseudo-randomly generates a three-digit number, unique per transaction *TrNo*, and then sends *AuthReq SMS* asking for Payer's OTP. The Payer presses a button on the OTP token, then sends *Auth SMS* containing the resulting OTP to authorise the transaction. The BPS checks the TrNo, generates OTP_{BPS} and compares to the received OTP_P. If any checks fail, *TransDenied SMS* is sent to the Payer. If all checks are passed then the BPS proceeds to making a Bitcoin payment, using one of the two proposed options.

Stage 2: Bitcoin Transaction Processing

Option 1: Multi-signature Process: The BPS first generates a Raw Transaction (*RawTr*) which includes the Full Redeem Script (*ReSc*), the new multi-signature address associated for the receiver where the payment is going to, *TrAmt* and *TrFee*. The *RawTr* then needs to be signed by minimum 2 participants in turn to generate a valid Bitcoin transaction. The BPS first signs the *RawTr* using the corresponding Payer private key S_P and forwards the Partial Signed Transaction (*ParTr*) to the HQ for signing.
$BPS \rightarrow HQ$: $ParTr = (ReSc, Addr_R, TrAmt, TrFee)Sign_{S_P}$

To authorise the payment request, HQ first verifies the *ParTr* to check the payment amount and number of signatures needed. Once satisfied, HQ signs this using its private payer Bitcoin key S_{P-HQ} to generate the Complete Signed Transaction *ComTr* and sends this back to the BPS.
$HQ \rightarrow BPS$: $ComTr = (ParTr)Sign_{S_{P-HQ}}$

The BPS then broadcasts the *ComTr* to the Bitcoin peer-to-peer network. Once broadcast, a unique transaction-id (TXID) or the recipient's Bitcoin address can be used to trace the transaction on the blockchain. The Bitcoin miner who first publishes the valid block in the blockchain that also includes our Bitcoin transaction is paid the *TrFee* for the payment. This is the first confirmation for the transaction. The BPS then waits for the transaction to be confirmed in the agreed number of blocks before generating the SMSs.

Option 2: Smart Contract Process: The BPS calls the Smart Contract and authorises the *TrAmt* and the *Gas* is paid by using the $S_{RSK-P-BPS}$.
$BPS \rightarrow RSK$: $RSK\text{-}Add_{SC}, RSK\text{-}Add_P, RSK\text{-}Add_R, TrAmt, Gas$

Once the message gets broadcast in the RSK network, the HQ or the HQB calls the smart contract which act as the second set of instructions needed by the smart contract to execute the transaction. HQ/HQB uses the $S_{RSK-P-HQ/HQB}$ to authorise the paid amount *TrAmt* and the transaction fee *Gas*.

Table 3. Credentials used in proposed SMS payment scheme

Entity	Keys and other assets
Payer/Recipient	No keys, PIN for phone, HOTP token (no PIN) for making payments
BPS	S_{P-BPS}, $Addr_{P-BPS}$, $Addr_{R-BPS}$, PK_{LO}, Ph_X, OTP_X
HQ	S_{P-HQ}, S_{RSK-HQ}, ReSc
HQB	S_{P-HQB}, $S_{RSK-HQB}$, ReSc
LO	SK_{LO}, physical OTP tokens, phone numbers (payers/recipients), plus registration details/OTP allocation details
Donor	S_{Donor}/V_{Donor}
Donor platform	$Addr_{Project}$

Table 4. SMS payment messages

Message	Content
PayReq SMS	Ph_P, TrAmt, Ph_R
AuthReq SMS	TrNo, AuthReq
Auth SMS	Ph_P, TrNo, OTP_P
TransDenied SMS	Ph_P, TrNo, Ph_R, Denied
PayConf SMS	TrNo, TrAmt, Ph_R, BAL_P, TXID
RecConf SMS	TrNo, TrAmt, Ph_P, BAL_R, TXID
PayConfRSK SMS	TrNo, TrAmt, Ph_R, BAL_P, RskHash
RecConfRSK SMS	TrNo, TrAmt, Ph_P, BAL_R, RskHash

HQ/HQB→RSK: $RSK\text{-}Add_{SC}$,$RSK\text{-}Add_{P\_HQ}$,$RSK\text{-}Add_R$,*TrAmt*,*Gas*
When instructions are received from both BPS and HQ/HQB, the Smart Contract executes a transaction to transfer the value *TrAmt* to the recipient. The unique transaction details are recorded instantly on the RSK blockchain in the format of a hash (RskHash). The BPS does not need to wait for a transaction confirmation as there is instant confirmation when using the RSK platform.

Stage 3: Payment Finalisation: Once the payment is done, the BPS updates the payer/recipient balances and sends confirmation messages via the SMS-Gateway: *PayConf SMS* or *PayConfRSK SMS* to the Payer and *RecConf SMS* or *RecConfRSK SMS* to the Recipient. TXID/RskHash are included as unique IDs that can be used to trace the transaction on the Bitcoin/RSK blockchains.

5 Analysis

In this section, we discuss SMS security and analyse the proposal against security requirements shown in Sect. 4.2. A summary of targets that adversaries may attack along with suggested countermeasures is shown in Table 5.

SMS Security Issues: SMS messages are not encrypted by default and the SMS service is vulnerable to man-in-the-middle attacks and spoofing [4]. Attack methods include interception/redirection using false base stations in GSM networks, eavesdropping at the Short Message Service Centre (SMSC), and SS7 hacking [13]. Adversaries who might target the SMS system are *SL*, *CC* and *In*, aiming to create fraudulent transactions. Although these issues are not addressed directly in our proposal, measures have been included which provide some deterrent to would-be attackers. The use of the OTP means that replay attacks will fail, and the *AuthReq SMS* from the charity should alert users to potentially fraudulent transactions. Additional assurance comes from including both *TXID*/*RSKHash* and *TrNo* in confirmation SMSs: these can be used to cross check with the Bitcoin/RSK blockchain and in a verbal comparison between Payer and Recipient respectively, to provide an extra level of assurance that the

transaction is correct. These measures provide a higher level of security than other SMS Bitcoin schemes: e.g. in Coinapult SMS, the user sends an SMS containing a security code sent by the payment service in a previous SMS, which offers limited assurance that the transaction is genuine.

Table 5. Attack targets, adversaries and countermeasures

Target	SL	CC	Ha	In	Countermeasure
Donor platform	y	y	y		Hosted on a secure web server adhering to industrial standard security controls to defend against: DDoS, website defacing, content manipulation
HQ/HQB/BPS (DDoS)	y		y		HQ/HQB has secure premises and backup servers: BPS managed under industrial standard security controls and best practices to prevent attacks
HQ/HQB/BPS (privilege escalation)	y	y		y	Use of security controls such as: access control, routine web-application vulnerability assessment/patching and storing keys encrypted
SMS (MNO/GSM)	y	y			GSM/SMS security issues partially mitigated by OTP 2FA and TXID/RSKhash on confirmations
SMS spoof	y	y		y	OTP/TXID/RSKhash gives some assurance that payment is genuine
SMS replay	y	y		y	OTP prevents replay attacks
Blockchain/RSK (DDoS)	y		y		DDoS attacks not viable in distributed ledger, and integrity is innate in blockchain solutions

Security Requirements

Confidentiality-***Security of Bitcoin private keys:*** If a Bitcoin private key or Bitcoin wallet is lost or not accessible, then the Bitcoin value recorded to that Bitcoin address cannot be transferred. The 2-of-3 multi-signature process avoids this risk by allowing any two out of the three private key holders to recover the Bitcoins. ***Donor anonymity:*** Anonymous donations may introduce management issues for the charity, as this may need special reporting and investigation due to possible money laundering/fraud regulations. To comply with these, a charity policy may be needed requiring identification for donations over a certain amount. ***Server attacks (HQ/HQB/BPS):*** Adversaries *SL* and *CC* will aim at obtaining keys, transaction data and identity information: *Ha* may wish to find embarrassing data. Table 5 shows recommended countermeasures.

Integrity-*Blockchain* ensures the integrity by providing an immutable record of past transactions. ***RSK blockchain*** is mined by the same miners in the Bitcoin peer to peer network. Double-spending prevented by using proof-of-work using SHA256 hashing similar to Bitcoin and uses a checkpointing service provided by a federation of well-known and respected Bitcoin community members [2]. ***Server attacks(HQ/HQB/BPS/Donor Platform):*** the donation platform may be targeted by: *CC* to change published content by replacing the charity's Bitcoin addresses with addresses belonging to the criminals; *Ha* may aim to vandalise the content; *SL* may tamper with it to undermine the credibility of the charity. Transaction records at the BPS may be tampered with by *CC*, *SL* to make fraudulent transactions. ***SMS Replay Attacks:*** countered using the OTP in the *SMSAuth* message. Potential attackers here are *SL*, *CC* and *In*.

Authentication-*Authenticating the payment request SMS:* OTP security tokens are used for two-factor authentication. The charity's BPS authenticates the user by verifying the OTP included in the SMS, so if a phone is lost/stolen, an attacker cannot make a valid transaction. The OTP is valid until it is received and processed by the BPS, so network delays will not cause adverse effects. This should give some protection against spoofing attacks by adversaries *Sl*, *CC* and *In*. ***Mobile Phones:*** The handset's PIN protection will present a barrier to attackers who steal the phone. ***Transaction Number:*** In a point-of-sale transaction, the beneficiary and the merchant can compare the TrNo received on confirmation messages before a purchased product is handed out. ***Social engineering:*** Aimed at obtaining privileged access to data at HQ/BPS, so security awareness training will be needed. However, an insider at the BPS/HQ/HQB is not able to transmit a transaction alone because of the use of multi-signatures.

Availability-*Recovering lost Bitcoins:* If any one of the three Bitcoin private key holders loses their key the remaining two parties can recover the Bitcoins. ***DDoS Attacks:*** The donor platform, HQ/HQB and BPS are an attractive targets for *SL*, *Ha* DDoS attacks: see Table 5 for countermeasures. DDOS on the blockchain are not viable due to its innate security and distributed nature.

6 Conclusion and Future Work

This paper first identified the advantages of blockchains for charities, then discussed how blockchain solutions could be employed, even with their potential constraints. The first contribution is a new philanthropic model that leverages the Bitcoin blockchain. The payment system uses either a 2-of-3 multi-signature transaction process with the Bitcoin network, or a smart contract for advanced functionality utilising the RSK network. The second contribution is an SMS Bitcoin payment system that can be used in an offline environment via the existing GSM network. This proposal was then evaluated against security requirements. It must be noted that, the volatility of Bitcoin exchange value poses a financial risk for the charity. However, in an environment where the banking system/economy may have collapsed, using Bitcoin might be the only viable

option. Our solution is aimed at a closed eco-system where payments are made within a constrained geographical environment, thus minimising the effects of Bitcoin price volatility. As a long-term solution to this, the charity may replace the Bitcoin blockchain with a private blockchain solution to give more control over exchange prices. Other future work include a practical implementation of the proposed scheme to identify potential limitations and take timing measurements. Also, we would like to investigate how the philanthropic model could be applied in different situations e.g. when smartphones and Internet connectivity are available or in an ad-hoc network that replaces the existing GSM network.

References

1. Coinapult SMS. https://coinapult.com/sms/info
2. Rootstock platform. http://www.rsk.co/
3. BTC For SMS (2017). http://www.btcforsms.com/
4. FlexiSpy (2017). https://www.flexispy.com/en/features/spoof-sms.htm
5. GiveTrack: Donation tracking, March 2017. https://bitgivefoundation.org/bitcoin-charity-2-0-initiative/
6. M-PESA, December 2016. https://www.safaricom.co.ke/personal/m-pesa
7. Birkwood, S.: Is Bitcoin the ideal charity currency or a cause for concern?, January 2015. http://www.thirdsector.co.uk/analysis-bitcoin-ideal-charity-currency-cause-concern/fundraising/article/1326549
8. Bitcoin.org: Bitcoin wiki (2014). https://en.bitcoin.it/wiki/Main_Page
9. Blockchain.info: Average transaction confirmation time. https://blockchain.info/charts/median-confirmation-time. Accessed Nov 2016
10. Charities Aid Foundation: Giving Unchained: Philanthropy and the Blockchain
11. Davies, R.: Public Good by Private Means: How philanthropy shapes Britain. Alliance Publishing Trust (2016)
12. Gautham: BTC.com Wallet App, September 2016. http://www.newsbtc.com/2016/09/19/btc-com-wallet-app-sms-bitcoin/
13. Gibbs, S.: SS7 hack explained: what can you do about it?, April 2016. https://www.theguardian.com/technology/2016/apr/19/ss7-hack-explained-mobile-phone-vulnerability-snooping-texts-calls
14. Gilman, D.: Cyber-Warfare and Humanitarian Space, October 2014. http://commstech-hub.eisf.eu/uploads/4/0/2/4/40242315/daniel_gilman_cyberwarfare_and_humanitarian_space_eisf_october_2014.pdf
15. ImpACT Coalition: Through a glass DARKLY: The case for accelerating the drive for accountability, clarity and transparency in the charity sector. Technical report (2013)
16. International Telecommunications Union (ITU): ICT Facts And Figures 2016. http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf. Accessed Oct 2016
17. Mankiw, N.: Principles of Microeconomics, vol. 10. Cengage Learning (2006)
18. Nakamoto, S.: Bitcoin: a peer-to-peer e-cash system. Bitcoin.org (2008). http://www.bitcoin.org/bitcoin.pdf
19. RFC 4226: HOTP: An HMAC-Based One-Time Password Algorithm, December 2005. http://www.ietf.org/rfc/rfc4226.txt
20. The World Bank: Remittance prices worldwide (2016). https://remittanceprices.worldbank.org/en. Accessed Dec 2016

21. Walport, M.: Distributed ledger technology: beyond blockchain. UK Government Office for Science, Technical report, p. 19 (2016)
22. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)
23. Wright, A., De Filippi, P.: Decentralized blockchain technology and the rise of lex cryptographia. SSRN 2580664 (2015)
24. Young, J.: Former Kipochi CTO Explains Controversial M-Pesa Deal, January 2016. http://www.newsbtc.com/2016/01/11/former-kipochi-ceo-explains-controversial-m-pesa-bitcoin-deal/

Security of Data

Long White Cloud (LWC): A Practical and Privacy-Preserving Outsourced Database

Shujie Cui(✉), Ming Zhang, Muhammad Rizwan Asghar, and Giovanni Russello

The University of Auckland, Auckland, New Zealand
scui379@aucklanduni.ac.nz,
{ming.zhang,r.asghar,g.russello}@auckland.ac.nz

Abstract. To fully benefit from a cloud storage approach, privacy in outsourced databases needs to be preserved in order to protect information about individuals and organisations from malicious cloud providers. As shown in recent studies [1,2], encryption alone is insufficient to prevent a malicious cloud provider from analysing data access patterns and mounting statistical inference attacks on encrypted databases. In order to thwart such attacks, actions performed on outsourced databases need to be oblivious to cloud service providers. Approaches, such as Fully Homomorphic Encryption (FHE), Oblivious RAM (ORAM), or Secure Multi-Party Computation (SMC) have been proposed but they are still not practical. This paper investigates and proposes a practical privacy-preserving scheme, named *Long White Cloud* (*LWC*), for outsourced databases with a focus on providing security against statistical inferences. Performance is a key issue in the search and retrieval of encrypted databases. *LWC* supports logarithmic-time insert, search and delete queries executed by outsourced databases with minimised information leakage to curious cloud service providers. As a proof-of-concept, we have implemented *LWC* and compared it with a plaintext MySQL database: even with a database size of 10M records, our approach shows only a 10-time slowdown factor.

1 Introduction

Cloud computing provides organisations with virtually unlimited storage and computational power at attractive prices. One of the main challenges of outsourcing large databases to the cloud is to secure data from unauthorised access. A naive approach is to use standard encryption techniques for protecting the data. However, using this approach, no operations could be performed on encrypted databases.

Ideally, one would like to perform operations such as search, insert and update directly on encrypted databases without letting cloud providers learn information about both the query and the data stored in the database. For such scenarios, the Searchable Symmetric Encryption (SSE) scheme has been proposed,

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 41–55, 2018.
https://doi.org/10.1007/978-3-319-93524-9_3

where symmetric keys are used for encrypting the data and queries. SSE was first introduced by Song *et al.* in [3] in 2000 and several more research efforts have been presented since then. Unfortunately, most of the existing SSE schemes suffer from one or more of the issues listed below.

- **Information Leakage.** During data search, the correlation between the queries and the matched data is leaked to the cloud server. This correlation can be exploited by a determined attacker to break the encryption scheme as shown in recent studies [1,4]. Approaches like Oblivious RAM (ORAM) [5–7] or Private Information Retrieval (PIR) [8,9] could be used to minimise information leakage. However, these schemes are very costly and/or can only be applied in static settings, meaning they do not scale well when dealing with dynamic data updates and delete operations.
- **Lack of Support for a Full-Fledged Multi-User Access.** The vast majority of existing approaches support a very basic key distribution scheme, where all users share the same key. We refer to these as Single User (SU) schemes. Another common approach is to have a read-only key shared among all the users and one special key for inserting/updating data. We refer to these as Semi Fledged Multiple User (SFMU) schemes. In both cases, if a user misplaces a key or needs to have her access revoked, then a new key needs to be generated and the data requires re-encryption under the new key. Instead, in a Full-Fledged Multi-User (FFMU) scheme, any authorised user is able to read and write data from and to the database, respectively [10]. An FFMU scheme better fulfils needs of modern organisations, where users need to access and update data and are able to join and leave the organisation at any time without affecting rest of the users.

Our contribution is to propose a very efficient sub-linear Dynamic SSE (DSSE) scheme, named *Long White Cloud* (*LWC*), able to support a high throughput of queries while minimising information leakage. In terms of efficiency, our scheme is similar to Stefanov *et al.* [11] and Ishai *et al.* [12]. However, unlike their approach, our scheme is designed for large organisations, where users might join and leave at any time. Therefore, we want to support an FFMU scheme to simplify the user registration and revocation. At the same time, we want that each user is able to insert a new record, update existing records, and retrieve any data from the database.

The main idea in *LWC* is to use a hybrid private/public cloud approach. In such an approach, organisations maintain a private part of the infrastructure on their local premises while outsourcing the rest to a public cloud provider, such as Amazon or Google. In *LWC*, the private cloud is used to maintain data structures for speeding up the query processing and to perform operations in order to minimise information leakage. The public cloud instead is mainly used as an encrypted data store.

The rest of this paper is structured as follows: Sect. 2 gives an overview of the related work. Section 3 presents the system model, threat model, an abstract architecture of *LWC*. Section 4 describes key management and explains how we

represent the data. Section 5 explains the database queries supported by the *LWC* scheme. Section 6 analyses security aspects of *LWC*. Section 7 evaluates the performance of *LWC* and compares it with a cleartext MySQL database. Section 8 concludes by highlighting the main contributions and results of the *LWC* scheme.

2 Related Work

Since the seminal work by Song *et al.* [3], many searchable schemes have been proposed and the research in this area has been extended in several directions. We focus mainly on three aspects of the encrypted search: key management, search efficiency and information leakage. A thorough and up-to-date survey and comparison of the current literature can be found in our paper [13]. We stress here that, with this work, we aim at proposing an FFMU scheme that minimises information leakage while supporting sub-linear search. Most importantly, our scheme is very efficient thus practical. Before discussing the rest, we first set the context and informally define the properties related to information leakage.

- **Search Pattern Privacy (SPP)** refers to the property where the cloud server is not able to distinguish if two (or more) queries are the same or not.
- **Access Pattern Privacy (APP)** means the cloud server is unable to learn if two (or more) real result sets overlap or not.
- **Size Pattern Privacy (SzPP)** is achieved if the cloud server is unable to learn the size of returned (real) records.
- **Operation Pattern Privacy (OPP)** ensures the cloud server cannot tell if the executed query is a select, update, delete, or insert.

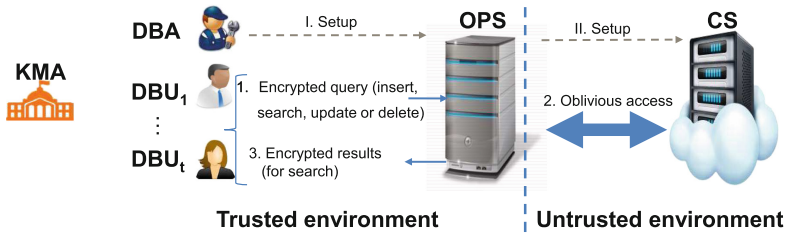

Fig. 1. System entities and their interactions in *LWC*: A DBA is responsible for running the setup (Step I then Step II). A DBU encrypts the query, which could be insert, select, update or delete. A DBU sends the encrypted query to the OPS (Step 1). The OPS communicates with the CS in an oblivious manner (Step 2). The OPS returns encrypted results (in case of select) back to the DBU (Step 3). The DBU can decrypt and get access to the requested data.

3 Overview of LWC

3.1 System Model

The system involves the following entities, also shown in Fig. 1:

- A **DataBase User (DBU)** represents an authorised user who is allowed to access, retrieve and update information in the outsourced database.
- A **DataBase Administrator (DBA)** is responsible for the setup and management of the outsourced database. It is also responsible for the management of DBUs. A DBA is considered to act on behalf of the organisation that outsources the database.
- The **Operations Proxy Server (OPS)** is under the direct control of the DBA. With the help of the OPS, a DBA can grant, revoke and regulate access to the outsourced database. For database operations, the DBU interacts with the OPS to run all kinds of queries including insert, select, update and delete.
- The **Cloud Server (CS)** is part of the public infrastructure that provides cloud storage services. The CS is usually hosted and operated by a third party service with full access control over its cloud storage. It is expected to provide only two main operations: read and write.
- The **Key Management Authority (KMA)** is responsible for issuing encryption keys. Once the scheme is initialised, the KMA supplies encryption keys to new DBUs and the OPS. Typically, the KMA is under the complete control of the DBA and is online only when the DBA requests encryption keys for new DBUs. In other words, it can go offline after a DBU gets registered.

3.2 Threat Model

The KMA is assumed to be a fully trusted entity. The DBUs are responsible for securely keeping their keys (and decrypted information). We assume that the OPS is deployed on the private cloud; whereas, the CS is hosted on a public cloud infrastructure. The OPS is assumed to be trusted but may be potentially vulnerable to external attacks since it communicates with the external world. As the vast majority of the existing SSE schemes consider, including all the works listed in [13], the CS is assumed to be honest-but-curious. That is, the CS follows the protocol correctly, but may be interested to read the information or to try and discover statistical inferences on its database access and operations. DBUs may collude among themselves but they do not learn anything more than what they would learn individually. A DBU colluding with the CS reveals no useful information. An active or revoked DBU is unable to decrypt communication between other DBUs and the OPS.

3.3 System Interactions

LWC aims at minimising potential information leakage while supporting sub-linear search in a multi-user setting. The interactions between the entities are

shown in Fig. 1. The following sequence of steps takes place to initialise and run the system. The DBA sets up the OPS (Step I) and then prepares the database on the CS (Step II). The DBA brings the KMA online to distribute encryption keys between participating DBUs and the OPS. Using her key, a DBU encrypts and issues queries including insert, search, update and delete.

As shown in Fig. 1, a query is processed in two main phases. In the first phase, the DBU sends the encrypted query to the OPS (Step 1) and then the OPS consults the local lookup table for finding locations of the records stored on the CS. In the second phase, the OPS uses these locations to fetch the data from the CS. To avoid any potential information leakage, the OPS queries the requested information in an oblivious manner (Step 2). The oblivious access hides from the CS the actual query issued by the DBU or the result set returned to the DBU. If the query is select, the OPS sends the encrypted result back to the DBU (Step 3). Finally, the DBU decrypts the encrypted results using her private key.

4 Key Management and Data Representation

4.1 Key Management

LWC supports the FFMU access. That is, if the DBU key is stolen or compromised, the system can still work without requiring re-encryption of the data with new keys and re-distribution of the new keys to the authorised users. In *LWC*, for each DBU, the KMA generates two keys (K_{DBU}, K_U). The KMA forwards both (K_{DBU}, K_U) keys to the DBU but only K_{DBU} to the OPS. The key K_U is a shared key across all DBUs and the key K_{DBU} is a DBU specific key that is shared with the OPS. The OPS stores all the DBU specific keys in a key store. Using both K_{DBU} and K_U, the DBU can issue queries and decrypt the search result. For revoking a DBU, the DBA instructs the OPS to remove the corresponding K_{DBU} key from its key store. Without loss of generality, instead of using K_{DBU}, a secure channel (say using SSL) can be established to protect communication between the DBU and the OPS.

4.2 Data Structure for the CS

The data stored on the CS is organised with three different data units, as shown in Fig. 2. As a whole, the database is a set of n blocks $\{b_0, \ldots, b_{n-1}\}$. The OPS uploads or downloads data to or from the CS block by block. Each block consists of m slots $\{s_0, \ldots, s_{m-1}\}$. Each slot stores a single record. A tuple of block number and slot number (b_i, s_j) uniquely identifies a stored record in the database. Specifically, we define a slot as *empty* if it holds no record. An empty slot may hold *null* or a random bit string. On the contrary, a slot is *full* if it is occupied by a DBU inserted record. Furthermore, each slot contains l cells. In other words, each encrypted record in a slot is divided into l data cells. Overall, each block is a set of $m * l$ data cells $\{c_0, \ldots, c_{m*l-1}\}$. Data re-encryption in

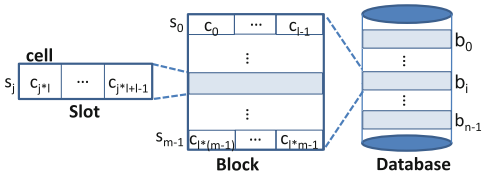

Fig. 2. Database on the CS: A database consists of a set of n blocks $\{b_0, \ldots, b_{n-1}\}$. Each block contains m slots $\{s_0, \ldots, s_{m-1}\}$, where a slot stores a single record. Each slot s_j has l cells.

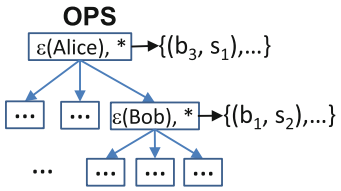

Fig. 3. A sample of B$^+$tree with 3 branches and 3 layers. The entry on each node is an encrypted keyword, and a pointer points to a list of (b_i, s_j) indicating the record store location on the CS.

LWC is implemented by permuting data cells (see Sect. 5.3). Assume each data cell is w-bit long, there are 2^w possible cell values, and each encrypted record is fixed to $w * l$ bits.

4.3 Data Structure for the OPS

In *LWC*, the search operation is performed on the OPS. Considering the OPS is trusted, any data structures that support sub-linear search, such as inverted index technique proposed in [14] and the red-black tree introduced in [15], could be used. In this work, we use a series of B^+trees [16] for managing indexing information on the OPS, which aims at supporting sub-linear search. Each field in a record is stored on a different B$^+$tree (see Fig. 3) and hence it provides a simple and efficient method to query for keywords across different fields. Each entry in the tree is a tuple of an encrypted keyword and a list of (b_i, s_j) indicating the locations of the records containing this keyword on the CS. We can notice that the unique keyword number in each field determines the size of the corresponding tree, which is much less than the number of records on the CS. To further reduce the storage overhead, the OPS could hash the encrypted keyword first before inserting it into the tree.

The OPS also stores a list of flags to mark if each slot is full or empty. Using this information, the OPS can pick an empty slot and store the inserted record.

5 Query Execution

In *LWC*, a DBU can issue queries for inserting, updating, searching, and deleting records. A query may include simple keywords or conjunctions/disjunctions of conditions, like "select * from Staff WHERE name = Alice AND age = 25". As mentioned in Sect. 3.3, the queries are mainly processed in two phases by *LWC*.

The first phase aims to get the locations of records (for a select query) or an empty slot (for an insert query) on the CS. Technically, it involves 4 steps on the DBU and the OPS. The detail of each step is shown in Fig. 4.

5.1 Encryption on the DBU

The first two steps, which are part the first phase, are completed by the DBU. The system components, running on the DBU end, include the encryption and decryption utilities. Each query is processed in two rounds of encryptions on the DBU using two different functions $\varepsilon : Q \rightarrow \varepsilon_{K_U}(Q)$ and $\epsilon : EQ \rightarrow \epsilon_{K_{DBU}}(EQ)$. ε is a deterministic symmetric encryption to make the encrypted data searchable and retrievable. In the first round of encryption, the query Q is encrypted using K_U, a key shared among all DBUs. On one hand, it ensures the data could be accessed by all the DBUs. On the other hand, it also means the encrypted query EQ and search pattern are not protected from other DBUs. To address this issue, we introduce the second round of encryption, where EQ gets re-encrypted under K_{DBU}. The second round of encryption ϵ is semantically secure. Because K_{DBU} is unknown to other DBUs, they are unable to learn EQ and search pattern. Without loss of generality, we could issue a password to each DBU for authentication and then use SSL/TLS to establish a secure channel between the DBU and the OPS to deliver EQ. ε^{-1} and ϵ^{-1} are the their corresponding decryption functions. Note that the first round of encryption is only performed over the keywords in Q. That is, the logical conditions and operators in EQ are in cleartext (for the OPS). For instance, "select * from Staff WHERE $\varepsilon_{K_U}(name) = \varepsilon_{K_U}(Alice)$ AND $\varepsilon_{K_U}(age) = \varepsilon_{K_U}(25)$". However, in the second round of encryption, EQ is encrypted as a whole, which means all the information in EQ is protected.

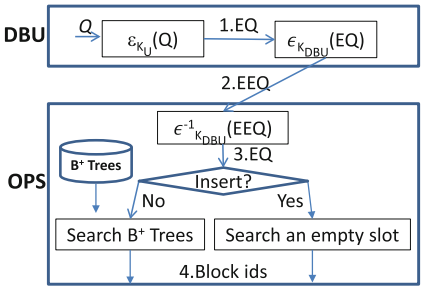

Fig. 4. Query execution: A DBU encrypts a query Q twice with K_U and then K_{DBU}. The encrypted query EEQ is sent to the OPS. The OPS decrypts it with K_{DBU} to get EQ, and then look ups an empty slot for inserting a record or searches B^+ trees according to the type of the query.

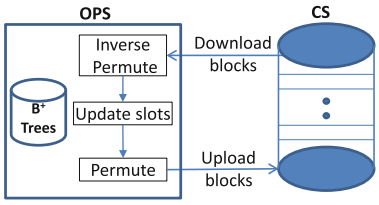

Fig. 5. Oblivious access between the OPS and the CS: The OPS first downloads a set of blocks, which contains the required slots, and then performs three steps (inverse permutation, updating empty slots, and permutation) after which the blocks are uploaded back to the CS.

5.2 Index Search on the OPS

As shown in Fig. 4, the last two steps of a query execution are performed by the OPS. The OPS is responsible for parsing the query, searching for the locations of the corresponding records on the CS database and fetching those records.

In case of a select query, the OPS returns those records to the DBU. Parsing *EEQ* on the OPS enables *LWC* to execute any kind of query, which could involve conjunctions, disjunctions and other logical operators for a multi-keyword search, update and delete queries. As shown in Fig. 4, the components on the OPS includes the ϵ^{-1} for decrypting *EEQ*, B^+ trees for managing indexing information to enable sub-linear search.

In the third step, the OPS removes the outer layer encryption of *EEQ* to get the executable query *EQ* by running the inverse stream cipher function $\epsilon^{-1} : EEQ \rightarrow EQ$.

For an insert query, the next step is to check the flags list and find an empty location (b_i, s_j) to hold the record. Next, the selected location (b_i, s_j) will be added to the corresponding entry in the B^+ tree for each field (see Fig. 3). If the related entry is already in the tree, we just add the (b_i, s_j) to its list. Otherwise, a new entry will be created. Note that the keywords in trees are encrypted, which ensures data confidentiality even if the OPS gets compromised.

In case of search, delete and update queries, the fourth step is to search the B^+ trees and find out which records on the CS match the query. The B^+tree search incurs $O(\log_b N)$, where N denotes the total number of nodes in the tree and b represents the branching factor of the tree. For efficiency reasons, we suggest having a separate tree for each field in the table. Recall the sample query, two predicates "$\varepsilon_{K_U}(name) = \varepsilon K_U(Alice)$" and "$\varepsilon(age)_{K_U} = \varepsilon_{K_U}(25)$" are searched over two trees separately. Once the location tuple list is searched out for each separate keyword, the final search result is the combination of them according to the logical operator between keywords, which is in cleartext on the OPS.

With these locations, the second phase is to execute (*i.e.,* insert, select, update or delete) on the CS database obliviously, which is explained in Sect. 5.3.

5.3 Oblivious Access

In *LWC*, we aim to protect operation pattern, size pattern, access pattern and search pattern from the CS. The OPS gets the location tuples for the query. If the OPS sends the tuples to the CS directly, the CS could learn these patterns easily. The data should be accessed in an oblivious manner. The detail of the oblivious access between the OPS and the CS is shown in Fig. 5. The OPS performs the following three steps: inverse permutation, update slots and permute. The detail of each step is given below.

Recall that all the data stored on the CS is encrypted using K_U, which is a key shared among all the DBUs. This implies that K_U is still known to DBUs that have been revoked. A revoked DBU can collude with the CS to decrypt the data stored on the CS. To address the problem, instead of re-encrypting the data, we propose a more efficient approach. The idea is to shuffle the data cells between different records in each block with a pseudo-random permutation π. That is, all the data cells in each block are permuted in an invertible way, which is only known to the OPS. Consequently, the DBU is unable to decrypt the data without the assistance of the OPS. In each permuted block, the data in each slot

is no longer a complete record. The permutation seed for each block is kept by the OPS. Note that to make the permutation invertible for the CS, we should set $m * l \gg 2^w$, where $m * l$ is the number of data cells in a block and w is bit length of a data cell.

Before any operation, the OPS has to download the matched records from the CS. If the matched slots are distributed in k blocks, the CS will download γ blocks, where $k < \gamma$. k of them are the matched blocks, the rest $\gamma - k$ blocks are picked randomly. Note that all blocks from the database must be picked through a single request from the OPS. After receiving γ blocks, the OPS first performs π^{-1} over the $m * l$ data cells for each block to recover the data order. In the second step, actions (described below) are taken based on the query type:

- If EQ is a select query, all the matched slots will be sent to the DBU.
- If EQ is a delete query, the OPS just changes flags of these matched slots to mark them empty and does nothing over the records in each block.
- If EQ is an update query, the matched records will be updated with the new values.
- If EQ is an insert query, the new records will be inserted into the selected slot.

After that, the OPS updates or fills a number of empty slots with random bit strings for each block. In the third step, all the data cells in each block are permuted again with a new seed. Finally, the γ updated blocks are written back to the CS. The security of *LWC* is analysed in Sect. 6.

5.4 Data Decryption

In case of a select query, there is one more phase between the OPS and the DBU, *i.e.,* the decryption of the result. Once the records in each block are retrieved, the OPS will extract the records satisfying the query. Before sending them to the DBU, the OPS first encrypts it with ϵ. Since the encryption key K_{DBU} is unique for each DBU, the search result could only be decrypted by the DBU who issued the query. Considering ϵ is semantically secure, the access pattern can not be inferred from the interaction between the DBU and the OPS. To hide the size pattern, the OPS could add a set of dummy data into the result. On the DBU, two rounds of decryptions will be performed to get the cleartext records.

6 Security Analysis

In this section, we analyse how the oblivious access described in Sect. 5.3 could protect size pattern, access pattern and operation pattern from the CS.

Recall that we say SzPP is achieved if the CS cannot learn how many records are matched for each query. *LWC* achieves SzPP with two techniques. First, the OPS downloads data from the CS block by block, rather than slot by slot. It is unknown to the CS how many slots are matched with the query in each block. Second, γ blocks are downloaded by the OPS for each access. Here γ is a random

number determined by load on the OPS, but independent from the real matched blocks. Note that, for the same query, if load on the OPS is different, the value of γ would be different. In other words, from the number of accessed blocks, the CS is unable to infer anything about search pattern.

Similarly, access pattern is also protected at two different levels. Since the data is downloaded block by block, it is unknown to the CS which slots in each block are matched with the query. If only the matched blocks are downloaded by the OPS, given the number of matched records is unknown to the CS, in the view of the CS, each record in a block could be a matched one or not with 50% probability. Furthermore, matched blocks are also protected by random blocks. When γ blocks are downloaded for a query, there are 2^γ possible block access patterns. The probability that the CS could guess the real block access pattern successfully is $\frac{1}{2^\gamma}$. Moreover, from the relationship between downloaded blocks, the CS is unable to infer search pattern.

Theorem 1. *If the $\gamma - k$ blocks are picked randomly, LWC partially achieves SPP.*

Proof. (sketch) Since EQ is never sent to the CS, the CS could only infer search pattern from the relationship between size patterns and the relationship between accessed blocks. If access patterns and size patterns of two queries are the same, there is a high probability that the queries are the same, and vice versa. As we mentioned above, search pattern cannot be inferred from size pattern, since they are always variable no matter whether the queries are the same or not. In *LWC*, due to the random blocks, access patterns are also always variable for all queries. More specifically, for the same queries, $\gamma - k$ random blocks could make their block access patterns different. However, for different queries, $\gamma - k$ random blocks may cause an overlap between their block access patterns. In the view of the CS, each block could be the matched one or random with the same probability. It has no advantage to infer search pattern from the overlap between block access patterns. There are 2^m possible slot access patterns in each block. The probability that queries have the same slot access patterns in one block is $\frac{1}{2^{2m}}$. However, the CS could learn the two queries are different when the two accessed block sets are totally different. To avoid such leakage, the OPS could download the whole database each time, but it is costly.

Theorem 2. *If the encrypted records are indistinguishable from random bit strings and if π is a pseudo-random permutation, LWC achieves OPP.*

Proof. (sketch) To protect operation pattern, the OPS always performs the same operations. Specifically, no matter what type the query is, the OPS updates or fills a number of the empty slots with random bit strings. If the query is insert, the selected empty slot will be filled with the new record. If the query is update, the matched slots will be updated with new values. But, in the view of the CS, there are always some slots that are updated or filled in each block for each access. Since the encrypted record is indistinguishable from a random bit string, the CS is unable to learn if the slot is filled with a record or random

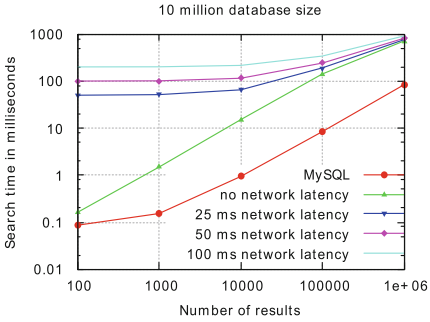

Fig. 6. End-to-end search time in *LWC*.

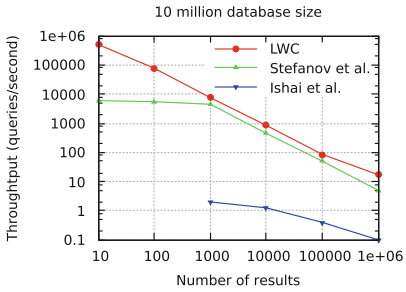

Fig. 7. Query throughput comparisons for database schemes.

bit string, making insert and update queries indistinguishable. If the query is delete, the records are removed by changing the flags, but the values in each block are unchanged like the select query. Moreover, the $m * l$ slots in each block are permuted again with a new seed before uploading them to the CS. Consequently, which slots and how many slots are updated or filled, and if slots are updated partially or completely are also protected from the CS.

The problem is, for select and delete queries, if all the slots in accessed blocks are filled with DBU inserted records, the OPS would not be able to insert or update random bit strings. Recall that there are 2^w different cell values in the system. Even if all the data cells will be permuted again, the number of each unique cell value in each block remains unchanged for select and delete operations. With the frequency information of each data cell value, the CS could distinguish update from select and delete operations. To solve this problem, we set a block is *full* when θ out of m slots are occupied by records and do not insert any records into it anymore, where $\theta < m$. That is, each block always has at least $m - \theta$ empty slots to fill random bit strings. Therefore, whatever the query type is, the frequency of data cell values is changed all the time. In practice, we can set $\theta = \lceil 2^w/l \rceil$. In this case, the frequency of each data cell value could be changed.

Although the OPS is trusted, considering the data stored in the B^+ trees are encrypted, the queries received from the DBUs are encrypted as well. Thus, compromising the OPS will not put any data at risk. However, SPP, APP and OPP might not be guaranteed, since the data and queries are encrypted using a deterministic algorithm. We aim at further investigating these issues in our future work.

7 Experimental Evaluation

In this section, we analyse performance of *LWC*. For all the experiments, we used a PC powered by Intel i5-4670 3.40 GHz processor and 8 GB of RAM using Linux Ubuntu 15.04. Note that we have chosen a very basic PC setup to show

that *LWC* can achieve high performance even when deployed on cheap hardware. The prototype is programmed in C and is compiled using GCC version 4.9.2. No parallel operations or hyper-threading were implemented. All data structures are stored in RAM. For the experiment, we set the OPS to pick up to $2 \cdot k$ blocks from the CS for each query, where k is the number of blocks needed to execute the query. We fixed the maximum size of a record inserted at 128 bytes the number of cells in a slot was set at 256 cells.

The experiments presented in the following were set up as follows. All the entities (the DBU, the OPS and the CS) were executed on the same machine. However, where required, we used a simulated round-trip network latency for the links between each entity. In the following, the queries with a single predicate were executed on a database with 10 million records and all the results were averaged over 10 trials.

First, we measured the end-to-end time for a DBU to perform a search operation varying the number of the returned records between 100 to 1 million. The results are shown in Fig. 6 in milliseconds (ms). These measurements include also the time on the DBU to encrypt the query and decrypt all the returned records. As a baseline, we executed the same experiments using a plaintext MySQL database where the client and the database were deployed on the same machine without any network latency. With no network latency, *LWC* end-to-end search time on an average was between 2 (when 100 records were returned) and 10 times slower than the plaintext MySQL. Given that in *LWC* the DBU and the CS do not interact directly but through the OPS, we have performed the same experiments but introducing a simulated round trip network latency of 25 ms, 50 ms and 100 ms. As we can see in Fig. 6, the effect of network latency on search time rapidly reduces when the result size increases. In any case, with a result size of 1 million records, the end-to-end search time is under 1 s even when 100 ms network latency is introduced.

To investigate the penalty introduced by the OPS, we have compared the query throughput of *LWC* with schemes proposed by Stefanov *et al.* [11] and Ishai *et al.* [12]. To the best of our knowledge, both schemes are currently the best in term of performance when compared with other SSE schemes. The results of this comparison are presented in Fig. 7. Note that, at the time of the writing of this paper, we could not get access to their implementations. The graphs for Stefanov's and Ishai's schemes in Fig. 7 are plotted using the data presented in their respective papers. The comparison shown in Fig. 7 is an approximation for two main reasons: (1) in Ishai's paper, there are no throughput values for result sizes smaller than 1K; (2) while Ishai's scheme has been tested on a hardware configuration very similar to ours, results by Stefanov *et al.* were collected on a top of the line hardware configuration with a maximum degree of parallelism at 32. Therefore, the results presented in Fig. 7 for Stefanov's scheme have been normalised as executed on a single core.

The comparison in Fig. 7 shows that *LWC* on average has a throughput about twice that of Stefanov's when no parallelism is used. For instance at 10,000 results, the throughput of *LWC* is 852 queries per second, where Stefanov's

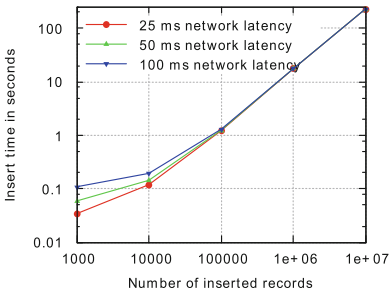

Fig. 8. Time taken for executing an insert query in *LWC*.

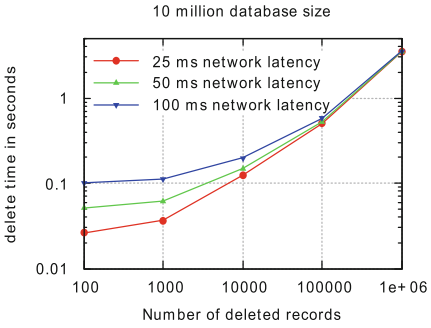

Fig. 9. Time taken for executing a delete query in *LWC*.

scheme achieves 450 queries per second. Though the comparisons are on the log scale, the results on individual data points indicate that *LWC* has a significantly higher throughput. *LWC* is also seen to easily outperform the results in Ishai's scheme.

We also provide the latency results for insert, update and delete operations with simulated end-to-end network latencies.

Figure 8 plots the times for an insert query including network latency in seconds. It is observed that *LWC* takes around over 0.1 s to insert 1K records and up to 100 s for inserting 10M records. We also observe that the effect of network latency diminishes when a large number of records are inserted.

Figure 10 plots the time for an update query with network latency. We can observe that *LWC* can update 1M records in just under 4 s. Again, the network latency effect diminishes when a large number of records are updated.

Finally, Fig. 9 plots the time for a delete query including network latency. LWC can delete 1M records in under 4 s. Note that, in *LWC*, the steps executed

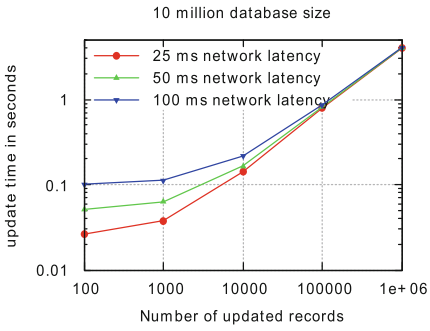

Fig. 10. Time taken for executing an update query in *LWC*.

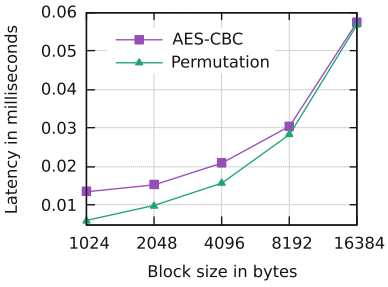

Fig. 11. The performance comparison between AES-CBC and permutation operations.

for the delete operation are similar to the update. The only difference is that in a delete operation there is not replacing of the old with new value like in an update.

To achieve OPP and resist against the collusion between the DBU and CS, the OPS could also encrypt each block with other cryptographic primitives, like AES-CBC, rather than permuting data cells. However, permuting data cells is more efficient than re-encryption. We did another test and compared the performance of permutation with AES-CBC encryption. AES-CBC with 256-bit key implemented in MIRACL 7.00 C library was used for the test. For permutation, we set the size of each data cell to 1 byte. Before getting all the required blocks, the OPS could first pre-generate new seeds for accessed blocks and pre-compute the new orders. As shown in Fig. 11, permutation is more efficient than AES-CBC when the block size is between 512 and 16384 bytes. When the block size is greater than 2^{16} bytes, the data cell can be set to 2 bytes, which will make the permutation more efficient.

We can conclude our analysis by discovering that even though *LWC* requires the OPS to perform most of the computation, its centralised nature does not degrade performance of the system. *LWC* achieves a high throughput performance and even with network latency simulations, returning 1 million records from a database of 10 million records takes less than a second including the time on the DBU to decrypt the returned results. When compared with a plaintext database, *LWC* results 10 times slower but achieves a high level of privacy. Finally, our comparison with other similar works, with all its current limitations, shows that *LWC* has a higher throughput for any result size while it still provides a very flexible key management scheme not supported by Stefanov's and Ishai's schemes. Moreover, *LWC* supports more privacy properties when compared with existing schemes.

8 Conclusion and Future Work

In this paper, we proposed *LWC*, a dynamic searchable encrypted scheme for hybrid outsourced databases with a full-fledged multi-user key management. *LWC* is a sub-linear scheme that does not leak information on its search pattern, access pattern, size pattern and operation pattern. The experimental results indicate that *LWC* is able to achieve high performance without requiring top of the line hardware. We have compared *LWC* with two relevant high-performance approaches: Stefanov *et al.* [11] and Ishai *et al.* [12]. *LWC* outperforms both while providing a higher level of privacy and a more flexible key management.

In our future work, we plan to extend the capabilities of the B^+ tree implementation to support range queries. It also remains to be explored how to move most of the operations to the untrusted cloud server, but without compromising on security.

References

1. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Ray, I., Li, N., Kruegel, C. (eds.) SIGSAC 2015, pp. 668–679. ACM (2015)
2. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Ray, I., Li, N., Kruegel, C. (eds.) SIGSAC 2015, pp. 644–655. ACM (2015)
3. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: S&P 2000, pp. 44–55. IEEE Computer Society (2000)
4. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: NDSS 2012. The Internet Society (2012)
5. Ostrovsky, R.: Efficient computation on oblivious rams. In: Ortiz, H. (ed.) STOC 1990, pp. 514–523. ACM (1990)
6. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM **43**(3), 431–473 (1996)
7. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) SIGSAC 2013, pp. 299–310. ACM (2013)
8. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
9. Williams, P., Sion, R.: Usable PIR. In: NDSS 2008. The Internet Society (2008)
10. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: Juels, A., Parno, B. (eds.) CCSW 2013, pp. 77–88. ACM (2013)
11. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS 2013, vol. 71, pp. 72–75 (2013)
12. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 90–107. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_6
13. Cui, S., Asghar, M.R., Galbraith, S.D., Russello, G.: Secure and practical searchable encryption: a position paper. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10342, pp. 266–281. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60055-0_14
14. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006, pp. 79–88. ACM (2006)
15. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_22
16. Jannink, J.: Implementing deletion in B+-trees. SIGMOD Rec. **24**, 33–38 (1995)

JACPoL: A Simple but Expressive JSON-Based Access Control Policy Language

Hao Jiang and Ahmed Bouabdallah(✉)

IMT Atlantique, Site of Rennes, 35510 Cesson-Sevigne, France
{hao.jiang,ahmed.bouabdallah}@imt-atlantique.fr

Abstract. Along with the rapid development of ICT technologies, new areas like Industry 4.0, IoT and 5G have emerged and brought out the need for protecting shared resources and services under time-critical and energy-constrained scenarios with real-time policy-based access control. The process of policy evaluation under these circumstances must be executed within an unobservable delay and strictly comply with security objectives. To achieve this, the policy language needs to be very expressive but lightweight and efficient. Many existing implementations are using XML (Extensible Markup Language) to encode policies, which is verbose, inefficient to parse, and not readable by humans. On the contrary, JSON (JavaScript Object Notation) is a lightweight, text-based and language-independent data-interchange format that is simple for humans to read and write and easy for machines to parse and generate. Several attempts have emerged to convert existing XML policies and requests into JSON, however, there are very few policy specification proposals that are based on JSON with well-defined syntax and semantics. This paper investigates these challenges, and identifies a set of key requirements for a policy language to optimize the policy evaluation performance. According to these performance requirements, we introduce JACPoL, a descriptive, scalable and expressive policy language in JSON. JACPoL by design provides a flexible and fine-grained ABAC (Attribute-based Access Control), and meanwhile it can be easily tailored to express a broad range of other access control models. This paper systematically illustrates the design and implementation of JACPoL and evaluates it in comparison with other existing policy languages. The result shows that JACPoL can be as expressive as existing ones but more simple, scalable and efficient.

Keywords: Real-time access control · Lightweight policy language
JSON · Fast policy evaluation

1 Introduction

Access control is an important security mechanism involving user specified policies to determine the actions that a principal can perform on resources. Typically,

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 56–72, 2018.
https://doi.org/10.1007/978-3-319-93524-9_4

the access requests are intercepted and analyzed by a PEP (Policy Enforcement Point), which then transfers the request details to a PDP (Policy Decision Point) for evaluation and authorization decision [1]. In most implementations, the stateless nature of PEP enables its ease of scale. However, the PDP has to consult the right policy set and apply the rules therein to reach a decision for each request and thus is often the performance bottleneck of policy-based access control systems. Therefore, a policy language determining how policies are expressed and evaluated is important and has a direct influence on the performance of the PDP.

Especially, in nowadays, protecting private resources in real-time has evolved into a rigid demand in domains such as home automation, smart cities, health care services and intelligent transportation systems, etc., where the environments are characterized by heterogeneous, distributed computing systems exchanging enormous volumes of time-critical data with varying levels of access control in a dynamic network. An access control policy language for these environments needs to be very well-structured, expressive but lightweight and easily extensible [2].

In this paper, we investigate the relationship between the performance of the PDP, the language that is used to encode the policies and the access requests that it decides upon, and identify a set of key requirements for a policy language to guarantee the performance of the PDP. We argue that JSON would be more efficient and suitable than other alternatives (XML, etc.) as a policy data format in critical environments. According to these observations, we proposed a simple but expressive access control policy language (JACPoL) based on JSON. A PoC (Proof of Concept) has been conducted through the implementation of JACPoL in a policy engine operated in reTHINK testbed [3]. At last we carefully positioned JACPoL in comparison with existing policy languages.

The main contribution of this work is therefore the definition of JACPoL, which utilizes JSON to encode a novel access control policy specification language with well-defined syntax and semantics. We identify key requirements and technical trends for future policy languages. We incidentally propose the new notion of *Implicit Logic Operators (ILO)*, which can greatly reduce the size and complexity of a policy set while providing fine-grained access control. We also elaborate on the applicability of JACPoL on ABAC model, RBAC model and their combinations or their by-products. Last but not least, our implementation leads to a novel and performant policy engine adopting the PDP/PEP architecture [1] and JACPoL policy language based on Node.js[1] and Redis[2].

The remainder of this paper is structured as follows. In Sect. 2, we refine our problematic by delimiting precisely its perimeter. In Sect. 3 we illustrate in depth with representative policy examples the design of our policy language in terms of the constructs, semantics and other important features like Implicit Logic Operators, combining algorithms and implementation. Section 4 further evaluates JACPoL and compares it with other existing access control policy specification languages. The ABAC-native nature of JACPoL is detailed in Sect. 5 along with a comprehensive discussion on other possible application of JACPoL to

[1] nodejs.org.
[2] redis.io.

ARBAC (Attribute-centric RBAC) and RABAC (Role-centric ABAC) security models. In Sect. 6 we summarize our work and discuss future research directions.

2 Problem Statement

In the past decades, a lot of policy languages have been proposed for the specification of access control policies using XML, such as EPAL [6], X-GTRBAC [7] and the standardized XACML [8]. Nevertheless, it is generally acknowledged that XACML suffers from providing poorly defined and counterintuitive semantics [9], which makes it not good in simplicity and flexibility. On the other hand, XML performs well in expressiveness and adaptability but sacrifices its efficiency and scalability, compared to which JSON is considered to be more well-proportioned with respect to these requirements, and even simpler, easier, more efficient and thus favored by more and more nowadays' policy designers [10–13].

To address the aforementioned inefficiency issues of the XML format, the XACML Technical Committee recently designed the JSON profile [18] to be used in the exchange of XACML request and response messages between the PEP and PDP. However, the profile does not define the specification of XACML policies, which means, after the PDP parses the JSON-formatted XACML requests, it still needs to evaluate the parsed attributes with respect to the policies expressed in XML. Leigh Griffin and his colleagues [12] have proposed JSONPL, a policy vocabulary encoded in JSON that semantically was identical to the original XML policy but stripped away the redundant meta data and cleaned up the array translation process. Their performance experiments showed that JSON could provide very similar expressiveness as XML but with much less verbosity. On the other hand, as much as we understand, JSONPL is merely aimed at implementing XACML policies in JSON and thus lacks its own formal schema and full specification as a policy specification language [19].

Major service providers such as Amazon Web Services (AWS) [20] have a tendency to implement their own security languages in JSON, but such kind of approaches are normally for proprietary usage thus provide only self-sufficient features and support limited use cases, which are not suitable to be a common policy language. To the best of our knowledge, there are very few proposals that combine a rich set of language features with well-defined syntax and semantics, and such kind of access control policy language based on JSON has not even been attempted before and as such JACPoL can be considered to be an original and innovative contribution.

3 JACPoL Detailed Design

This section presents JACPoL in depth. We first recall the foundations of JACPoL, and then introduce its structures with an overview of how an access request is evaluated with respect to JACPoL policies. After that, we describe the syntax and semantics in detail along with policy examples.

3.1 Fundamental Design Choices

The goal is to design a simple but expressive access control policy language. To achieve this, we beforehand introduce the important design decisions for JACPoL as below.

First, JACPoL is JSON-formatted [21].

Second, JACPoL is attribute-based by design but meanwhile supports RBAC [14]. When integrating RBAC, user roles are considered as an attribute (ARBAC) [26], or attributes are used to constrain user permissions (RABAC) [27], which obtains the advantages of RBAC while maintaining ABAC's flexibility and expressiveness.

Third, JACPoL adopts hierarchically nested structures similar to XACML. The layered architecture as shown in Fig. 1 not only enables scalable and fine-grained access control, but also eases the work of policy definition and management for policy designers.

Forth, JACPoL supports *Implicit Logic Operators* which make use of JSON built-in data structures (*Object* and *Array*) to implicitly denote logic operations. This allows a policy designer to express complex operations without explicitly using logical operators, and makes JACPoL policies greatly reduced in size and easier to read and write by humans.

Fifth, JACPoL supports *Obligations* to offer a rich set of security and network management features.

3.2 Policy Structure

JACPoL uses hierarchical structures very similar to the XACML standard [22]. As shown in Fig. 1, JACPoL policies are structured as *Policy Sets* that consist of one or more child policy sets or policies, and a *Policy* is composed of a set of *Rules*.

Because not all *Rules*, *Policies*, or *Policy Sets* are relevant to a given request, JACPoL includes the notion of a *Target*. A *Target* determines whether a *Rule*/*Policy*/*Policy Set* is applicable to a request by setting constraints on attributes using simple Boolean expressions. A *Policy Set* is said to be *Applicable* if the access request satisfies the *Target*, and if so, then its child *Policies* are evaluated and the results returned by those child policies are combined using the policy-combining algorithm; otherwise, the *Policy Set* is skipped without further examining its child policies and returns a *Not Applicable* decision. Likewise, the *Target* of a *Policy* or a *Rule* has similar semantics.

The *Rule* is the fundamental unit that is evaluated eventually and can generate a conclusive decision (*Permit* or *Deny* specified in its *Effect* field). The *Condition* field in a rule is a simple or complex Boolean expression that refines the applicability of the rule beyond the predicates specified by its target, and is optional. If a request satisfies both the *Target* and *Condition* of a rule, then the rule is applicable to the request and its *Effect* is returned as its decision; otherwise, *Not Applicable* is returned.

For each *Rule*, *Policy*, or *Policy Set*, an *id* field is used to be uniquely identified, and an *Obligation* field is used to specify the operations which should be performed (typically by a PEP) before or after granting or denying an access request, while a *Priority* is specified for conflict resolution between different *Rules*, *Policies*, or *Policy Sets*.

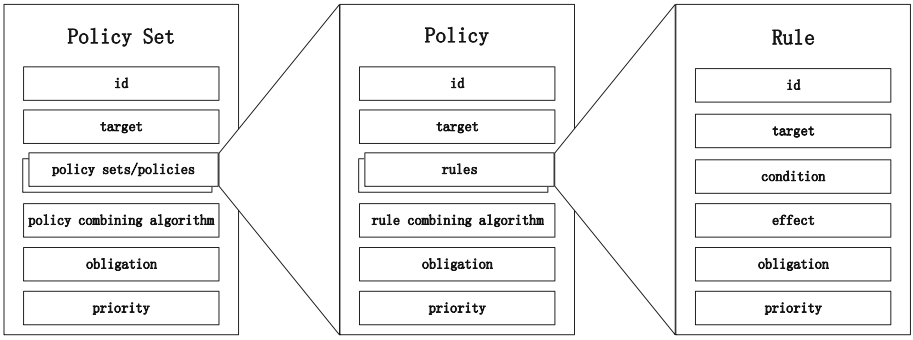

Fig. 1. JACPoL's hierarchical nested structure

3.3 Syntax and Conventions

JACPoL uses JSON syntax to construct and validate its policies. A policy must follow correct JSON syntax to take effect. In this subsection, we do not provide a complete description of what constitutes valid JSON [21]. However, below is a list of fundamental characteristics of JSON:

- JSON is built on two universal data structures: *object* and *array*.
- An *object* is denoted by braces ({}) that can hold multiple name-value pairs. For each name-value pair, a colon (:) is used to separate the name and the value, whilst multiple name-value pairs are separated by comma (,) as in the following example: {"id": 1, "effect": "permit"}.
- An *array* is denoted by brackets ([ ]) that can hold multiple values separated by commas (,) as in the following example: ["Monday", "Friday", "Sunday"].
- A *value* can be a string in double quotes, or a number, or a Boolean value (*true* or *false*), or *null*, or an *object* or an *array*.
- Whitespace can be inserted between any pair of JSON tokens ({ } [ ] " , :).

In the subsequent subsections, we will elaborate on the syntax and semantics for each policy element. To illustrate better, the following conventions are used:

- The following characters are special characters used in the description of the grammar and are not included in the policy syntax: < > ... ( ) |.
- If an element allows multiple values, it is indicated using the repeated values, commas, and an ellipsis (...). Example: [<rule_block>, <rule_block>, ...].
- A question mark (?) following an element indicates that element is optional. Example: {"condition"?: <boolean_expression>}.

- A vertical line (|) between elements indicates alternatives. Parentheses define the scope of the alternatives. The default value is underlined if the field is optional. Example: {"algorithm"?: ("permitOverrides"|"firstApplicable")}.
- Elements that must be literal strings are enclosed in double quotation marks.

3.4 Policy Sets, Policies and Rules

This subsection describes the grammar of the *Policy Set*, the *Policy* and the *Rule*. In JACPoL, a policy set, a policy, or a rule always starts and ends with a brace, which denotes a policy set block, a policy block, or a rule block.

Policy Set Block. Figure 2 describes the grammar of the policy set block, which is composed of six name-value pairs that exactly correspond to the six elements of a policy set. As shown in the figure, the "id" field is a string which can be either numeric or descriptive to uniquely identify a policy set. The "target" specifies a Boolean expression indicating the resources, subjects, actions or the environment attributes to which the policy set is applied. The "policies" stores a list of policy blocks with each one corresponding to a policy. The "algorithm" field specifies the name of a decision-combining algorithm to compute the final decision according to the results returned by its child policies. The "obligation" specifies actions to take in case a particular conclusive decision (*Permit* or *Deny*) is reached. The "priority" provides a numeric value indicating the weight of the policy set when its decision conflicts with other policy sets under the *highest-Priority* algorithm.

```
{
  "id":          <string>,
  "target"?:     <boolean_expression>,
  "policies":    [<policy_block>, <policy_block>, ...],
  "algorithm"?:  ("permitOverrides"|"denyOverrides"|"firstApplicable"|"highestPriority"),
  "obligation"?: <obligation_statement>,
  "priority"?:   <number>
}
```

Fig. 2. Grammar of the policy set block

Note that elements like *target, algorithm, obligation* and *priority* are optional and, if omitted, the predefined default values would be taken (e.g., target: *true*, algorithm: *firstApplicable*, obligation: *null*, priority: 0.5).

Policy Block. As shown in Fig. 3, a policy block contains an id, a target, an algorithm, an obligation and a priority similar to a policy set. The difference is, it has a "rules" list that holds one or more rule blocks instead of policy blocks.

Rule Block. Figure 4 describes the grammar of the rule block. Unlike a policy set block or a policy block, a rule block does not contain any leaf nodes like child policies or child rules and thus a decision-combining algorithm field is not needed

```
{
  "id":          <string>,
  "target"?:     <boolean_expression>,
  "rules":       [<rule_block>, <rule_block>, ...],
  "algorithm"?:  ("permitOverrides"|"denyOverrides"|"firstApplicable"|"highestPriority"),
  "obligation"?: <obligation_statement>,
  "priority"?:   <number>
}
```

Fig. 3. Grammar of the policy block

either. Instead, it possesses a "condition" element that specifies the condition for applying the rule, and an "effect" element that, if the rule is applied, would be the returned decision of the rule as either *Permit* or *Deny*. In comparison to a target, a condition is typically more complex and often includes functions (e.g., "greater-than") for the comparison of attribute values, and logic operations (e.g., "and", "or") for the combination of multiple conditions. If either the target or the condition is not satisfied, a *Not Applicable* would be taken as the result instead of the specified effect. Note that the *Condition* is by default true if omitted.

```
{
  "id":          <string>,
  "target"?:     <boolean_expression>,
  "effect":      ("permit"|"deny"),
  "condition"?:  <boolean_expression>,
  "obligation"?: <obligation_statement>,
  "priority"?:   <number>
}
```

Fig. 4. Grammar of the rule block

3.5 Targets and Conditions

As aforementioned, a *Target* or a *Condition* is a Boolean expression specifying constraints on attributes such as the subject, the resource, the environment, and the action of requests. The Boolean expression of a *Target* is often simple and very likely to be just a test of string equality, but that of a condition can be sometimes complex with constraints on multiple attributes (*attribute conditions*).

Attribute Condition is a simple Boolean expression that consists of a key-value pair as shown below:

```
{"<attribute_expression>": <condition_expression>}
```

The key is an *attribute expression* in string format that specifies an attribute or a particular computation between a set of attributes; the value is a *condition expression*, which is a JSON block composed of one or more operator-parameter

pairs specifying specifically the requirements that the *attribute expression* needs to meet. The simplest format of an *attribute condition* is to verify the equality /inequality between the attribute (e.g., time) and the parameter (e.g., 10:00:00) using comparative operators (e.g., greater-than, less-than, equal-to, etc.):

```
{"<attribute>": {"<comparative_operator>": <parameter>}}
```

However, there are also cases where we have multiple constraints (operator-parameter pairs) on the same attribute, connected by logical relations like *AND, OR, NOT*, which are respectively denoted by the keywords *allOf*, *anyOf* and *not*.

Logical Operators. JACPoL uses logical operators in a form of constructing key-value pairs. The logical operator is the key and, depending on the number of arguments, *allOf* and *anyOf* operators are to be followed by an *array* ([ ]) of multiple constraints, while the *not* operator is to be followed by an *object* ({ }). An *allOf* operation would be evaluated to true only if all subsequently included constraints are evaluated to true, but an *anyOf* operation would be true as long as there is at least one of the constraints which is true. An *not* operation would be true if the followed constraint is evaluated to false. Logical operators can be nested to construct logical relations such as *not any of, not all of*. For example, an *attribute condition* containing multiple constraints with nested logical operators as below:

```
{"sumOf x y": {"not": {"anyOf": [
        {"between": "j k"},
        {"equals": "z"}]}}}
```

in which the string "sumOf x y" is an *attribute expression*. The keyword *sumOf* defines a function to compute the sum of attributes x and y, which is to be evaluated by the subsequent *condition expression*. Please note that a parameter like j, k or z can also possibly be another attribute instead of an explicit value.

In addition, logical operators can also be used to combine multiple *attribute conditions* in order to express complex constraints on more than one attribute easily and flexibly. As an example, the condition below expresses constraints on two attributes and would be evaluated to true only when both (*allOf* the two) constraints are met:

```
{"allOf": [<attribute_condition>, <attribute_condition>]}
```

Less Is More: Implicit Logical Operators. A complex condition might contain many logical operators which make the policy wordy and hard to read. To overcome this, we make use of JSON's built-in data structures, *object* and *array*, and define following implicit logical operators as alternatives to *allOf* and *anyOf*:

- An *object* is implicitly an *allOf* operator which would be evaluated to true only if all the included key-value pairs are evaluated to true.
- An *array* is implicitly an *anyOf* operator which would be evaluated to true as long as at least one of its elements is evaluated to true.

For example, below is a condition statement using implicit logical operators to verify if it is working hour.

```
{
  "time":{"between": ["09:00 12:00", "14:00 18:00"]},
  "weekday": {"not": {"equals": ["saturday", "sunday"]}}
}
```

As a comparison, below is for the same verification with explicit logical operators.

```
{
  "allOf": [{
      "time": {"anyOf": [
          {"between": "09:00 12:00"},
          {"between": "13:00 18:00"}]}
    },{
      "weekday": {"not": {"anyOf": [
          {"equals": "saturday"},
          {"equals": "sunday"}]}}}]
}
```

Apparently, implicit operators save a lot size and make policies more readable, which has later turned out to be very useful in our policy engine implementation.

3.6 Combining Algorithms

In JACPoL, policies or rules may conflict and produce totally different decisions for the same request. JACPoL resolves this by adopting four kinds of decision-combining algorithms: *Permit-Overrides, Deny-Overrides, First-Applicable*, and *Highest-Priority*. Each algorithm represents a different way for combining multiple local decisions into a single global decision:

- *Permit-Overrides* returns *Permit* if any decision evaluates to *Permit*; and returns *Deny* if all decisions evaluate to *Deny*.
- *Deny-Overrides* returns *Deny* if any decision evaluates to *Deny*; returns *Permit* if all decisions evaluate to *Permit*.
- *First-Applicable* returns the first decision that evaluates to either of *Permit* or *Deny*. This is very useful to shortcut policy evaluation.
- *Highest-Priority* returns the highest priority decision that evaluates to either of *Permit* or *Deny*. If there are multiple equally highest priority decisions that conflict, then *deny-overrides* algorithm would be applied among those highest priority decisions.

Please note that for all of these combining algorithms, *Not Applicable* is returned if not any of the child rules (or policies) is applicable. Hence, the set of possible decisions is 3-valued.

3.7 Obligations

JACPoL includes the notion of obligation. An *Obligation* optionally specified in a Rule, a Policy or a PolicySet is an operation that should be performed by the PEP in conjunction with the enforcement of an authorization decision. It can be triggered on either Permit or Deny. We employ the format as below to express obligations in JACPoL:

```
{"<decision>": {"<operation>": [<parameter>, <parameter>,
    ...]}}
```

For example, the obligation below is for the access control of a document.

```
{
  "permit": {"watermark": ["DRAFT"]},
  "deny": {
    "feedback": ["ACCESS DENIED"],
    "notify": ["admin@gmail.com", "hr@gmail.com"]
  }
}
```

It specifies that if an access request is denied, the user would be informed with an access denied message and the administrator and HR would also be notified; if an access request is approved, watermark the document "DRAFT" before delivery. It worths to be mentioned that the referred obligations have an eminently locale nature in the sense that their execution is the exclusive prerogative of the PEP which can possibly rely on the information available in the PIP [1]. More general and distributed obligations deserve a dedicated investigation.

3.8 Implementation

We implemented JACPoL in a Javascript/Node.js/Redis based policy engine [24] which is available on Github. As shown in Fig. 5, the policy engine employs the classical PDP/PEP architecture [1]. The PDP retrieves policies from the PRP (Policy Retrieval Point), and evaluates authorization requests from the PEP by examining the finite relevant attributes against the policies. If more attributes are required to reach a decision, the PDP will request the PIP (Policy Information Point) as an external information source. The latter may also be requested in the case of obligations.

The non-blocking nature of Node.js allows the system to provide an efficient and scalable access control, and a Redis server was employed to enable flexible and high-performance data persistence and caching. In order to validate its functionality, we have deployed this policy engine on a messaging node of reTHINK project [23]. This reTHINK policy engine [24] adopted the ABAC model and customized the vocabulary of JACPoL for the requirements of reTHINK framework. With JACPoL based lightweight policies, Node.js based non-blocking I/O, and Redis based fast caching, it provided a highly performant access control according to various tailored attributes in an expressive and flexible way [25]. In

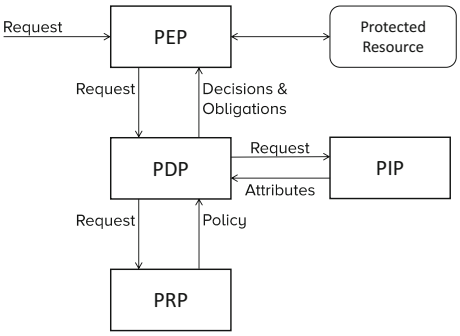

Fig. 5. Policy engine architecture

reTHINK, in addition to comparative operators *greaterThan, lessThan, equalsTo*, JACPoL is extended to support more operators as listed in Fig. 6 with a rule example:

Other Operators
x **in** [a, b, c]
x **between** "a b"
x **contains** "c"
x **like** "/resources/*"
x **exists**

```
{
    "id": "dpt-01-ac-01",
    "target": {"type": {"equalsTo": "create"}},
    "condition": {
        "weekday": {"in": ["saturday", "sunday"]},
        "time": {"between": "18:00 07:00"},
        "url": {"like": "*/core/*/contactlist"}
    },
    "effect": "deny"
}
```

Fig. 6. Other supported operators (left) and a rule example (right)

4 Comparative Analysis

This section evaluates JACPoL with a comprehensive comparison to other pre- and post-XACML policy languages, which respectively are JSONPL [19], AWS IAM [20], XACML [8], Ponder [5], Rei [29], XACL [30], KAoS [31], EPAL [6], and ASL [32], followed by a simple quantitative comparison with XACML in terms of processing delay. To begin with, we have identified the following requirements for an access control policy language to meet the increasing needs of security management for today's ICT systems:

- *Expressiveness* to support wide range of policy needs and be able to specify various complex, advanced policies that a policy maker intend to express [4].
- *Extensibility* to cater for new features or concepts of policy in the future [5].
- *Simplicity* to ease the policy definition and management tasks for the policy makers with different levels of expertise. This includes both conciseness and readability to avoid long learning curve and complex training.

- *Efficiency* to ensure the speed for machines to parse the policies defined by humans. This can be affected by policy structure, syntax, and data format.
- *Scalability* to ensure the performance as the network grows in size and complexity. This is important especially in large-scale or multi-domain networks.
- *Adaptability* to be compatible with any access control tasks derived from an ICT system. Any user could directly tailor the enforcement code and related tool set provided by the policy language to their authorization systems.

Table 1 shows the complete evaluation of these policy languages. In the table, '✓' and '✔' respectively indicate 'support' and 'strongly support', while '+', '++', '+++' and '++++' mean 'poor', 'good', 'very good' and 'excellent'. The comparison mainly focuses on their design and implementation choices regarding *authorization, obligation, index, syntax and scheme*, and their performance with respect to the six previously defined criteria. Among these features, *index* refers to whether there exists a special item for policy engine to retrieve the required policies more efficiently.

Table 1. Evaluation and comparison between JACPoL and other policy languages

Policy Languages	Year	Authorization	Obligation	Index	Syntax	Scheme	Expressiveness	Extensibility	Simplicity	Efficiency	Scalability	Adaptability
JACPoL	2017	✔	✓	✔	JSON-based	ABAC	++++	++	+++	+++	+++	+++
JSONPL	2012	✔		✔	JSON-based	ABAC	++	++	+++	+++	+++	++
AWS IAM	2010	✔			JSON-based	RBAC	++	++	+++	+++	++	+
XACML	2003	✔	✓	✔	XML-based	ABAC	+++	+++	+	++	++	+++
Rei	2003	✔	✔		Logic-based	OBAC	++++	+++	+	++	+++	+++
EPAL	2003	✔	✓	✔	XML-based	RBAC	+++	+++	+	++	++	+++
Ponder	2001	✔	✔		Specific	RBAC	+++	+	++	++	+++	+++
XACL	2000	✔			XML-based	RBAC	+	+	+	++	+	+
KAoS	1997	✔	✔		OWL-based	OBAC	++++	+++	+	++	+++	+++
ASL	1997	✔			Logic-based	RBAC	+	+	+	++	+	++

Like many other languages, JACPoL provides support for authorization and obligation capabilities as previously introduced. In addition, it includes a concept of *Target* within each *Policy Set*, *Policy* and *Rule* to allow efficient policy index. In terms of expressiveness, JACPoL, Rei and KAoS extensively support the specification of constraints, which can be set on numerous attributes in a flexible expression [33].

On the other hand, compared to XML-based languages, JSON-based JACPoL is simpler and more efficient, but meanwhile, we have to admit that JSON is less sophisticated than XML, which accordingly may make JACPoL less extensible. JACPoL is scalable and the reasons are twofold: first, its efficient performance in policy index and evaluation allows it to deal with complex policies under a large-scale network environment; second, its concise semantics and lightweight data representation make it easily replicable and transferable for distributed systems. As for adaptability, compared to other languages, the application specific nature of AWS IAM makes it relatively harder to be adapted to other systems.

A more comprehensive, quantitative and systematic performance evaluation (in terms of speed, memory consumption, etc.) of JACPoL is in progress at

the moment and would be detailed in a future paper [34]. However, we present the results of a preliminary test as below in Figs. 7 and 8 which hopefully can provide some insights on the outperformance of JACPoL over XACML. We assessed respectively how both languages' policy processing time increases with the growth of nesting layers (policy depth) and with the growth of sibling rules (policy scale). First we used the two languages to express the same policy set with 400 recursively nesting child policies, and recorded the delay when each child policy was evaluated until the deepest one was reached. The result is depicted in Fig. 7. Then we did the same with a policy set containing 400 sibling child policies and got the result in Fig. 8. Each experiment was repeated 1000 times conducted using Python on a Windows 10 PC with 16 GB memory and a 2.6 GHz Intel core i7-6700HQ processor. From this simple test we can preliminarily see that JACPoL is more efficient and scalable and processed with less latency than XACML thanks to its JSON syntax and well-defined semantics.

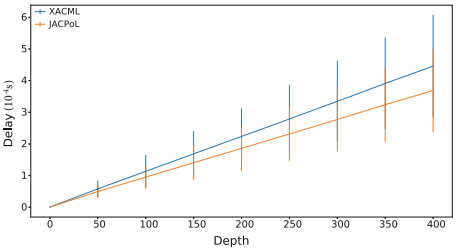

Fig. 7. Effect of policy depth

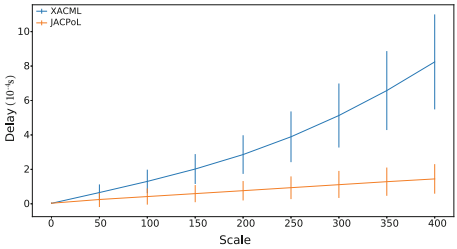

Fig. 8. Effect of policy scale

5 Application of JACPoL to Security Models

5.1 RBAC vs ABAC

In policy-based access control systems, a request for access to protected resources is evaluated with respect to a policy that defines which requests are authorized. The policy itself conforms to a security model upstream chosen by the system security administrator because it elegantly copes with the constraints associated with the targeted information system. In the RBAC model, roles are pre-defined and permission sets for resources are pre-assigned to roles. Users are then granted one or more roles in order to receive access to resources [16]. ABAC, on the other hand, relies on user attributes for access decisions. These include: *subject attributes*, which are attributes concerning the actor being evaluated; *object attributes*, which are attributes of the resource being affected; *action attributes*, which are attributes about the operation being executed; and *environment attributes*, which provides other contextual information such as time of the day, etc. [15]. Generally speaking, RBAC is simple, static and auditable, but is not expressive nor context-aware, while ABAC, by contrast, provides

fine-grained, flexible and dynamic access control in realtime but is complex and unauditable. Combining these two models judiciously to integrate their advantages thus becomes an essential work in recent research [16,17,27,28].

5.2 Attribute-Centric RBAC Application

JACPoL can be implemented to express permission specification policies (PSP) in an attribute-centric RBAC model. For example, Fig. 9 defines a policy set with each policy specifying permissions that are associated to the targeted role. When evaluating a request, the PDP first retrieves all the roles (e.g., from the PIP) that are pre-assigned to the requester, and then examines the permission policies that are associated to these roles to reach a decision. Unlike other traditional statically defined RBAC permissions, JACPoL allows its permissions to be expressed in a quite dynamic and flexible way similar to ABAC. Please note that the role attribute is suggested to be placed as the target for the topmost level of policies in order to allow an easier view of user permissions as shown in Fig. 9.

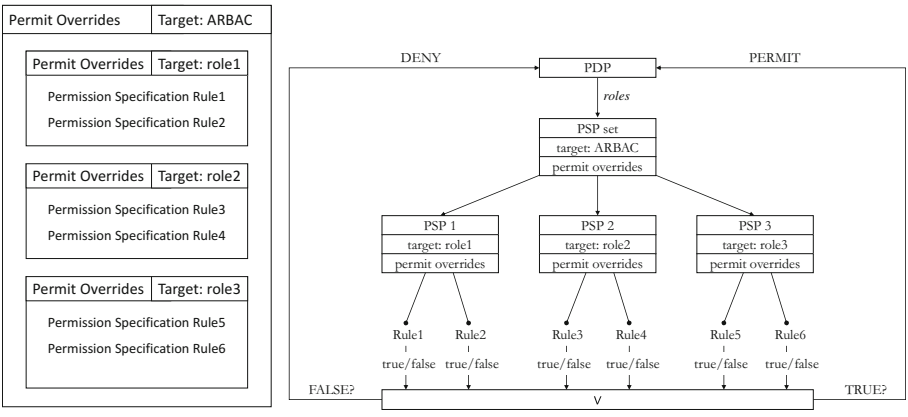

Fig. 9. An example ARBAC permission specification policy and its structure tree

5.3 Role-Centric ABAC Application

JACPoL can also be used to implement a language for permission filtering policies (PFP) in a role-centric ABAC model [27]. Similar to the ABAC model in Fig. 5, but in addition to external attributes, the PDP also relies on the PIP to get role permission sets which, as defined by NIST RBAC model, specify the maximum set of available permissions that users can have. These permission sets can be further constrained by the filtering policies based on JACPoL, as shown in Fig. 10. Note that this time the target of each PFP maps each object to a subset of the filtering rules. At the same time, the target and condition of each rule determine whether or not the rule is applicable. The applicable filtering rules are invoked one by one against each of the permissions in the permission set. If

any of the rules return FALSE, the permission is then blocked and removed from the available permission set for the current session. At the end of this process, the final available permission set available to users therefore will be the intersection of P and R, where P is the set of permissions assigned to the subject's active roles and R is the set of permissions specified by the applicable JACPoL rules [28].

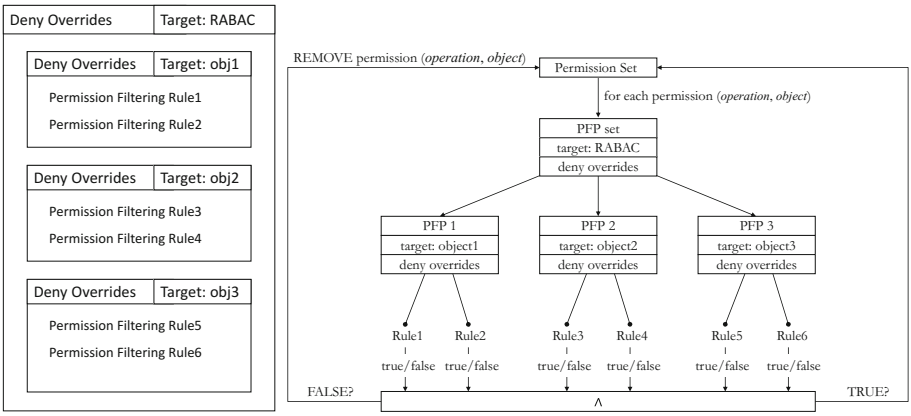

Fig. 10. An example RABAC permission filtering policy and its structure tree

6 Conclusion

Traditionally, performance has not been a major focus in the design of access control systems. Applications are emerging, however, that require policies to be evaluated with a very low latency and high throughput. Under this background, we designed and implemented JACPoL, a fast JSON-based, attribute-centric and light-weight access control policy language. JACPoL provides a good solution for policy specification and evaluation in such applications with low processing delay. We evaluated our policy language with respect to a set of representative criteria in comparison with other existing policy languages. The evaluation showed that JACPoL can be as expressive as XACML but more simple, scalable and efficient. On the other hand, JACPoL leaves room for future improvements in many areas. For example, obligation capabilities can be further enhanced and delegation support can be formally introduced. By priority, we are currently conducting a more comprehensive experimental performance evaluation with extensive policy sets for various real-world use cases, in which a more systematic and quantitative evaluation criteria (e.g., speed, memory usage, etc.) would be considered.

Acknowledgement. This work has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No. 645342, project reTHINK. We gratefully acknowledge support from our colleagues in this

project, Jamal Boulmal (Apizee), Jean-Michel Crom and Simon Becot (Orange Labs). This work would hardly be possible without their valuable suggestions and help.

References

1. Yavatkar, R., Pendarakis, D., Guerin, R.: A Framework for Policy-Based Admission Control. IETF, RFC 2753, January 2000
2. Borders, K., Zhao, X., Prakash, A.: CPOL: high-performance policy evaluation. In: The 12th ACM Conference on Computer and Communications Security. ACM (2005)
3. reTHINK Project Testbed: Deliverable D6.1: Testbed Specification (2016). https://bscw.rethink-project.eu/pub/bscw.cgi/d35657/D6.1%20Testbed%20specification.pdf. Accessed 17 May 2017
4. He, L., Qiu, X., Wang, Y., Gao, T.: Design of policy language expression in SIoT. In: Wireless and Optical Communication Conference, pp. 321–326. IEEE (2013)
5. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Sloman, M., Lupu, E.C., Lobo, J. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44569-2_2
6. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language (EPAL). IBM Research, March 2003
7. Bhatti, R., Ghafoor, A., Bertino, E., Joshi, J.B.: X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **8**(2), 187–227 (2005)
8. OASIS XACML Technical Committee: eXtensible access control markup language (XACML) Version 3.0. Oasis Standard, OASIS (2013). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. Accessed 17 May 2017
9. Crampton, J., Morisset, C.: PTaCL: a language for attribute-based access control in open systems. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 390–409. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28641-4_21
10. Crockford, D.: JSON – The fat-free alternative to XML, vol. 2006. http://www.json.org/fatfree.html. Accessed 17 May 2017
11. El-Aziz, A.A., Kannan, A.: JSON encryption. In: 2014 International Conference on Computer Communication and Informatics (ICCCI). IEEE (2014)
12. Griffin, L., Butler, B., de Leastar, E., Jennings, B., Botvich, D.: On the performance of access control policy evaluation. In: 2012 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), pp. 25–32. IEEE (2012)
13. W3schools: JSON vs XML. www.w3schools.com/js/js_json_xml.asp. Accessed 24 May 2017
14. Ferraiolo, D.F., Kuhn, D.R.: Role-based Access Controls. arXiv preprint arXiv: 0903.2171, 12 March 2009
15. Hu, V.C., Ferraiolo, D., Kuhn, R., et al.: Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800.162 (2013)
16. Empower ID: Best practices in enterprise authorization: The RBAC/ABAC hybrid approach. Empower ID, White paper (2013)
17. Coyne, E., Weil, T.R.: ABAC and RBAC: scalable, flexible, and auditable access management. IT Prof. **15**(3), 0014–16 (2013)

18. David, B.: JSON Profile of XACML 3.0 Version 1.0. XACML Committee Specification 01, 11 December 2014. http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cs01/xacml-json-http-v1.0-cs01.pdf. Accessed 26 May 2017
19. Steven, D., Bernard, B., Leigh, G.: JSON-encoded ABAC (XACML) policies. FAME project of Waterford Institute of Technology. Presentation to OASIS XACML TC concerning JSON-encoded XACML policies, 30 May 2013
20. Amazon Web Services: AWS Identity and Access Management (IAM) User Guide. http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html. Accessed 27 May 2017
21. ECMA International: ECMA-404 The JSON Data Interchange Standard. http://www.json.org/. Accessed 27 May 2017
22. Ferraiolo, D., et al.: Extensible access control markup language (XACML) and next generation access control (NGAC). In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control. ACM (2016)
23. reTHINK Project. github.com/reTHINK-project/. Accessed 27 May 2017
24. reTHINK CSP Policy Engine. github.com/reTHINK-project/dev-msg-node-nodejs/tree/master/src/main/components/policyEngine. Accessed 27 May 2017
25. reTHINK Deliverable 6.4: Assessment Report, reTHINK H2020 Project
26. Obrsta, L., McCandlessb, D., Ferrella, D.: Fast semantic attribute-role-based access control (ARBAC) in a collaborative environment. In: 2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), Pittsburgh, PA, USA, 14–17 October 2012
27. Jin, X., Sandhu, R., Krishnan, R.: RABAC: role-centric attribute-based access control. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 84–96. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33704-8_8
28. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. Computer **43**(6), 79–81 (2010)
29. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: IEEE 4th International Workshop on Proceedings of Policies for Distributed Systems and Networks, POLICY 2003. IEEE (2003)
30. Hada, S., Kudo, M.: XML Access Control Language: provisional authorization for XML documents (2000)
31. Uszok, A., Bradshaw, J.M., Jeffers, R.: KAoS: a policy and domain services framework for grid computing and semantic web services. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 16–26. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24747-0_2
32. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: Proceedings of IEEE Symposium on Security and Privacy. IEEE (1997)
33. Neuhaus, C., Polze, A., Chowdhuryy, M.M.: Survey on healthcare IT systems: standards, regulations and security. No. 45. Universitätsverlag Potsdam (2011)
34. Jiang, H., Bouabdallah, A.: Towards A JSON-Based Fast Policy Evaluation Framework. Work in progress

Trusted Execution

EmLog: Tamper-Resistant System Logging for Constrained Devices with TEEs

Carlton Shepherd(✉), Raja Naeem Akram, and Konstantinos Markantonakis

Smart Card and Internet of Things Security Centre, Information Security Group, Royal Holloway, University of London, Surrey, UK
{carlton.shepherd.2014,r.n.akram,k.markantonakis}@rhul.ac.uk

Abstract. Remote mobile and embedded devices are used to deliver increasingly impactful services, such as medical rehabilitation and assistive technologies. Secure system logging is beneficial in these scenarios to aid audit and forensic investigations particularly if devices bring harm to end-users. Logs should be tamper-resistant in storage, during execution, and when retrieved by a trusted remote verifier. In recent years, Trusted Execution Environments (TEEs) have emerged as the go-to root of trust on constrained devices for isolated execution of sensitive applications. Existing TEE-based logging systems, however, focus largely on protecting server-side logs and offer little protection to constrained source devices. In this paper, we introduce EmLog – a tamper-resistant logging system for constrained devices using the GlobalPlatform TEE. EmLog provides protection against complex software adversaries and offers several additional security properties over past schemes. The system is evaluated across three log datasets using an off-the-shelf ARM development board running an open-source, GlobalPlatform-compliant TEE. On average, EmLog runs with low run-time memory overhead (1 MB heap and stack), 430–625 logs/second throughput, and five-times persistent storage overhead versus unprotected logs.

Keywords: System logging · Embedded security · Trusted computing

1 Introduction

System logs record features such as user activity, resource consumption, peripheral use and error details. Logs are also used to enforce user accountability and to establish audit trails for forensics, event reconstruction and intrusion detection [16]. Consequently, logs are routinely targeted by attackers to conceal evidence of wrongdoing, and should be stored securely to preserve the auditability of a compromised system – as recommended by NIST [16] and ISO 27001:2013 [14]. Not only should logs be stored in a way that cryptographically preserves their confidentiality and integrity, but trusted computing primitives, e.g. Trusted Platform Modules (TPMs), have been identified as desirable in

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 75–92, 2018.
https://doi.org/10.1007/978-3-319-93524-9_5

existing proposals [4,32]. Such technologies have been used for tamper-resistant storage of logging keys, performing cryptographic operations, and providing evidence of platform integrity to third-party verifiers using remote attestation.

However, the advent of low-cost, mass-produced Internet of Things (IoT) devices complicates the use of trusted computing for tamper-resistant logging. Numerous proposals suggest using IoT devices for remote health monitoring [24], identifying fires and gas leakages [6], and detecting falls and injuries in the homes of the elderly and disabled [26] – all of which are natural applications for tamper-resistant logging. Unfortunately, discrete hardware TPMs, which underpin many existing proposals, cannot directly host arbitrary applications without additional processes, such as launching and locally attesting applications from a TPM-backed virtual machine [4,25]. Including such processes within the device's Trusted Computing Base (TCB) – the set of software and hardware components essential to its security – widens the scope for introducing security and performance defects [21,31]. One promising solution is the Trusted Execution Environment (TEE), which offers TPM-like functionality alongside strong isolated execution of critical applications, while using the core execution hardware of conventional operating systems. TEEs have become widely-deployed in recent years, notably in the form of Intel Software Guard eXtensions (SGX) and TEEs built on ARM TrustZone. Indeed, Trustonic estimated that one billion devices contained their TrustZone-based TEE alone in early 2017 [33]. However, TEE-based logging schemes – discussed in Sect. 2 – have hitherto applied only server-side TEEs to protect logs transmitted from remote devices.

In this paper, we present EmLog, which leverages the GlobalPlatform TEE and ARM TrustZone for protecting logs *at source* on mobile and embedded systems. EmLog offers further security benefits over past work, including public verifiability of log origin, resilience to TEE key compromise, and supports secure I/O with peripheral devices. After reviewing related work (Sect. 2), we formalise the requirements and threat model in Sect. 4. EmLog is implemented on an off-the-shelf ARM development board hosting OP-TEE [19] – an open-source and GlobalPlatform-compliant TEE that uses TrustZone (Sect. 6) – and evaluated using three datasets in Sect. 7. Finally, we conclude our work in Sect. 8 and identify future areas of research. To our knowledge, this is the first attempt at preserving logs on constrained devices using a standardised TEE. The contributions of this paper are: **(1)**, the development of a novel secure logging scheme for creating tamper-resistant logs with trust assurances, tailored for ARM-based constrained devices, like wearables and sensing platforms; and **(2)**, a test-bed implementation using a GlobalPlatform-compliant TEE that uses ARM TrustZone, with performance benchmarks across three datasets. The results indicate that EmLog has low run-time memory footprint, five-times persistent storage overhead, and 430–625 logs/sec throughput.

2 Related Work

Existing proposals may be categorised as: **(1)**, *secure untrusted system logging*, focusing on cryptographic methods for detecting tampered logs on untrusted

platforms; and **(2)**, *trusted logging*, for applying trusted hardware primitives for log preservation. We briefly examine key proposals and their contributions.

2.1 Secure Untrusted System Logging

Schneier and Kelsey [27] propose the use of MACs with linear one-way hash chains to protect log integrity. Each chain entry is found by successively hashing the log content with the previous log's hash, which is accompanied by a MAC keyed under the hash of the previous MAC key. The initial key is a pre-shared key (PSK) between the logging device and a trusted verifier, which allows the MAC hash chain to be recomputed and verified. Bellare and Yee [3] propose a similar scheme using the formalised notion of *forward integrity* in which it is computationally infeasible to alter past entries after a key compromise. This is achieved by updating the secret key at regular time intervals (epochs) using an update process based on a chain of pseudo-random functions to key the log MACs in each epoch. Holt [13] proposed Logcrypt, which uses public-key cryptography alongside MACs to achieve *public verifiability*, so third-parties can authenticate the origin of log entries without knowledge of a secret PSK – shortfalls of [3,27]. Ma and Tsudik [20] introduce FssAgg, which uses an aggregated chain of signatures to achieve public verifiability and to thwart *truncation attacks*, where an attacker aims to delete a tail-end subset of log entries. Yavuz et al. [34] proposed LogFAS, which addresses both challenges with better storage and computational complexity than [13,20] using the Schnorr signature scheme. Recently, Hartung [12] presented four attacks against LogFAS [34] and two variants of FssAgg [20], which enables secret key recovery and log forgery; as a result, both schemes are dissuaded from use.

2.2 Secure Logging with Trusted Hardware

Early work by Chong et al. [7] explored trusted hardware (Java iButton) to protect the initial PSK of the Schneier and Kelsey scheme [27]. Later, Sinha et al. [32] suggested a using a TPM with a forward integrity scheme based on branched key chaining. Logs are divided into epochs (blocks), each comprising a sequence of hash-chained log entries (sub-epochs). The root entries of each epoch are hash-chained with past epochs, which creates a two-dimensional hash chain to prevent *re-ordering attacks* in which an attacker re-orders log blocks to mislead auditors. For each new epoch, the previous epoch's logs are securely stored using the TPM's seal functionality, which encrypts the logs with a TPM-bound key so only that particular TPM can decrypt/'unseal' them. Böck et al. [4] explore the use of AMD's Secure Virtual Machine (SVM) – an early inception of the TEE – for launching a `syslog` client daemon and logging application from the TPM's secure boot chain. The logger executes with access to TPM-bound key-pairs for encrypting and signing log entries. Upon request, the logs are decrypted and transmitted to the verifying party; the TPM keys are certified for authenticating that signed logs originated from the SVM.

Nguyen et al. [23] propose streaming medical logs to a server application in Intel SGX (see Sect. 3) that applies the tamper-resistance. Logs are sent to the Intel SGX application ('enclave') over TLS, which computes a hash chain comprising a signature of each record; TPMs are used to authenticate the medical devices to the server, and on the server's end to securely store log hash chains using its sealing mechanism. Karande et al. [15] introduce SGX-Log, which protects server-side device logs received from remote devices. SGX-Log, like [32], uses block-based hash chains with SGX's secure storage for log integrity and confidentiality. The authors note that continual sealing also provides resilience to attacks in which large volumes of logs in memory are lost due to an unauthorised power loss. Remote attestation is also suggested to authenticate the server enclave before transmitting the logs. The proposed scheme is evaluated using three datasets, yielding a small (<7%) overhead versus a non-SGX implementation.

2.3 Discussion

Modern TPM- and TEE-based approaches [4,15,23,32] still fall short of satisfying many desirable properties identified in past work. Public verifiability of origin, as in [4], has not been addressed in recent TEE loggers, which could be potentially useful to authenticate system data from remote devices, e.g. generating trust scores from log data for access control [2] and continuous authentication [22,29]. Recent TEE-based schemes, i.e. [15,23], focus primarily on protecting logs *after* being received by a server-side log processing application; an attacker on the source device may simply tamper the logs before reaching the server that applies some tamper-resistance algorithm. To complicate matters, source devices are unlikely to transmit logs in real-time to minimise network and computational overhead, and so secure storage methods should be used to preserve unsent logs. Additionally, TEEs typically contain other security-critical applications, e.g. for fingerprint matching (as in Android[1]) and payment tokenisation (see Samsung Pay[2]). As a result, a TEE-based logging mechanism should operate with reasonable resource consumption, e.g. run-time memory, to limit the rise of Denial of Service (DoS) conditions.

3 Trusted Execution Environments (TEEs)

GlobalPlatform defines a TEE as an isolated execution environment that *"protects from general software attacks, defines rigid safeguards as to the data and functions a program can access, and resists a set of defined threats"* [9]. TEEs aim to isolate applications from integrity and confidentiality attacks from a conventional operating system. Applications in the conventional OS and TEE – referred to as 'untrusted' and 'trusted' worlds respectively in GlobalPlatform

[1] https://source.android.com/security/authentication/fingerprint-hal.
[2] http://developer.samsung.com/tech-insights/pay/device-side-security.

nomenclature – reside in separate memory address spaces, and trusted hardware is used to monitor and prevent unauthorised memory accesses from the untrusted world. TEE applications may allocate shared memory spaces or expose predefined functions via an API mediated by a high-privilege secure monitor. Next, we summarise the leading commercial TEE architectures.

The **GlobalPlatform (GP) TEE** maintains two worlds for all trusted and untrusted applications. A TEE-based kernel is used for scheduling, memory management, cryptographic methods and other basic OS functions; TEE-resident Trusted Applications (TAs) may access OS functions exposed by the GP TEE Internal API (see Fig. 1). The GP TEE Client API [9] defines the interfaces for communicating with TAs from untrusted world applications. The GP specifications also cover the use of external secure elements (GP Secure Element API), secure storage, and networking (GP Sockets API) [11]. One method for instantiating the GP TEE is using ARM TrustZone, which enables two isolated worlds to co-exist in hardware. This is achieved using two virtual cores for each world per physical CPU core and an extra CPU bit, the NS bit, for distinguishing between untrusted/secure world execution modes. TrustZone provides secure I/O with peripheral devices connected over standard interfaces, e.g. SPI and GPIO, by routing interrupts to the TEE OS. This is performed via the TrustZone Protection Controller (TZPC), responsible for securing on-chip peripherals, and the TrustZone Address Space Controller (TZASC) for protecting memory-mapped devices from untrusted world accesses.

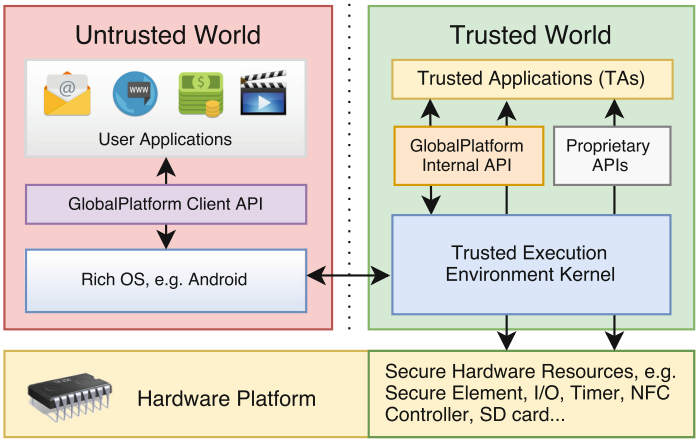

Fig. 1. GlobalPlatform TEE architecture.

Intel Software Guard Extensions (SGX) is an extension to the X86-64 instruction set that enables on-demand creation of 'enclaves' per application. Enclaves reside in isolated memory regions within RAM with accesses mediated by the CPU, which is considered trusted [8]. Enclaves may access the memory space of a regular OS, but not vice-versa, and enclaves cannot access other

enclaves arbitrarily. Like TPMs, SGX offers secure storage through 'sealing' in which data is encrypted and made accessible only to that enclave. Remote attestation enables third-party verification of enclaves and secret provisioning using Enhanced Privacy ID (EPID) – a Direct Anonymous Attestation (DAA) protocol by Brickell and Li [5]. SGX has been supported from the release of the Skylake microarchitecture (from 2015).

Despite some high-level similarities, SGX is not GlobalPlatform-compliant. Intel SGX is currently restricted solely to Intel CPUs, while the GP TEE is typically deployed on ARM System-on-Chips (SoCs) using TrustZone, as used by many IoT devices, e.g. Raspberry Pi 3[3], NEST thermostat[4], and 95% of consumer wearables according to ARM [1]. The reader is referred to [30] for a detailed survey of secure and trusted execution environments for IoT devices.

4 System Requirements

We formalise the requirements for a TEE-based system for protecting logs on constrained devices. The proposal should satisfy the following security and functional requirements drawn from the issues identified in Sect. 2:

R1. *Isolated execution*: the system shall process logs in an environment isolated from a regular 'rich' OS, e.g. Android, to provide strong integrity assurances of the application and data under execution.
R2. *Forward integrity*: the integrity of a given block of logs shall not be affected by a key comprise of a previous block.
R3. *Log confidentiality*: on-device log confidentiality should be preserved to prevent the disclosure of potentially sensitive entries.
R4. *Remote attestation*: the proposal shall allow third-parties to verify the logging application's integrity post-deployment to provide assurances that logs were sourced from an integral and authentic platform.
R5. *Secure log retrieval*: authorised third-parties shall be able to securely retrieve device logs without human intervention.
R6. *Public verifiability*: the system shall allow third-parties to authenticate the origin of log entries without access to private key information.
R7. *Truncation attack-resistant*: the system shall be resistant to attacks that aim to delete a contiguous subset of tail-end log entries.
R8. *Re-ordering attack-resistant*: the proposal shall resist attempts to change the order of entries in the log sequence.
R9. *Power-loss resilience*: the loss of tamper-resistant logs shall be minimised in the event of a device power-loss.
R10. *Suitable root of trust*: a root of trust for constrained device architectures shall be used, ideally without requiring additional security hardware.

[3] https://www.raspberrypi.org/products/.
[4] https://nest.com.

The threat model considers two adversary types:

- *On-device software adversary*: a software-based attacker that compromises the system at time t and attempts to arbitrarily alter, forge or observe logs produced before t. This may operate at any protection level in the untrusted world, i.e. Rings 0–3, including arbitrarily altering execution flow and accessing non-TEE kernel space services.
- *Network adversary*: an adversary that attempts to arbitrarily alter, forge, replay or observe logs between the source device and the verifier over a network channel, e.g. WiFi/802.11. The attacker may also attempt to masquerade as a legitimate party to either end-point to collect logs illicitly.

Like past work, we trust the TEE and do not attempt to secure untrusted world logs *after* a compromise after time t, since a kernel-mode adversary may simply read/write directly to the kernel message buffer used to queue log entries (see Sect. 5.1). We also consider hardware and related side-channel attacks, e.g. power analysis, beyond the scope of this work, as these threats fall outside the security remit of TEEs. The reader is referred to the GlobalPlatform TEE Protection Profile [9] for a specification of their protection scope.

5 EmLog Architecture Design

We assume the presence of a GlobalPlatform-compliant TEE, a service provider that provisions EmLog into the TEE before deployment, and a third-party wishing to retrieve all or a partial set of the device's logs. The GP TEE, which maintains two sets of applications for each world, necessitates two logging components: one that collects logs from untrusted world applications and transmits these to the TEE over the GP Client API, and another that applies the protection algorithm within the TEE and responds to retrieval requests. An extension of the hash matrix in [15,32] is proposed to apply the tamper-resistance scheme within the GP TEE, which achieves integrity protection and public verifiability (Sect. 5.2). Next, the log blocks are stored every n blocks, or at a time epoch t, using the secure storage functionality of the GP TEE. After receiving a retrieval request, the source TEE authenticates the remote verifier and vice-versa, after which the blocks are unsealed and transmitted over a secure channel between the TEEs (Sect. 5.3). We illustrate this process in Fig. 2.

5.1 Log Collection

A conventional (Linux) kernel uses an internal message ring buffer to store log messages, which is made available to user space monitoring applications, such as `dmesg` and `klogd`, using the `sys_syslog` syscall. For user-mode logging, `syslogd` listens on `/dev/log`, where logs are registered to using the `syslog` function from the C standard library. Logs are subsequently written to file or transmitted through the `syslog` protocol to a remote server over UDP. Some implementations, e.g. `syslog-ng`, provide further functionality like streaming logs over TCP with TLS. For collecting untrusted world logs, we suggest a `syslogd` variant that transmits logs to the EmLog TA within the GP TEE via the GP Client API.

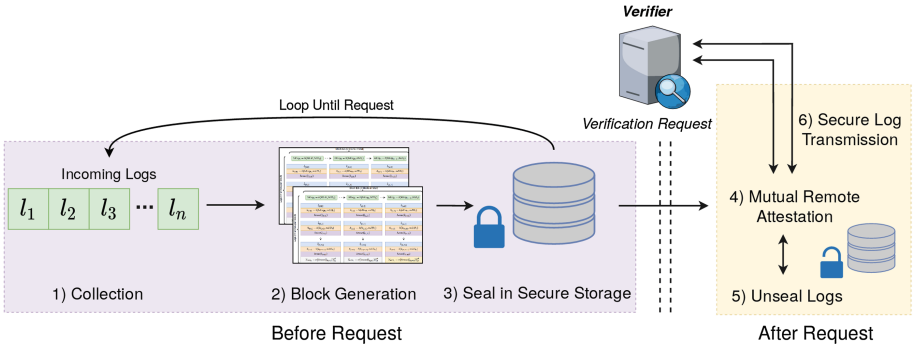

Fig. 2. High-level TEE-based logging workflow.

5.2 Block Generation

We propose a variant of the hash matrix used in [15,32] for log sequence integrity. Here, hash sequences are created in which each block key, bK, is derived using a one-way hash function, h, over the previous block key and current block ID, bID; that is, $bK_{bID} = h(bK_{bID-1}, bID)$. The initial block key ($bID = 0$) is derived from a device-specific Root Logging Key (RLK). Each block key is used to derive an individual message key, k, for keying an HMAC in a similarly chained fashion, i.e. $k_{(bID,mID)} = h(k_{\{bID,(mID-1)\}}, mID)$ for log entry mID in block bID, up to the block size m. Note that bK is used to derive k when $mID = 0$. A block-based approach provides power-loss resilience and truncation resistance (developed further in Sect. 5.3) while allowing the retrieval of subsets, i.e. blocks i to j, without transmitting all logs from the genesis block ($bID = 0$) to the remote verifier.

As it stands, this scheme is vulnerable to forgery attacks if just a single block key is compromised: an adversary can apply h on the leaked key with the next block ID to forge subsequent blocks and entries therein. Storing RLK and deriving keys within trusted hardware, e.g. an secure element (SE) or TPM, is desirable, but this adds hardware complexity to already-constrained devices with respect to raw component and integration costs. TEEs provide strong resilience to software attacks, but, unfortunately, are not invulnerable to developer-induced programming and API errors. The impact of RLK and block key divulgence, however, can be limited using key derivation, as described below.

Key Derivation. We suggest a simple scheme as follows: **(1)**, intermediate keys (IKs) are derived from the RLK using a secure key derivation function, each of which serves c blocks; **(2)**, each IK derives an initial block key, bK_{bID}, for that block group, before sealing the IK immediately to storage; **(3)**, bK_{bID} is used to generate the block's message-specific keys; **(4)**, the next bK_{bID} is derived using $kdf(bK_{bID-1}, bID)$ for up to c blocks, after which another IK is generated. In past proposals, a block key disclosure would require re-provisioning RLK – a device-specific, possibly hardware-infused key, which would affect the device in

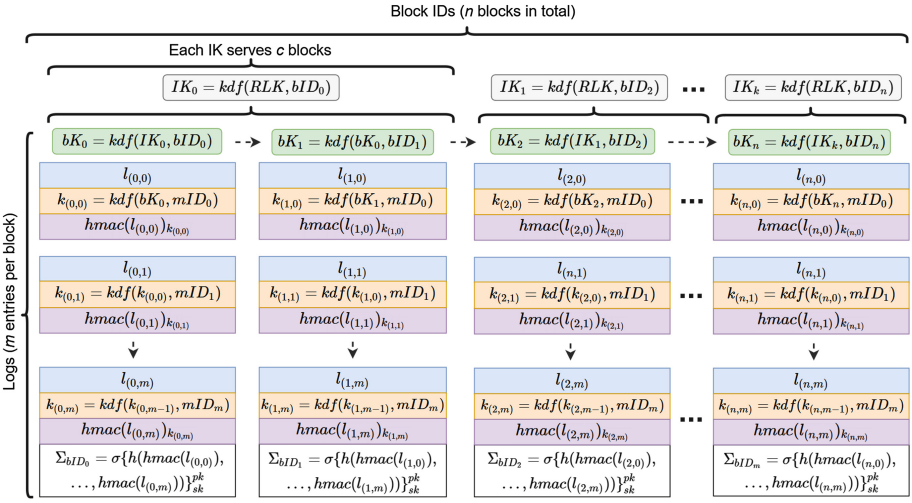

Fig. 3. Proposed two-dimensional, signature-based log structure.

perpetuity without potentially costly intervention. Our approach (Fig. 3) limits the damage wrought by a compromised block key by affecting only future blocks *in that group*. In the worst case, besides divulging RLK, the exposure of IK can compromise only c blocks at most.

For the key derivation function, we suggest the HMAC-based extract-and-expand KDF (HKDF) by Krawczyk [17,18] (RFC 5689). HKDF takes keying material and a non-secret salt as input, and repeatedly generates HMACs under the input to return cryptographically strong output key material. Unlike plain hash functions, used prevalently in past work, HKDF produces provably strong key material from as-strong or weaker input key material.

Log Integrity and Verifiability. For log message integrity, first compute $hmac(\ell_{(bID,mID)})$ for message ℓ with block ID, bID, and message ID, mID, under key $k_{(bID,mID)}$ – derived from the previous message key or, for $mID = 0$, the block key. Each k should be immediately deleted from memory to limit memory consumption and exposure. This does not prevent auditing log sequences, since message keys may be regenerated from the pre-shared RLK.

In current symmetric-only schemes [7,15,27,32], public verification of log origin (R6 in Sect. 4) requires knowledge of block and message keys on all interested devices, derived ultimately from RLK. Revealing RLK is evidently undesirable because it enables the malicious creation and manipulation of valid blocks. Rather, we propose signing each block with an efficient signature scheme, σ, such as ECDSA, and a device-specific signing key-pair (pk, sk) over the concatenation of the block message HMACs (Fig. 3). This key-pair should be certified to provide data origin authentication. The RLK and key-pair should be accessible only to the TEE, which is achievable using the TEE's secure storage mechanism or, for hardware tamper-resistance (with its complexities), using an external SE as

suggested by GlobalPlatform [9]. In some circumstances, logs may contain sensitive data, in which case we suggest limiting verifiability to whitelisted entities, e.g. devices from the same manufacturer or service provider. It is also observed that the block size, m, is inversely proportional to the number of signing operations; smaller block sizes will incur more signing operations for a given set of log entries (see Sect. 7 for this overhead).

5.3 Secure Storage and Remote Retrieval

Real-time log streaming is likely to be detrimental for power- and network-limited devices, and we suggest storing blocks prior to eventual transmission within the TEE's secure storage. Secure storage can be implemented in two ways according to GlobalPlatform: **(1)**, using the file system and storage medium, e.g. flash drive, controlled by the untrusted world *"as long as suitable cryptographic protection is applied, which must be as strong as that used to protect the TEE code and data itself"* [10]. Or **(2)**, using hardware controlled only by the TEE, e.g. an external SE. Method **(2)** is resilient against adversaries that aim to delete encrypted records from the file system[5], but naturally requires additional security hardware. For method **(1)**, log blocks are sealed using authenticated encryption (AES in GCM mode) with a key derived specifically for the TA under execution from a separate, device-specific root storage key. This prevents other TAs or other entities from accessing secured data, thus providing on-device log confidentiality (R3), integrity and authenticity.

Securely storing every completed block, i.e. in [15], may yield undesirable performance overhead for the devices targeted in this work. Rather, the parameters c (block group size) and m (block length) can control the number and size of blocks kept in RAM respectively. This satisfies truncation attack-resistance (R7) and power-loss resilience (R9), in addition to R3, by limiting the number of new blocks kept in memory (for sufficiently small values of c and m).

In past work, log retrieval is proposed using TLS [23], or one-way remote attestation for authenticating the platform of the remote verifier [15]. (Many remote attestation protocols, e.g. [5], typically enable secure channels to be bootstrapped, over which unsealed logs can be transmitted securely). However, the remote authority, which may itself process logs in its own TEE [15,23], is likely to request reciprocal trust assurances from the source TEE, i.e. remote attestation for both the source *and* verifying entities. Rather than performing one-way attestation separately for both entities, one alternative is mutual TEE attestation [28] in which both communicating TEEs are attested and authenticated within the protocol run. Similarly, a secure channel can be bootstrapped from [28] between the TEE end-points over which unsealed logs can be transmitted securely without exposing them to untrusted world elements.

[5] Note that, in general, arbitrary log deletion is difficult to prevent robustly without dedicated WORM (Write-Once, Read-Many) storage.

6 Implementation

We implemented EmLog using OP-TEE – an open-souce, GlobalPlatform compliant TEE by Linaro [19] – with Debian (Linux) as the untrusted world OS. An untrusted world application was developed for collecting log entries from file, which were sent subsequently to the EmLog TA using the GP Client API [9]. Each entry was processed into a data structure comprising a 4-byte message ID, 32-byte HMAC (SHA-256) tag, and 256-byte field for the entry text. The GP Internal API [10] was used to interface with the cryptographic and secure storage methods; in OP-TEE, cryptographic methods are implemented using the `LibTomCrypt` library, and we opted for secure storage in which data is encrypted to the untrusted world file system (residing on 32 GB eMMC flash memory). 256-bit ECDSA (NIST `secp256r1` curve) was used to sign each block, which was placed into a separate data structure comprising the processed messages and a 4-byte block ID; currently, only the NIST curves are defined in the GP TEE specifications. The GP Internal API defines its own memory allocation functions, i.e. `TEE_Malloc` and `TEE_Free`, for dynamically (de-)allocating memory to regions accessible only to the TA, which were used frequently for memory-managing blocks and messages at run-time.

Unsurprisingly, memory consumption quickly became problematic when working with large datasets (discussed in Sect. 7). For the current OP-TEE release, 32 MB RAM is allocated for the TEE kernel and all resident TAs, with the rest allocated to the untrusted world OS. For a standard TA, the Linaro Working Group[6] stipulates a default stack and heap size at 1 kB (stack) and 32 kB (data) respectively, both of which can be increased up to a maximum 1 MB per TA.

7 Evaluation

EmLog was evaluated using a HiKey LeMaker – an ARM development board with a Huawei HiSilicon Kirin 620 SoC with 2 GB RAM and an ARM Cortex A53 CPU (eight-cores at 1.2 GHz with TrustZone extensions). Such specifications are typical of modern medium-to-high end IoT-type systems, such as the Raspberry Pi 3 and Nest Thermostat. The proposal was benchmarked using three log file datasets, described briefly:

1. *U.S. Securities and Exchange Commission (SEC) EDGAR*: Apache logs from access statistics to SEC.gov. We use the latest dataset[7] with over a million entries (192 MB). (Mean entry length: 115.08 characters; S.D.: 5.73).

[6] https://wiki.linaro.org/WorkingGroups/Security/OP-TEE.
[7] http://www.sec.gov/dera/data/Public-EDGAR-log-file-data/2016/Qtr2/log20160630.zip.

2. *Mid-Atlantic Collegiate Cyber Defense Competition (CDC)*: IDS logs from the U.S. National CyberWatch MACCDC event, with ∼166,000 (27 MB) of Snort fast alert logs[8]. (Mean entry length: 165.27 characters; S.D.: 38.21).
3. ***EmLogs***: Our dataset from OP-TEE OS boot, initialisation and GlobalPlatform test suite logs via the `xtest` command, and untrusted world logs from `dmesg`. Over 25,000 records (1.7 MB). (Mean entry length: 94.14; S.D.: 49.33).

The results are shown in Tables 1, 2, 3 and 4 and Fig. 4. Table 1 shows the mean CPU time to derive 256-bit IKs, block and message keys from a pre-generated RLK using HKDF. These were measured over 1,000 iterations within the EmLog TA using the GlobalPlatform `TEE_Time` method for system time, implemented using the ARM Cortex `CNTFRQ` (CPU frequency) and `CNTPCT` (count) timing registers. Table 1 shows the mean time for sealing and unsealing IKs and blocks (for $m = 100$, averaged across all entries) via the GP Internal API. Table 2 lists the mean 256-bit ECDSA and HMAC-SHA256 times computed across all entries, while Table 3 shows the mean creation and verification times of message blocks for each dataset (for varying values of m, the number of entries per block), as well as block groups. In this context, verification encompasses the time to reconstruct the hash matrix in Fig. 3 and to verify the block signatures and message HMACs. Group creation and verification time was measured for varying values of c (blocks per group), which included the time for sealing and unsealing blocks to secure storage respectively. Table 4 shows the mean persistent memory consumption of logs in secure storage, which was measured directly from `/data/tee` in the untrusted world file system, where OP-TEE stores sealed TA files. Lastly, Fig. 4 shows the relative performance of secure storage, key derivation and block and group creation/verification times from Table 1.

Table 1. Mean key derivation and secure storage times (milliseconds; S.D. in brackets).

Key derivation			Secure storage seal		Secure storage unseal	
IK	Block Key	Message Key	IK	Block	IK	Block
1.530 (0.067)	1.541 (0.062)	1.547 (0.088)	59.46 (3.78)	115.8 (5.36)	48.22 (2.73)	94.88 (2.80)

Table 2. Mean HMAC and ECDSA generation and verification times (milliseconds).

	HMAC (SHA-256)	ECDSA (NIST P256)
Create	0.056 (0.020)	20.14 (1.29)
Verify	0.059 (0.014)	20.77 (1.33)

[8] http://www.secrepo.com/maccdc2012/maccdc2012_fast_alert.7z.

Table 3. Mean block and group generation and verification times (milliseconds).

Dataset		Block				Group (fixed at $m = 100$)			
		$(m =)10$	100	250	500	$(c =)1$	10	25	50*
(1)	**Create**	40.14 (2.1)	70.45 (2.2)	115.6 (2.6)	198.7 (3.6)	229.4 (6.4)	1898 (30.2)	4622 (48.1)	9101 (80.2)
	Verify	41.18 (1.5)	71.88 (1.7)	114.8 (1.9)	204.3 (2.1)	213.4 (5.7)	1634 (23.7)	3967 (50.0)	8302 (78.3)
(2)	**Create**	39.84 (1.9)	68.34 (1.7)	117.2 (1.7)	202.3 (2.0)	231.7 (7.3)	1910 (28.1)	4687 (64.0)	9323 (81.5)
	Verify	40.05 (1.6)	66.15 (1.8)	119.9 (1.8)	199.0 (2.0)	215.2 (6.1)	1658 (23.8)	4046 (47.3)	8165 (73.9)
(3)	**Create**	42.14 (3.0)	69.07 (2.1)	118.6 (2.3)	201.9 (3.2)	230.0 (6.9)	1890 (27.0)	4621 (42.8)	9274 (79.0)
	Verify	40.01 (1.6)	69.28 (1.8)	120.4 (1.8)	200.4 (1.9)	217.3 (6.4)	1656 (25.2)	4132 (48.1)	8188 (75.5)

* Heap size set to 2 MB to accommodate all data. All other experiments recorded with the maximum recommendation of 1 MB.

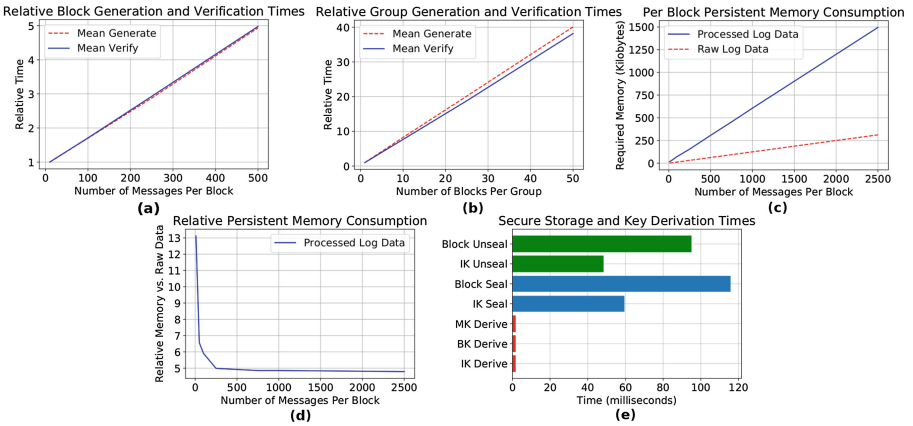

Fig. 4. **(a)**, Relative block creation and verification times versus block length; **(b)**, relative group generation and verification for varying numbers of blocks; **(c)**, persistent memory consumption for per block secure storage; **(d)**, relative memory consumption for group secure storage; and **(e)**, raw key derivation and secure storage times.

Table 4. Persistent memory consumption for per block secure storage (kilobytes).

Block Sizes							
$(m =)10$	50	100	250	500	750	1000	2500
16.38	40.96	73.73	155.65	307.20	454.66	606.21	1495.04

7.1 Discussion

Little to our surprise, block generation and verification time scales linearly with message length, which, for large values of m, is influenced heavily by the key derivation operations (approximately 1.5 ms per message, shown in Table 1).

At smaller values, e.g. $m = 10$, this is dominated mostly by the ECDSA overhead (∼20 ms, as per Table 2). Figure 4 indicates that the relative timing overhead is ∼80–100% for every 100 message increase in the block length.

Group creation and verification times rise significantly with the number of blocks, c, kept in RAM before secure storage. This is driven significantly by the secure storage overhead, which is measured at approximately 115.8 ms and 94.88 ms for sealing and unsealing respectively (Table 1). Despite this, however, even the largest group sizes, $c = 25$ and $c = 50$ (2,500 and 5,000 entries in total), completed between 4.0 to 9.3 s, corresponding to a throughput of approximately 538 and 625 logs per second. At first, it seems attractive to maximise c to avoid the expense of secure storage operations, which caused the throughput to drop to ∼430 and 525 entries for the smallest groups ($c = 1$ and $c = 10$ blocks). Maintaining many blocks in RAM, however, increases the impact of a power-loss; systems that log infrequently may see significant data loss if large numbers of logs spread over a large period of time are lost. Consequently, c should be set based on the expected log and transmission frequencies.

For memory consumption, all experiments were conducted within the Linaro Working Group's run-time recommendations (1 MB stack and heap), except for $c = 50$ blocks (5,000 entries), which required 2 MB of each. Expectedly, persistent memory consumption of block secure storage (Fig. 4) scales linearly with message size. Our test-bed uses a fixed 256-byte text field for each log entry, which accounts for the broadly similar performance across all datasets. We also calculated the persistent memory consumption compared with the mean size of raw logs; the relative consumption is large for small block sizes ($m < 250$), due likely to the fixed-size meta-data used by OP-TEE to manage cross-TA secure storage objects. For larger block sizes, this converges to slightly under five-times overhead versus raw logs; the absolute size of smaller block sizes remains low, however, at 16–155 kilobytes, according to Table 4.

7.2 Requirements Comparison

We compare the features of EmLog's with previous work in Table 5 using the requirements in Sect. 4. Notably, the use of ARM TrustZone and the GlobalPlatform TEE makes it appropriate for mobile and embedded devices targeted in this work (R10), unlike SGX-based schemes, which are restricted to Intel CPUs associated with laptop, desktop and server machines. EmLog satisfies the features of related cryptographic and trust-based proposals, such as resistance to truncation (R7) and re-ordering (R8) attacks, and public verifiability of log origin (R6). We also offer forward integrity protection for compromised block keys (R2) using more sophisticated key derivation, thus moving the cost-reward ratio further away from an attacker. By avoiding TPMs, however, we relinquish strong hardware tamper-resistance, and we urge caution of our work in high-security domains, e.g. military and governmental use, where complex hardware and side-channel attacks are reasonable threats.

Table 5. Security requirements comparison of related work.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	Root of trust
Untrusted world schemes											
Schneier and Kelsey [27]	–	✗	✓	–	–	✗	✗	✗	–	–	–
Bellare and Yee [3]	–	✓	✓	–	–	✗	✗	✗	–	–	–
FssAgg [20]	–	✓	✗	–	–	✓	✓	✓	–	–	–
Logcrypt [13]	–	✓	✓	–	–	✓	✗	✗	–	–	–
LogFAS [34]	–	✓	✗	–	–	✓	✓	✓	–	–	–
Trusted logging											
Chong et al. [7]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	Java iButton
Sinha et al. [32]	✗	¶	✓	✓	✗	✗	✓	✓	✓	✗	TPM
Böck et al. [4]	✓	✗	¶	✓	✗	✓	✗	✗	✗	✗	TPM & AMD SVM
Nguyen et al. [23]	✓	✗	¶	✓	¶	✗	✓	✓	✓	✗	Intel SGX
SGX-Log [15]	✓	¶	✓	✓	¶	✗	✓	✓	✓	✗	Intel SGX
EmLog	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	GlobalPlatform TEE

✓– Satisfies requirement; ✗– Does not satisfy; ¶– Partially satisfies; (–) – N/A.

8 Conclusion

In this paper, we introduced EmLog – a tamper-resistant logging scheme for modern constrained device using the GlobalPlatform TEE and ARM TrustZone. We began with a two-part review of related work in Sect. 2 by summarising cryptographic proposals and those reliant upon trusted hardware for tamper-resistant logging. Next, the features of TEEs were assess in further detail in Sect. 3, before formulating the requirements and the threat model in Sect. 4 using past work. After this, we introduced the architectural design and proposed an improved log preservation algorithm for providing public verifiability of log origin and key exposure resilience. We described the implementation of EmLog in Sect. 6 and presented indicative performance results using diverse datasets in Sect. 7. For the first time, our work brings secure, TEE-based logging to mobile and embedded devices, and protects against strong software-based untrusted world and network adversaries. Our evaluation shows that EmLog yields five-times persistent storage overhead versus raw logs for applying tamper-resistance; runs within reasonable run-time memory constraints for TEE applications, as stipulated by the Linaro Working Group; and has a throughput of up to 625 logs/sec. In future work, we aim to investigate the following avenues:

- *Group logging schemes for multiple devices.* Expand EmLog to allow secure and efficient sharing of logs with nearby devices. This could be used in schemes that compute trust scores prior to making group decisions [2], e.g. authenticating users via contextual data from multiple wearable devices.
- *Privacy-preserving log usage.* At present, devices that wish to use logs will receive raw logs, which may reveal privacy-sensitive data. In future work, we

aim to explore privacy-preserving methods for using logs without exposing raw entries to other devices.
– *TEE performance comparison.* We hope to evaluate EmLog under other TEE instantiations, namely Intel SGX and other GP-compliant TEEs, such as TrustTonic's Kinibi, especially for micro-controllers on low-end IoT devices.

Acknowledgements. Carlton Shepherd is supported by the EPSRC and the British government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. ARM: Markets: Wearables (2017). https://www.arm.com/markets/wearables
2. Bao, F., Chen, I.-R.: Dynamic trust management for Internet of Things applications. In: International Workshop on Self-aware Internet of Things, pp. 1–6. ACM (2012)
3. Bellare, M., Yee, B.: Forward integrity for secure audit logs. Technical report, Computer Science and Engineering Department, University of California at San Diego (1997)
4. Böck, B., Huemer, D., Tjoa, A.M.: Towards more trustable log files for digital forensics by means of trusted computing. In: 24th International Conference on Advanced Information Networking and Applications, pp. 1020–1027. IEEE (2010)
5. Brickell, E., Li, J.: Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. Int. J. Inf. Privacy Secur. Integrity **1**(1), 3–33 (2011)
6. Chen, D., Wang, M.: A home security ZigBee network for remote monitoring applications. In: International Conference on Wireless, Mobile and Multimedia Networks, pp. 1–4. IET (2006)
7. Chong, C.N., Peng, Z., Hartel, P.H.: Secure audit logging with tamper-resistant hardware. In: Gritzalis, D., De Capitani di Vimercati, S., Samarati, P., Katsikas, S. (eds.) SEC 2003. ITIFIP, vol. 122, pp. 73–84. Springer, Boston, MA (2003). https://doi.org/10.1007/978-0-387-35691-4_7
8. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive, 2016:86 (2016). https://eprint.iacr.org/2016/086.pdf
9. GlobalPlatform: TEE Protection Profile (v1.2) (2014)
10. GlobalPlatform: TEE Internal Core API (v1.1.1) (2016)
11. GlobalPlatform: TEE System Architecture (v1.1) (2017)
12. Hartung, G.: Attacks on secure logging schemes. IACR Cryptology ePrint Archive, 2017:95 (2017). https://eprint.iacr.org/2017/095.pdf
13. Holt, J.E.: Logcrypt: forward security and public verification for secure audit logs. In: Proceedings of the 2006 Australasian Workshops on Grid Computing and E-research, pp. 203–211. Australian Computer Society Inc. (2006)
14. International Standards Organisation: ISO/IEC 27001:20133 - Information Technology, Security Techniques, Information Security Management Systems, Requirements (2013). https://www.iso.org/standard/54534.html
15. Karande, V., Bauman, E., Lin, Z., Khan, L.: SGX-log: securing system logs With SGX. In: Proceedings of the 2017 Asia Conference on Computer and Communications Security, ASIA CCS 2017, NY, USA, pp. 19–30. ACM (2017)

16. Kent, K., Souppaya, M.: Guide to computer security log management. NIST Spec. Publ. 800-92 (2006)
17. Krawczyk, H.: Cryptographic extraction and key derivation: the HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_34
18. Krawczyk, H., Eronen, P.: RFC 5869 - HMAC-based Extract-and-expand Key Derivation Function (HKDF), May 2010. https://tools.ietf.org/html/rfc5869
19. Linaro: OP-TEE: Open Portable Trusted Execution Environment (2017). https://www.op-tee.org/
20. Ma, D., Tsudik, G.: A new approach to secure logging. ACM Trans. Storage **5**(1), 2 (2009)
21. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: efficient TCB reduction and attestation. In: 2010 IEEE Symposium on Security and Privacy, pp. 143–158. IEEE (2010)
22. Micallef, N., Kayacık, H.G., Just, M., Baillie, L., Aspinall, D.: Sensor use and usefulness: trade-offs for data-driven authentication on mobile devices. In: IEEE International Conference on Pervasive Computing and Communications, pp. 189–197. IEEE (2015)
23. Nguyen, H., Acharya, B., Ivanov, R., Haeberlen, A., Phan, L.T.X., Sokolsky, O., Walker, J., Weimer, J., Hanson, W., Lee, I.: Cloud-based secure logger for medical devices. In: IEEE 1st International Conference on Connected Health: Applications, Systems and Engineering Technologies, pp. 89–94, June 2016
24. Patel, S., Park, H., Bonato, P., Chan, L., Rodgers, M.: A review of wearable sensors and systems with applications in rehabilitation. J. Neuro-Eng. Rehabil. **9**(1), 21 (2012)
25. Perez, R., Sailer, R., van Doorn, L., et al.: vTPM: virtualizing the trusted platform module. In: Proceedings of the 15th USENIX Security Symposium, pp. 305–320 (2006)
26. Rashidi, P., Mihailidis, A.: A survey on ambient-assisted living tools for older adults. IEEE J. Biomed. Health Inform. **17**(3), 579–590 (2013)
27. Schneier, B., Kelsey, J.: Secure audit logs to support computer forensics. ACM Trans. Inf. Syst. Secur. (TISSEC) **2**(2), 159–176 (1999)
28. Shepherd, C., Akram, R.N., Markantonakis, K.: Establishing mutually trusted channels for remote sensing devices with trusted execution environments. In: 12th International Conference on Availability, Reliability and Security (ARES), pp. 7:1–7:10. ACM (2017)
29. Shepherd, C., Akram, R.N., Markantonakis, K.: Towards trusted execution of multi-modal continuous authentication schemes. In: Proceedings of the 32nd Symposium on Applied Computing, pp. 1444–1451. ACM (2017)
30. Shepherd, C., Arfaoui, G., Gurulian, I., Lee, R.P., Markantonakis, K., Akram, R.N., Sauveron, D., Conchon, E.: Secure and trusted execution: past, present, and future - a critical review in the context of the Internet of Things and cyber-physical systems. In: 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 168–177 (2016)
31. Singaravelu, L., Pu, C., Härtig, H., Helmuth, C.: Reducing TCB complexity for security-sensitive applications: three case studies. In: ACM SIGOPS Operating Systems Review, vol. 40, pp. 161–174. ACM (2006)
32. Sinha, A., Jia, L., England, P., Lorch, J.R.: Continuous tamper-proof logging using TPM 2.0. In: Holz, T., Ioannidis, S. (eds.) Trust 2014. LNCS, vol. 8564, pp. 19–36. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08593-7_2

33. Trustonic: Adoption of Trustonic Security Platforms Passes 1 Billion Device Milestone, February 2017. https://www.trustonic.com/news/company/adoption-trustonic-security-platforms-passes-1-billion-device-milestone/
34. Yavuz, A.A., Ning, P., Reiter, M.K.: Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 148–163. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_12

How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip

Sebanjila Kevin Bukasa[1,2(✉)], Ronan Lashermes[1,2], Hélène Le Bouder[3], Jean-Louis Lanet[1,2], and Axel Legay[2]

[1] LHS-PEC, Rennes, France
[2] TAMIS INRIA, Rennes, France
{sebanjila.bukasa,ronan.lashermes,helene.lebouder,
jean-louis.lanet,axel.legay}@inria.fr
[3] IMT Atlantique, Rennes, France
helene.lebouder@imt-atlantique.fr

Abstract. Side-channel attacks (SCA) exploit the reification of a computation through its physical dimensions (current consumption, EM emission, ...). Focusing on Electromagnetic analyses (EMA), such analyses have mostly been considered on low-end devices: smartcards and microcontrollers. In the wake of recent works, we propose to analyze the effects of a modern microarchitecture on the efficiency of EMA (here Correlation Power Analysis and template attacks). We show that despite the difficulty to synchronize the measurements, the speed of the targeted core and the activity of other cores on the same chip can still be accommodated. Finally, we confirm that enabling the secure mode of TrustZone (a hardware-assisted software countermeasure) has no effect whatsoever on the EMA efficiency. Therefore, critical applications in TrustZone are not more secure than in the normal world with respect to EMA, in accordance with the fact that it is not a countermeasure against physical attacks. For the best of our knowledge this is the first application of EMA against TrustZone.

Keywords: ARM TrustZone · Side-Channel Analysis (SCA)
Raspberry Pi 2

1 Introduction

Embedded devices have never been so pervasive in our everyday life. Smartphones are already widespread and will continue to be present all around the world. Internet-of-Things (IoT) devices are poised to become ubiquitous. In parallel, more diverse and more critical applications are accessed with these devices: credit cards or other payment methods such as Apple Pay [5,23], password managers [10] or two factors authentication (2FA) are some examples.

Considering the information security, these devices offer a new set of possible threats: the attacker can have physical access to the device. Think of a stolen

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 93–109, 2018.
https://doi.org/10.1007/978-3-319-93524-9_6

smartphone or a set-top box allowing access to pay-TV whose user would prefer the TV without having to pay.

These problems have already been faced by the smartcard industry in the past [16].

Yet apart from embedded secure elements (eSE, which are basically smartcard IPs in a chip), modern System-on-Chip (SoC) fail to implement countermeasures against physical attacks. The reasons may be the highly competitive market for these chips that do not play well with the prudent pace needed to obtain secure devices. The fact that the sensitivity of these modern SoCs to physical attacks is not well known does not support a much needed consideration for these attacks.

Modern chips are increasingly complex, including several peripherals, working at high clock speed, including several memory levels, numerous cores, and security extensions such as TrustZone (defined in Sect. 2), the influence of all this activity on a SoC is of particular interest when it comes to physical attacks. This is what we want to verify in this paper by experimentally setup physical attacks.

Motivations: Our main objective is to expose the risks faced by users using embedded devices for critical operations when they are not protected against physical attacks. Most physical attacks are performed on low-end devices (microcontrollers and smartcards) and we investigate if they can be transposed easily to high-end devices. We verify that the TrustZone implementation does not offer any protection against side-channels leakages (TrustZone is only advertised as a software security solution by ARM, not hardware), and even multicore computations has no influence on these leakages.

Contribution: Two different attacks have been experimentally realized on a device representing a modern smartphone. We will use two use-cases for that purpose: the secret-key cryptography key recovery, and a Personal Identification Number (PIN) recovery.

These attacks have been realized while monitoring the impact of different features of the device.

Organization of this paper: The paper is organized as follows. Trusted Environment Execution and TrustZone specific implementation are presented in Sect. 2. In Sect. 3, Side-channel attacks (SCA) are presented and our two use cases are detailed. Our setup and experimental results are given in Sect. 4. Finally, the conclusion is drawn in Sect. 5.

2 Trusted Execution Environment and TrustZone

2.1 Trusted Execution Environment

Trusted Execution Environment Protection Profile (TEE PP) is a profile specified by GlobalPlatform in 2014 [1]. This TEE PP has been made specifically

for mobile devices, the goal was to deal with new threats in the mobile world and to allow customers to have confidence in the TEE in order to deliver mobile wallets, or secure mobility solutions.

A TEE is an execution environment that respect the TEE PP, meaning that it can be trusted by users. A Trusted Application (TA) is an application running in the TEE, it provides security related functions, and its assets include code, program data, cryptographic keys, etc. A TEE is designed to provide isolated execution, integrity of TA but also integrity and confidentiality of TA assets.

In order to provide a secure environment, a TEE might be working alongside a Rich Execution Environment (REE). A TEE is in fact another execution environment with its own Operating System (OS), the Trusted OS (TOS) and its own applications, TAs, executing over it. The OSes are isolated and the Rich OS (ROS) has just one entry point to the TOS.

There are several implementations of TEEs. The most notable are ARM TrustZone, detailed in this paper and Intel SGX [11].

2.2 TrustZone

TrustZone (TZ) is a specific implementation of a TEE proposed by ARM. This proposition is based on a few hardware-specific parts only. Security functions are mainly performed by software implementations or by optional pieces of hardware that can be added by each integrator. It respects GlobalPlatform specifications, and goes into TEE PP.

TrustZone hardware architecture leads to specific software implementations, a ROS, a TOS and the entry point (managing accesses to the secure world).

It consists in separating hardware resources into two worlds, Secure and Normal (Non-Secure) worlds. This is achieved by adding a security-related bit on the system bus (and therefore all bus accesses are impacted), and by adding new privilege levels. In that way each peripheral, each data or code section are separated between the two worlds and a core can only handle them if it is in the right privilege mode.

Typically, a non-secure ARMv7 core has three privilege modes:

- PL0 or user mode is the least privileged and is the mode where user application are executed in.
- PL1 or kernel mode is an intermediary privileged level and is typically where the kernel (of the ROS) is executed.
- PL2 or hypervisor mode is the highest level and is where the hypervisor (if any) is executed.

Now the TZ adds new modes. In addition to the previously defined non-secure PL0, PL1 and PL2, the new modes are:

- Secure PL0, the secure user mode is where trusted applications are executed.
- Secure PL1, the secure kernel mode is the mode for the secure kernel execution for the TOS.

– Monitor mode is a new mode that must be used to transition between the non-secure and the secure worlds.

When the core is in a secure state, the *non-secure (NS)* bit on the bus is set to 0. As a consequence, the systems on the bus can adapt to the security state (e.g. a peripheral may forbid the access when *NS* is 0).

3 Side Channel Attacks

3.1 Definition

When discussing about the security of an algorithm, numerous mathematical tools allow developers to prove its security. Unfortunately those tools cannot consider the interaction of the computing unit with its physical environment. Physical attacks are a real threat, even for algorithms proved secure mathematically. Side-channel analyses (SCA) are physical attacks based on the observation of the circuit behavior during a computation. They exploit the fact that some physical quantities depend on intermediary values of the computation in the device. This is the so-called leakage of information. The most classic leakages are timing [15,17], power consumption [21] and electromagnetic emissions (EM) [27]. In this paper we will focus on Electromagnetic analyses (EMA).

Side-channel analyses can be split in three families.

– The Simple Power Analysis (SPA) [20] uses directly the leakage to obtain the secret information.
– More complex attacks that need a mathematical model for the leakage and a statistic tool called a distinguisher. These attacks have generally a divide-and-conquer approach. The first ones were the Differential Power Analysis (DPA) [26] and Correlation Power Analysis (CPA) [7]. That is why this family of Side-Channel Attack (SCA) is often called the "DPA/CPA attacks". The differences between attacks of this family generally come from the choice of the model and the distinguisher.
– Template attacks are based on a statistical classification which replaces the mathematical model for the leakage in a DPA. To implement a template attack, a pair of identical devices is needed. One is called the profiling device, the attacker has full control of it, and is used to learn the leakage characteristics. The other is the targeted device on which the attack is carried. Template attacks were introduced as the strongest possible side-channel attacks from an information theoretic point of view [8]. They have been used in many attacks, such as [3,6] since.

SCA are threats for all standard cryptosystems as Data Encryption Standard (DES) [7,26], Advanced Encryption Standard (AES) [14,20], RSA cryptosystem [12], Elliptic Curve Cryptography (ECC) [25] and for critical applications not using cryptography, e.g. PIN verification [6]. SCA can also be used to reverse engineer algorithms [9].

3.2 Previous Works

Historically, the targeted devices were mostly smartcards, microcontrollers or cryptographic coprocessors [16] since they were the only embedded devices with critical information on it (for credit cards and pay-TV notably). These devices are relatively slow (under 10 MHz) and have simple architectures: a simple cache architecture if any, shallow pipeline, ... Features that make SCA easier. As a consequence, the smartcard industry developed strong countermeasures against physical attacks.

However, today the same applications are more and more executed on smartphones, and they do not build upon the knowledge from the smartcard industry in terms of SCA. The chips used in these devices are full featured processors running at high speed (1GHz+, >100X faster than a smartcard), with complex architectures (multiple cache levels, deep pipelines). SCA are therefore more complex. Measurement equipments have to be adapted to accept much higher frequencies. Jitter (temporal variance) is more important due to the caches, making synchronization harder. Yet SCAs on smartphones have already been performed. In [2], the authors have developed an EMA on an Android phone (running at 400 MHz). They targeted the secret key of an AES implementation in the Android BouncyCastle library. They show a key recovery in 250 traces with their special techniques designed to take into account the desynchronization added by the java virtual machine behavior.

How to do EMA on fast modern systems has been described in [4,19]. They both target a Beaglebone Black board running at 1 GHz. They had to filter out traces that where not properly synchronized due to interrupts and other operating system operations. Resynchronization and filtering are necessary to obtain clean traces. In [4], the authors attacked their own optimized AES implementation and show that their countermeasures are effective. In [19], the authors compared the leakage of an AES on the ARM core with respect to the NEON coprocessor.

From these papers, we can see that transposing EMA on high end processors requires a lot more experimental work. In particular, synchronization can be a problem because of a virtual machine (in [2]) or because of the operating system (in [4,19]). That in why in our case, we work in a bare-metal environment (i.e. without an operating system) to focus on the microarchitecture influence on the efficiency of EMA.

A second category of EMA on smartphone has been performed targeting public-key cryptography (RSA [12,29] or ECC [13]). Synthetically, these attacks have the same leakage principle. Public-key cryptography deals with big numbers (3072-bit for RSA, 256-bit for ECC at the 128-bit security level). Handling these big numbers in software requires to split them in words whose size is determined by the chip architecture (32-bit or 64-bit in most architectures).

As an example, a multiplication of two big numbers requires looping over the words constituting the operands. The sequence of operation in each iteration has a leakage signature that can be identified in the frequency domain (since the iteration is repeated several times). Interestingly, the leakage in the frequency

domain occurs at a relatively low frequency compared to the clock frequency (it corresponds to several instructions). Thereby, the attacker can identify the leakage signature of a big numbers multiplication versus squaring (for RSA) or point addition versus point doubling (for ECC). From this leakage, the authors devised methods to recover the secret.

Timing attacks are possible on high-end devices, in particular [18,30] highlighted that cache timing attacks can be performed in order to retrieve some informations on mobile devices, even from TrustZone.

3.3 Use Cases

To test the possibility to attack a modern SoC with EMA, two representative use-cases have been chosen: a CPA on the AES and a template attack on a Verify PIN algorithm.

CPA on AES. The first use-case is the well established CPA [7] applied to the AES.

Targeted Algorithm. The Advanced Encryption Standard is a standard established by the NIST [24] for symmetric key cryptography. It is a block-cipher, the encryption first consists in mapping the plaintext T of 128 bits into a two-dimensional array of $4 \cdot 4 = 16$ bytes called the State (4 rows and 4 columns). Then, after a preliminary xor (noted $\oplus$) between the input and the key K_0, the AES executes 10 times a round-function that operates on the State. The operations used during these rounds are:

- **SubBytes**, composed by non-linear transformations: 16 S-boxes noted SB, working independently on individual bytes of the State. In the targeted implementation, the S-boxes use a lookup table.
- **ShiftRows** noted SR, a byte-shifting operation on each row of the State.
- **MixColumns** noted MC, a linear matrix multiplication on $GF(2^8)$, working on each column of the State.
- **AddRoundKey** a xor between the State and the round-key K_r, $r \in [\![0, 10]\!]$.

The Attack. In this attack the target is the key of the first round K_0. To have a divide-and-conquer approach, K_0 is attacked byte per byte. There are only 256 possible values for each byte. The leakage is the EM emission measured at the S-boxes output. It depends on known plaintexts T and the secret key K_0.

So for each text T, and each guess k in $[\![0, 255]\!]$ a prediction P can be computed.

$$P(T, k) = SB(T \oplus k)$$

In particular it can be noted that the leakage is "value-based" and not, as usually the case, the Hamming Weight (HW) of the value (no leakage is observed in the HW model experimentally).

These predictions are confronted to the traces collected with a Pearson correlation that shows a linear relation between measurements and predictions.

Template Attack on a Verify PIN Algorithm. The attack summarized here is the attack of Bouder *et al.* presented in [6].

The Target: Verify PIN Algorithm. In many devices, a Personal Identification Number (PIN) is used to authenticate the user. The main protection in the use of a PIN comes from the fact that the user has a limited number of trials. A PIN is an array of m digits. A True PIN is embedded in the device; and a Candidate PIN is proposed by the user. The different values taken by the PIN are defined in $[\![0, 9]\!]^m$. The Verify PIN is the algorithm which tests if the Candidate PIN is correct or not. A good countermeasure against fault attacks is to compare the correct and Candidate PIN twice. The algorithm for the PIN comparison in constant time, from [28], is shown in Algorithm 1.

Algorithm 1. Comparison between candidate PIN and true PIN

```
1: procedure COMPARISON(candidate PIN CP, true PIN TP)
2:     status = FALSE
3:     diff = FALSE
4:     fake = FALSE
5:     for b = 0 to m − 1 do
6:         if CP[b] ≠ TP[b] then
7:             diff = TRUE
8:         else
9:             fake = TRUE
10:        end if
11:        if (b = m − 1) and (diff = FALSE) then
12:            status = TRUE
13:        else
14:            fake = TRUE
15:        end if
16:    end for
17:    return status
18: end procedure
```

The Attack. The goal of this attack is to retrieve the true PIN even if the number of trials is limited. That is why a template attack is used. It is supposed that an attacker has a profiling device and she can:

- change the true PIN in her profiling device;
- obtain many traces on her profiling device.

The attack starts with a measurement campaign on the profiling device. Template attacks use a divide-and-conquer approach, the digits of the true PIN are attacked separately. For each value of the pairs, candidate PIN and true PIN (one digit each), (v, k) in $[\![0, 9]\!]^2$, traces are collected. $M_{v,k} = \{xk_{(i,j)}\}$, i for trace, j for time.

For effective computations, both for proper floating point arithmetic and to limit the amount of data to handle, a principal component (PCA) preprocessing is used on the data and the matrix $M_{v,k}$ is only considered here after this preprocessing step. After the PCA, our traces have always 10k points each.

The covariance matrix $S_{v,k} = \left\{sk_{(j,j')}\right\}$ is computed.

$$sk_{(j,j')} = \frac{1}{n-1} \cdot \left(xk_j - \overline{xk}_j\right)^t \left(xk_{j'} - \overline{xk}_{j'}\right) \quad .$$

On the targeted device, a first trace $T_v = \{x_j\}$ is collected, where all the candidate PIN digits are equals to $v = 0$. For each digit the attacker confronts the trace T_v to the template matrix $S_{v,k}$, with the general formula in template attacks (Mahalanobis distance):

$$F_v\left(T_v | S_{v,k}, \overline{xk}\right) \propto \cdot \exp\left(-\frac{1}{2} \cdot \left(T_v - \overline{xk}\right) \cdot S_{v,k}^{-1} \cdot \left(T_v - \overline{xk}\right)^t\right) \quad .$$

The attack returns the guess k_v for which F_v is maximal for a given T_v, or ranks the guesses k according to the value of $F_v(T_v, k)$.

4 Experiments

4.1 Targeted Device

A chip which is representative of what can be found in modern smartphones is targeted. Unfortunately, the public documentation on modern SoC is often sparse. To ease our bare-metal development process, we chose a relatively open platform with some documentation (mostly thanks to the community around it): the Raspberry Pi 2 (RPi2). This board possesses a BCM2836 SoC from Broadcom and features:

- a quad-core Cortex-A7 processor (ARMv7) running at 900 MHz,
- a VideoCore IV GPU running at 250 MHz,
- 1 GB of LPDDR2 RAM memory running at 400 MHz (or 450 MHz in Turbo mode),
- TrustZone is available.

The processor uses 2 levels of cache:

L1 : 32 KB, 4-way associative, separate data and instructions,
L2 : 512 KB, 8-way associative, unified data and instructions, shared between GPU and CPU.

The RPi2 board features GPIOs allowing to send electric signals directly from our code. In particular, a GPIO has been used to obtain a trigger when our targeted code is executed. This signal is used by the oscilloscope to know when to measure the leakage.

4.2 Software Implementation

Since we want to measure the impact of the micro-architecture on the side-channel leakage, our targeted applications run on bare-metal (without OS). Therefore there is no context switching due to the processes scheduling or any other interruptions during the measured computation.

Our custom kernel is an aggregation of different functions, it contains two applications. First **AES** is a simple naive AES implementation without any countermeasures. Plaintexts are sent to the board through the UART and ciphertexts are returned. **PIN** is the other application, implementing a secure *verify pin* algorithm (from [28]). PIN candidates are sent to the board through the UART which answers if it is correct or not (for experimental purposes, an unlimited number of tries could be done without the secret destruction).

Both applications are always present in the final binary and one is selected at start-up with a proper command sent on the UART. Additionally, a configuration can be chosen at this step to enable or disable several features, namely the TrustZone activation, and the multicore execution. The following configurations have been tested:

1. *Default (D)*: the applications are run in non-secure kernel mode on a single core. The other cores are trapped in an empty infinite loop. Data and instruction caches are enabled but not the MMU.
2. *Multicore (M)*: same as default but now the other cores are active. They generate two pseudo random numbers (using the xorshift [22] algorithm) and compute their GCD in an infinite loop. This simulates an intense computation on all cores (asynchronously from our targeted applications).
3. *Secure (S)*: same as default but the application is executed in secure kernel mode. The other core are inactive.
4. *Secure and Multicore (S+M)*: this configuration executes our application in secure kernel mode while the other cores are active, computing GCD in non-secure kernel mode.

4.3 Test Bench

Our EM measurement test bench includes the following devices:

- An EM probe from Langer either a *RF-R 0.3-3* or a *RF-R 400-1* both with a 30 MHz to 3 GHz bandwidth (see below),
- a Langer PA 303 preamplifier (3 GHz bandwidth) to amplify the signal from the probe,
- a DSOS404A oscilloscope from Keysight with a 4 GHz bandwidth able to capture up to 20 GSamples per second.

A control computer (Xeon E5-1603v3 @ 2.80 GHz, 4 cores, 40 GB RAM) is used to orchestrate the measurements and perform the analysis with home-made tools. In particular, special care was needed when developing our tools to achieve a nice measurement speed, and be able to manage the vast quantity of data that

resulted from the experiments. Around 7 h and 100 GB of free disk space are necessary to capture the needed traces for the template on our test bench, and 70 h to perform an attack.

4.4 Preliminary Experiments

Now that the applications and the test bench have been described, preliminary experiments need to be done in order to find information leakage. Where should the EM probe be located, what can we see at this location?

For this part, a *RF-R 0.3-3* probe from Langer was used (bandwidth from 30 MHz to 3 GHz). The probe is moved over the board (not only the chip since leakage can occur at other locations) and on both sides of the board.

Then data is measured for 250 ns both during a real sbox computation and when no computation at all is performed (idle state).

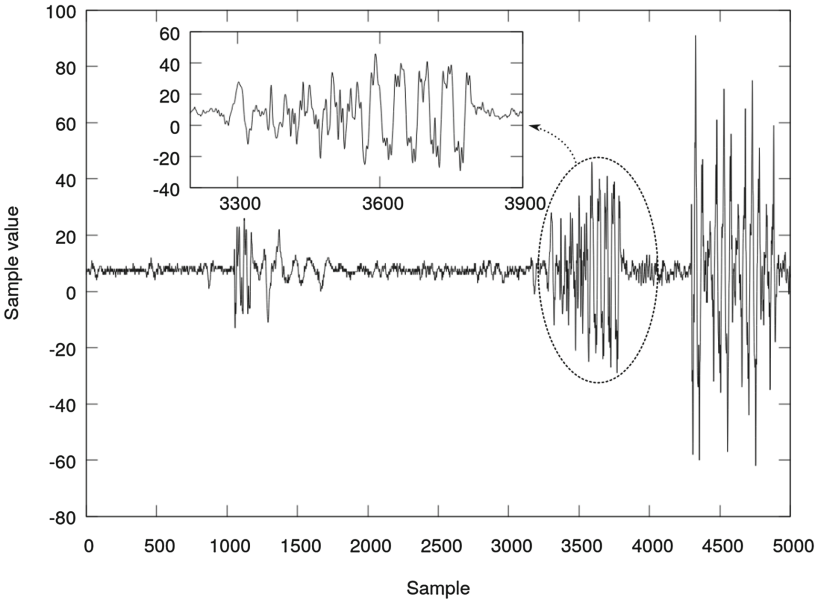

Fig. 1. A trace measured during an s-box computation. A pattern which may be a memory access is stressed out.

A measured trace is shown on Fig. 1 corresponding to an s-box computation. Patterns can clearly be seen but do not seem synchronized with our computation. A spectrum measurement shows that this pattern main frequency is 400 MHz which corresponds to the RAM memory frequency.

Yet even in the presence of these patterns, no leakage was detected with the small *RF-R 0.3-3* probe.

Using the big *RF-R 400-1* probe instead, the same patterns are present but a leakage can now be detected.

Finally, after a trial-and-error phase, we settle to use the big *RF-R 400-1* probe on the back side of the board since this configuration was the best for us to detect a leakage with a CPA. We cannot guarantee that there is no location were the leakage is more spatially concentrated in the absence of a systematic spatial characterization of the leakage (requiring additional apparatus).

4.5 Experimental Results

CPA. The first attacks are CPA on the first round of an AES encryption. Since we are mostly interested in the leakage, we focus on the recovery of one key byte only, yet the attack can easily be generalized to retrieve the full key. The S-boxes are implemented using a look-up table (C array) and no countermeasures are used.

What Does Not Work. To perform the attack, various tries (probe, synchronization points, models) have to be done to find a leakage on the *default (D)* configuration.

We do not find any leakage with the small *RF-R 0.3-3* probe, whatever the location (which does not mean that a leaking location for this probe does not exists). The *RF-R 400-1* probe is used instead and allows to find a leakage.

The classic Hamming Weight model (meaning that the EM leakage is linearly dependant on the Hamming Weight of the value of interest) does not work. We find the EM leakage is directly linked to the value instead. This was not expected and may hint that new leakage models must be developed for modern SoC.

Finally, the synchronization is the most difficult part of an experimental SCA. In our case, our application runs on bare-metal and no synchronization issue can be caused by an operating system or a virtual machine. Nonetheless, S-boxes are implemented with look-up tables. Meaning that desynchronization can be caused by the memory hierarchy fetching the s-box result. Indeed, when measuring the duration of an s-box with a GPIO raised and fallen before and after the s-box, we observe an important jitter (which can also be attributed to the GPIO subsystem). In order to properly synchronize trigger signal on the first s-box result, a GPIO is raised just before the second s-box execution. Since this second execution is done as soon as the first result is obtained we are synchronous with the first s-box result.

CPA Leakage Analysis. The CPA results can give information on the leakage characteristics. A CPA is carried on 200k traces of 5k points under 20 min: 18 min to acquire the data and 2 min to compute the CPA results.

The highest positive correlation peak lasts around 20 samples, i.e. 1 ns. By comparison, the core clock has a 900 MHz frequency, i.e. a clock cycle lasts 1.1 ns. In particular, it means that our measurement jitter is low enough to detect a leakage that lasts 1 ns. Additionally it explains why synchronization is so hard on high end devices: a 2 ns jitter can add a significant noise over a 1 ns signal

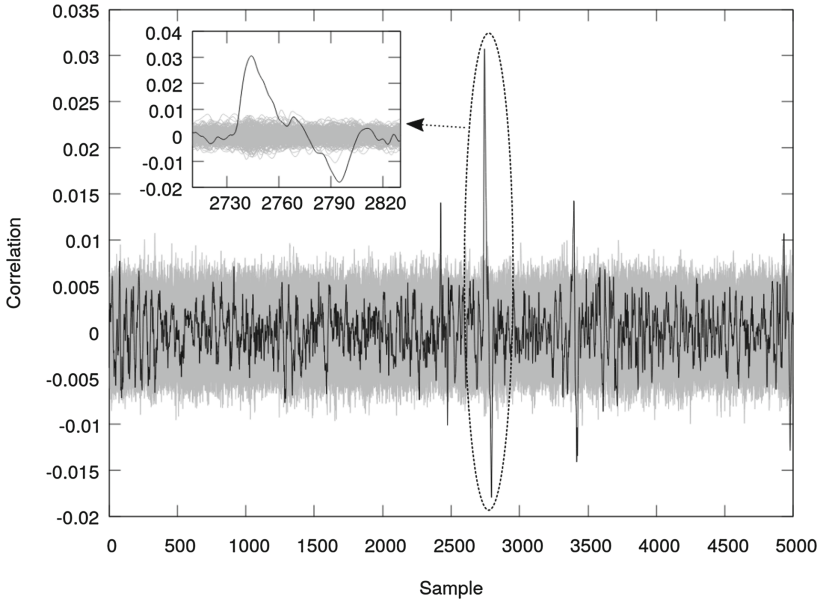

Fig. 2. CPA result for the Default (D) configuration, 200k traces and 20 GS/s (smallest features take 7 samples as per the maximum bandwidth (3 GHz)). Black is for the good key value, grey for all others.

(by comparison, light can travel only around 30 cm in 1 ns). Smaller correlation peaks can be observed: one is 12 ns before the main peak, a second is 41 ns after the same main peak (timing measured for positive peak) (Fig. 2).

All positive peaks are followed by a negative one, this common feature in CPA is usually explained by a value-dependent temporal shift, i.e. in the Hamming Weight model, a 1 or a 0 on a wire have different associated capacitance and as a consequence different speed. However in our settings, the Hamming Weight model is not correct and therefore this explanation does not hold. Yet we did not found a proper explanation for this phenomenon.

We cannot have any certitude to explain these peaks without further information on the chip microarchitecture. Yet the timings suggest a fast memory access (cache): we may be visualizing the data transiting in the memory hierarchy.

CPA Results for the Different Configurations. In this paragraph, the influence of the different configurations is tested on the CPA results.

The correlation values as shown on Table 1 around 3%, is low in all cases. The leakage signal is hidden behind the noise generated by the numerous SoC subsystems. Yet the power of the CPA means that the secret can be found even in noisy cases. The results shown here required $200k$ traces acquired in 18 min.

Table 1. For 5000 samples, 200k traces and a total duration of 250 ns (20 GS/s)

Configuration	Maximum correlation	Peak sample value
`D`	3.1%	2745
`S`	3.4%	2746
`M`	3.7%	2746
`S+M`	3.6%	2747

The leakage peaks appeared nearly at the same sample for all configurations, simply meaning that the S and M configurations do not impact the timing of the leakage (no cache/memory interference).

Template Attacks. The second attack is a template attack on a Verify PIN algorithm. At first, the small *RF-R 0.3-3* probe has been used. Attacks where successful but results were not reproducible: any small variation in our setup (probe position in particular) gave different results.

Since we are interested in the variation of the efficiency of the attack with or without some microarchitecture features enabled, the big *RF-R 400-1* probe was preferred (giving reproducible results).

Finally, we use the same *RF-R 400-1* probe for both attacks.

The results of these attacks are presented in the Table 2. One attack (one row of Table 2) requires 7 h of measurements and 70 h of computation split as follows:

- 1 h to perform the PCA learning step (create the transformation matrix),
- 3 h to apply the PCA transformation over all acquired traces,
- 10 h to compute the template classes,
- 50–60 h to perform the attack on the 150k test traces.

The template attack [6] is particularly efficient to detect whether the secret PIN digit is equal to the candidate or not. Here, the success metric used is the ratio of successful detection that the secret PIN digit is equal to the candidate (for 10 digit values, a purely random method would have a 10% success rate).

For the 10 secret digit values (for 1 candidate value only), template classes where created from the measurement of 200k traces. In other words, for each secret value, 200k traces where measured and used to characterize the signal corresponding to this secret (forming one template class). This is the learning dataset.

Then 15k additional traces where measured, here again for each secret value. This is the test dataset. Our success rate is measured by trying to match these $10 \cdot 15k = 150k$ traces to each one of the 10 template classes.

Let r be the actual, true, success rate and p the measured success rate. We can define the maximum error as a value E guaranteeing $|p - r| < E$. Let σ be our confidence level, i.e. $\sigma = 3$ means that we have a 3 standard deviations confidence (≈99.7%).

We can link our maximum error and our confidence in this error with the Eq. 1.

$$E = \sigma \cdot \sqrt{\frac{p \cdot (1-p)}{n}} \tag{1}$$

where n is the number of measurements (normal approximation method of the binomial confidence interval).

For $p = 0.5, n = 15000$ and $\sigma = 3$, we obtain $E = 1.22\%$ (E is maximized for $p = 0.5$). In the following result tables, the success rate is given with a confidence of 3 standard deviations (99.7%) that the error is below 1.22%.

Table 2. Success rates of the template attacks.

Setup	Sampling rate	Measure duration	Success rate
`D`	10 Gsamples/s	2 μs (20 kpts)	37.84%
`S`	10 Gsamples/s	2 μs (20 kpts)	38.3%
`M`	10 Gsamples/s	2 μs (20 kpts)	37.88%
`S+M`	10 Gsamples/s	2 μs (20 kpts)	36.40%
`D`	20 Gsamples/s	2.5 μs (50 kpts)	35.19%
`S`	20 Gsamples/s	2.5 μs (50 kpts)	32.85%
`M`	20 Gsamples/s	2.5 μs (50 kpts)	18.45%
`S+M`	20 Gsamples/s	2.5 μs (50 kpts)	17.81%

First, we observe on Table 2 that with a 10 GS/s sampling rates, the attacks success rate is approximately 37.3% independently of the configuration (all results are in our error interval of 1.22%). **The TrustZone and the activity of other cores have no effect on the attack success rate.**

It can be observed on the last two rows of Table 2 that using a bigger sampling rate can **decrease** the success rate, below our error interval. We attribute this paradoxical effect to our principal component analysis (PCA) preprocessing that select the dimensions maximizing the variance. Since at the end of the PCA 10 kpts traces are obtained whatever the size of the input traces (20 kpts or 50 kpts), it is possible that the preprocessing reduces our success rate by maximizing variance not linked with the secret, such as variance generated by the activity of other cores.

Therefore, the preprocessing is able to efficiently compress the template data (traces size divided by 5), but at the cost of a reduced success rate. Template attacks may be less sensitive to bad synchronization but they require much bigger computing resources and one has to be careful with the preprocessing (as Table 2 shows).

5 Conclusion

The difficulty of adapting side-channel attacks to high-end devices has often been supposed. We have seen that the true difficulty arises from the need for a more precise synchronization with the leakage.

If the attacker is able to synchronize, simple attacks (CPA) have been shown to be successful. The activity of other cores and the utilization of TrustZone have no effect on the attacks efficiency. As expected the TrustZone, a hardware-assisted software protection mechanism, offers no SCA protection. It may be a problem when most TrustZone capable devices are embedded systems that may endure physical attacks.

Yet if synchronization is not possible, there are some attacks (e.g. template attacks) that are able to exploit the time dimension as any other dimension. We hint that such techniques may be more common in the future to overcome the true difficulty with high-end devices: dealing with time precision (problem even worse with an OS or a virtual machine). Here again TrustZone or the activity of other cores have no incidence. But with these attacks, managing the big amount of data generated by our measures may prove to be the limiting factor, requiring better computing resources.

Even if side-channel attacks techniques have to be adapted to the new challenges created by high-end devices, there is no reasons that they will not be as successful as on smartcards. The smartphone industry in particular should rapidly move to SCA-proof technologies before the development of these attacks.

References

1. TEE Protection Profile. http://www.commoncriteriaportal.org/files/ppfiles/anssi-profil_PP-2014_01.pdf
2. Aboulkassimi, D., Agoyan, M., Freund, L., Fournier, J., Robisson, B., Tria, A.: Electromagnetic analysis (EMA) of software AES on Java mobile phones. In: 2011 IEEE International Workshop on Information Forensics and Security, pp. 1–6, November 2011
3. Archambeau, C., Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Template attacks in principal subspaces. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 1–14. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_1
4. Balasch, J., Gierlichs, B., Reparaz, O., Verbauwhede, I.: DPA, bitslicing and masking at 1 GHz. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015, pp. 599–619. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_30
5. Betters, E.: Apple pay: How it works (2016). http://www.pocket-lint.com/news/130870-apple-pay-explained-what-is-it-and-how-does-it-work. Accessed 14 Feb 2017
6. Bouder, H.L., Barry, T., Couroussé, D., Lashermes, R., Lanet, J.L.: A template attack against VERIFY PIN algorithms. In: Secrypt (2016)
7. 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

8. 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
9. Clavier, C.: An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. In: McDaniel, P., Gupta, S.K. (eds.) Information Systems Security, pp. 143–155. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77086-2_11
10. Corpuz, J.: Mobile password managers (2017). http://www.tomsguide.com/us/pictures-story/662-best-mobile-password-managers.html. Accessed 14 Feb 2017
11. Costan, V., Devadas, S.: Intel SGX explained. Cryptology ePrint Archive, Report 2016/086 (2016). http://eprint.iacr.org/2016/086
12. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E.: Stealing keys from PCs using a radio: cheap electromagnetic attacks on windowed exponentiation. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 207–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_11
13. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., Yarom, Y.: ECDSA key extraction from mobile devices via nonintrusive physical side channels. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1626–1638. ACM, New York (2016). https://doi.org/10.1145/2976749.2978353
14. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_27
15. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
16. Koeune, F., Standaert, F.-X.: A tutorial on physical security and side-channel attacks. In: Aldini, A., Gorrieri, R., Martinelli, F. (eds.) FOSAD 2004-2005. LNCS, vol. 3655, pp. 78–108. Springer, Heidelberg (2005). https://doi.org/10.1007/11554578_3
17. Foo Kune, D., Kim, Y.: Timing attacks on pin input devices. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 678–680. ACM (2010)
18. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: Armageddon: cache attacks on mobile devices. In: USENIX Security Symposium, pp. 549–564 (2016)
19. Longo, J., De Mulder, E., Page, D., Tunstall, M.: SoC It to EM: electromagnetic side-channel attacks on a complex system-on-chip. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 620–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_31
20. Mangard, S.: A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 343–358. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36552-4_24
21. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards, vol. 31. Springer, Heidelberg (2008). https://doi.org/10.1007/978-0-387-38162-6
22. Marsaglia, G., et al.: Xorshift RNGs. J. Stat. Softw. **8**(14), 1–6 (2003)
23. Nguyen, L.: Samsung pay: How it works (2016). http://www.androidauthority.com/samsung-pay-everything-you-need-to-know-678123/. Accessed 14 Feb 2017
24. NIST: Specification for the Advanced Encryption Standard. FIPS PUB 197 197, November 2001

25. Oswald, E.: Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 82–97. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_8
26. 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
27. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
28. Riviere, L.: Sécurité des implémentations logicielles face aux attaques par injection de faute sur systemes embarqués. Ph.D. thesis, Telecom Paris Tech (2015)
29. Uno, H., Endo, S., Hayashi, Y.I., Homma, N., Aoki, T.: Chosen-message electromagnetic analysis against cryptographic software on embedded OS. In: 2014 International Symposium on Electromagnetic Compatibility, Tokyo, pp. 314–317, May 2014
30. Zhang, N., Sun, K., Shands, D., Lou, W., Hou, Y.T.: Truspy: cache side-channel information leakage from the secure world on arm devices (2016)

Defences and Evaluation

Formalising Systematic Security Evaluations Using Attack Trees for Automotive Applications

Madeline Cheah(✉), Hoang Nga Nguyen, Jeremy Bryans, and Siraj A. Shaikh

Institute for Future Transport and Cities, Coventry University, Coventry CV1 5FB, UK
{cheahh2,hoang.nguyen,jeremy.bryans,siraj.shaikh}@coventry.ac.uk

Abstract. Vehicles are insecure. To protect such systems, we must begin by identifying any weaknesses. One approach is to apply a systematic security evaluation to the system under test. In this paper we present a method for systematically generating tests based on attack trees. We formalise the attack trees as provably-equivalent process-algebraic processes, then automatically generate tests from the process-algebraic representation. Attack trees may include manual input (and thus so will some test cases) but scriptable test cases are automatically executed. Our approach is inspired by model based testing, but allows for the fact that we do not have a specification of the system under test. We demonstrate this methodology on a case study and find that this is a viable method for automation of systematic security evaluations.

Keywords: Automotive security · Attack trees · Secure design
Security testing · Bluetooth

1 Introduction

Vehicles are extremely reliant on software and connectivity to enable the functionality desired by customers. The explosion of diverse technologies in cars has meant that cybersecurity of vehicles has become a mainstream concern for manufacturers; such concerns arise out of a number of attacks on components [4,14], internal vehicular network [5,12] and through external interfaces [4,10]. Systematic and automated testing to assure against many such attacks is a challenge, and an ad-hoc approach (i.e. a subjective prioritisation of when and where to test) usually means there is less than optimal coverage of vulnerabilities.

The contribution of this paper is a well-founded methodology to systematically evaluate the cybersecurity of a vehicle, using a model checker and a translation of attack trees into a process algebra. The methodology is inspired by model-based security testing.

The rest of the paper is organised as follows. In Sect. 2, we discuss related work. Section 3 contains the semantics of attack trees as source-sink graphs and

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 113–129, 2018.
https://doi.org/10.1007/978-3-319-93524-9_7

the traces model of CSP [20]. This is followed by an overview of our methodology in Sect. 4, including a translation function for attack trees into the traces model of CSP, and a demonstration of the equivalence of the two models. We then discuss the implementation of this methodology (Sect. 5) and apply it to a case study involving diagnostic devices that attach to the vehicular on-board diagnostics port (Sect. 6). Section 7 concludes the paper.

2 Related Work

We discuss the attack tree formalism in an automotive context in Sect. 2.1 and model based security testing in Sect. 2.2.

2.1 Attack Trees

The foundations of formal descriptions of attacks were laid by [15] and in particular their process algebraic nature was recognised in [28]. Automotive specific attack trees have also been discussed in literature where [24] looked at attack tree generation and gave formal descriptions of the trees. This is orthogonal to our research (in translating low level attack trees) as in our case the trees have already been pre-built based on reconnaissance of a black box system, rather than the automatic generation of an attack tree from a fully specified system-under-test (SUT).

Attack tree generation is still challenging as the SUT has unknown specifications (black box). Other examples of attack trees in the automotive context are those as described in the 'E-safety vehicle intrusion protected applications' (EVITA) project [21] and the SAE J3061 Cybersecurity Guidebook for Cyber-Physical Vehicle Systems [23], although these are informal. Application of executable attack trees has been demonstrated [3], however, automation was limited to test case execution. The attack trees were informally described and manually created.

2.2 Model-Based Security Testing

Model-based security testing (MBST) is a special case of model-based testing (MBT) with a focus on security requirements. One can find a succinct classification of different approaches with respect to MBT and MBST in [8,27], where MBT is comprised of the following: (1) A formal model of the SUT is collected usually from specification documents as the result of the design phase of the SUT, or constructed from system requirements. This model can be at different abstraction levels where the most abstract one contains all possible behaviours; (2) Test case filter criteria are then determined. They are usual informal descriptions capturing a subset of behaviours of the formal model which can be drawn from the requirements or the structure of the formal model (e.g., state coverage, transition coverage). In terms of MBST, these criteria concentrate on system security such as security properties (e.g., confidentiality, integrity and availability), security mechanisms, and the environment (e.g., attack strategies).

(3) These criteria are then formalised into test case specifications, usually compatible with the formal model. (4) Given the formal model of the SUT and the test case specifications, test cases are automatically generated by different technologies such as model checking, theorem proving, or graph search algorithms. Finally, (5) these test cases are executed on the SUT to provide verdicts. This involves translation of test cases from the abstract level of the formal model into concrete test scripts that are executable on SUT. Then, the execution result is translated back to the abstract level for comparison with the expected output of the generated test cases.

3 Semantic Models

In this section we give the formal notation and source-sink semantics of attack trees (Sect. 3.1) and the notation and finite trace semantics of the formal language Communicating Sequential Process (CSP) (Sect. 3.2).

3.1 Attack Trees

Attack trees were first created to describe the actions of an attacker in an methodical manner [25].

Attack trees contain a set of leaf nodes, structured using the operators conjunction (`AND`) and disjunction (`OR`). The leaf nodes represent atomic attacker actions, the `AND` nodes are complete when all child nodes are carried out, and `OR` nodes are complete when at least one child node is complete.

Extensions have been proposed using `Sequential AND` (or `SAND`) [13]. There are two types of ordering: time dependent or condition dependent. In this paper we adopt the condition dependent paradigm.

We follow the formalisation of attack trees given in [13,15]. If $\mathbb{A}$ is the set of possible atomic attacker actions, the elements of the attack tree $\mathbb{T}$ are $\mathbb{A} \cup \{OR, AND, SAND\}$, and an attack tree is generated by the following grammar, where $a \in \mathbb{A}$:

$$t ::= a \mid OR(t, \ldots, t) \mid AND(t, \ldots, t) \mid SAND(t, \ldots, t)$$

Attack tree semantics have been defined by interpreting the attack tree as a set of series-parallel (SP) graphs [13]. Definition of SP graphs requires first the definition of source-sink graphs and here we use the definitions from [13].

Definition 1: A source-sink graph over $\mathbb{A}$ is a tuple $G = (V, E, s, z)$ where V is a set of vertices, E is a multiset of edges with support $E* \subseteq V \times \mathbb{A} \times V$, $s \in V$ is a unique source and $z \in V$ is a unique sink, and $s \neq z$.

The sequential composition of G and another graph G', denoted by $G \cdot G'$ results from the disjoint union of G and G' and linking the sink of G with the source of G'. Thus, if $\dot\cup$ denotes the disjoint union and $E^{[s/z]}$ denotes the multiset of E where vertices z are replaced by s, then $G.G'$ can be defined as:

$$G \cdot G' = (V \setminus \{z\} \dot\cup V', E^{[s'/z']} \dot\cup E', s, z')$$

Parallel composition, denoted by $G \parallel G'$ is similar (differing only in that two sources and two sinks are identified) and can be defined as:

$$G \parallel G' = (V \setminus \{s, z\} \dot{\cup} V', E^{[s'/s, z'/z]} \dot{\cup} E', s', z')$$

Definition 2: The set $\mathbb{G}_{\mathcal{SP}}$ over $\mathbb{A}$ is defined inductively by:

For $a \in \mathbb{A}$, $\xrightarrow{a}$ is an SP graph,
If G and G' are SP graphs, then so are $G \cdot G'$ and $G \parallel G'$.

Hence, the full SP graph semantics for attack tree $\mathbb{T}$ can be given by the function:

$$[\![\cdot]\!]_{\mathcal{SP}} : \mathbb{T} \to \mathcal{P}(\mathbb{G}_{\mathcal{SP}})$$

This is defined recursively. If $a \in \mathbb{A}$, $t_i \in \mathbb{T}$, and $1 \leq i \leq k$, then

$[\![a]\!]_{\mathcal{SP}} = \{\xrightarrow{a}\}$
$[\![OR(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \bigcup_{i=1}^{k} [\![t_i]\!]_{\mathcal{SP}}$
$[\![AND(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \{G_1 \parallel \ldots \parallel G_k \mid (G_1, \ldots, G_k) \in [\![t_1]\!]_{\mathcal{SP}} \times \ldots \times [\![t_k]\!]_{\mathcal{SP}}\}$
$[\![SAND(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \{G_1 \cdot \ldots \cdot G_k \mid (G_1, \ldots, G_k) \in [\![t_1]\!]_{\mathcal{SP}} \times \ldots \times [\![t_k]\!]_{\mathcal{SP}}\}$
where $[\![t]\!]_{\mathcal{SP}} = \{G_1, \ldots, G_k\}$ corresponds to a set of possible attacks G_i

Since, in this paper, we base the construction of the attack tree on penetration testing techniques, all leaves on the tree can be considered actions. The combination of these actions can be translated into the processes that form part of a test case. This is conducive to the use of process algebra such as Communicating Sequential Processes (CSP) and furthermore the equivalence of the semantics (see Sect. 3.2) means that we can use synonymous operators to transform a pre-built attack tree.

3.2 CSP

We give here a brief overview of the subset of CSP that we use in this paper. A more complete introduction may be found in [20].

Given a set of events Σ, CSP processes are defined by the following syntax:

$$P ::= \mathit{Stop} \mid e \to P \mid P_1 \,\Box\, P_2 \mid P_1;\ P_2 \mid P_1 \underset{A}{\parallel} P_2 \mid P_1 \,|||\, P_2$$

where $e \in \Sigma$
and $A, B \subseteq$ events. For convenience, the set of CSP processes defined via the above syntax is denoted by CSP.

To mark the termination of a process, a special event ✓ is used.

In the above definition, the process *Stop* is the most basic one, which does not engage in any event and represents deadlock. In addition, *Skip* is an abbreviation for $\checkmark \to \mathit{Stop}$. It only exhibits ✓ and then behaves as *Stop*.

The prefix $e \to P$ specifies a process that is only willing to engage in the event e, then behaves as P. The external choice $P_1 \,\Box\, P_2$ behaves either as P_1

or as P_2. . The sequential composition $P_1;\ P_2$ initially behaves as P_1 until P_1 terminates, then continues as P_2.

The generalised parallel operator $P_1 \underset{A}{\parallel} P_2$ requires P_1 and P_2 to synchronise on events in $A \cup \{\checkmark\}$. All other events are executed independently. Finally, the interleaving operator $P_1 \,|||\, P_2$ allows both P_1 and P_2 to execute concurrently and independently, except for $\checkmark$.

There are different semantics models for CSP processes [20]. For the purpose of this paper, we recall the finite trace semantics. A *trace* is a possibly empty sequence of events from Σ and may terminate with $\checkmark$. As usual, let Σ^* denote the set of all finite sequences of events from Σ, $\langle\rangle$ the empty sequence, and $tr_1 \frown tr_2$ the concatenation of two traces tr_1 and tr_2; then the set of all traces is defined as $\Sigma^{*\checkmark} = \{tr \frown en \mid tr \in \Sigma^* \wedge en \in \{\langle\rangle, \langle\checkmark\rangle\}\}$.

The trace tr_1 is a *prefix* of a trace tr_2, written as $tr_1 \leq tr_2$, iff $\exists\, tr' : tr_1 \frown tr' = tr_2$. Events in $A \subseteq \Sigma \cup \{\checkmark\}$ may be abstracted away from a trace tr by a hiding operator, written as $tr \setminus A$ and defined as

$$tr \setminus A = \begin{cases} \langle\rangle & \text{if } tr = \langle\rangle \\ \langle a\rangle \frown (tr' \setminus A) & \text{if } tr = \langle a\rangle \frown tr' \wedge a \notin A \\ tr' \setminus A & \text{if } tr = \langle a\rangle \frown tr' \wedge a \in A. \end{cases}$$

For convenience, when $A = \{a\}$, we shall simply write $tr \setminus a$. In general, the trace semantics of a process P is a subset $traces(P)$ of $\Sigma^{*\checkmark}$ consisting of all traces which the process may exhibit. It is formally defined recursively as follows:

- $traces(Stop) = \{\langle\rangle\}$;
- $traces(e \rightarrow P) = \{\langle\rangle\} \cup \{\langle e\rangle \frown tr \mid tr \in traces(P)\}$;
- $traces(P_1 \,\square\, P_2) = traces(P_1) \cup traces(P_2)$;
- $traces(P_1;\ P_2) = traces(P_1) \cap \Sigma^*$
  $\cup\, \{tr_1 \frown tr_2 \mid tr_1 \frown \langle\checkmark\rangle \in traces(P_1) \wedge tr_2 \in traces(P_2)\}$;
- $traces(P_1 \underset{A}{\parallel} P_2) = \{tr \in tr_1 \underset{A}{\parallel} tr_2 \mid tr_1 \in traces(P_1) \wedge tr_2 \in traces(P_2)\}$
  where $tr_1 \underset{A}{\parallel} tr_2 = tr_2 \underset{A}{\parallel} tr_1$ is defined as follows with $a, a' \in A$ and $b, b' \notin A$:
  - $\langle\rangle \underset{A}{\parallel} \langle\rangle = \{\langle\rangle\}$; $\langle\rangle \underset{A}{\parallel} \langle a\rangle = \emptyset$; $\langle\rangle \underset{A}{\parallel} \langle b\rangle = \{\langle b\rangle\}$;
  - $\langle a\rangle \frown tr_1 \underset{A}{\parallel} \langle b\rangle \frown tr_2 = \{\langle b\rangle \frown tr \mid tr \in \langle a\rangle \frown tr_1 \underset{A}{\parallel} tr_2\}$;
  - $\langle a\rangle \frown tr_1 \underset{A}{\parallel} \langle a\rangle \frown tr_2 = \{\langle a\rangle \frown tr \mid tr \in tr_1 \underset{A}{\parallel} tr_2\}$
  - $\langle a\rangle \frown tr_1 \underset{A}{\parallel} \langle a'\rangle \frown tr_2 = \emptyset$ where $a \neq a'$;
  - $\langle b\rangle \frown tr_1 \underset{A}{\parallel} \langle b'\rangle \frown tr_2 = \{\langle b\rangle \frown tr \mid tr \in tr_1 \underset{A}{\parallel} \langle b'\rangle \frown tr_2\} \cup$
    $\{\langle b'\rangle \frown tr \mid tr \in \langle b\rangle \frown tr_1 \underset{A}{\parallel} tr_2\}$
- $traces(P_1 \,|||\, P_2) = \{tr \in tr_1 \,|||\, tr_2 \mid tr_1 \in traces(P_1) \wedge tr_2 \in traces(P_2)\}$
  where $tr_1 \,|||\, tr_2 = tr_1 \underset{\emptyset}{\parallel} tr_2$.

  Therefore, $traces(P_1 \,|||\, P_2) = traces(P_1 \underset{\emptyset}{\parallel} P_2)$.

A process P is said to *trace-refine* a process Q (written $Q \sqsubseteq_T P$) if $traces(P) \subseteq traces(Q)$. There are other flavors of refinement, but we restrict ourselves to trace refinement below.

4 Methodology

In this section we present an overview of the whole methodology (Fig. 1) of our paper. The context of our work is the automotive industry. Vehicle manufacturers often incorporate off-the-shelf (OTS) components into their work, and their specifications are not always available. Manufacturers thus have to approach testing with this uncertainty.

We begin by assuming a System under Test (SUT), which may be either an OTS component or contain one. We also assume the existence of a corresponding attack tree (see Fig. 1). Section 6 illustrates our approach with an attack tree developed for a vehicle network that includes a Bluetooth connection. It is worth observing here that an attack tree for Bluetooth systems essentially systematises the known attacks on Bluetooth, and therefore although it may be updated as new attacks are constructed, the development of the attack tree is a one-off cost. The same attack tree will work for any Automotive Bluetooth system.

If a specification (or abstract model) of the SUT is available we use it, but in many cases (including the example in Sect. 6) the specification of the SUT is confidential or contains confidential components. In this case we can under-approximate a specification to begin the process. Successive iterations of the process allow us to refine this under-approximation. We illustrate this under-approximation in Sect. 5. The approach is to generate tests against this model. Tests are automatically generated using FDR, the refinement checker for CSP. The specification and the attack tree are compared. Each possible route through the attack tree represents a potential attack, and the Test Case Generator compiles a list of all the attacks that the specification permits. Note that in the case of an under-approximated specification all possible attacks will be permitted.

The next step is to convert the formal tests into implementation tests. This process is detailed in Sect. 5.2. Note that these implementation tests are in fact attacks (or potential attacks) on the SUT. Not all the test implementation tests generated from an attack tree can be fully automated. The attack tree may contain nodes that require manual input, in which case the implementation tests will require (partial) manual interaction. The ones that do not require manual input may be executed directly on a Testbed. The report contains the test results. Since tests are really attacks on the SUT, we consider a test to be successful if the attack succeeds.

In the remainder of this section we present in more detail the transformation of attack trees to CSP processes (Sect. 4.1), as well as a proof of the equivalence of the semantic models.

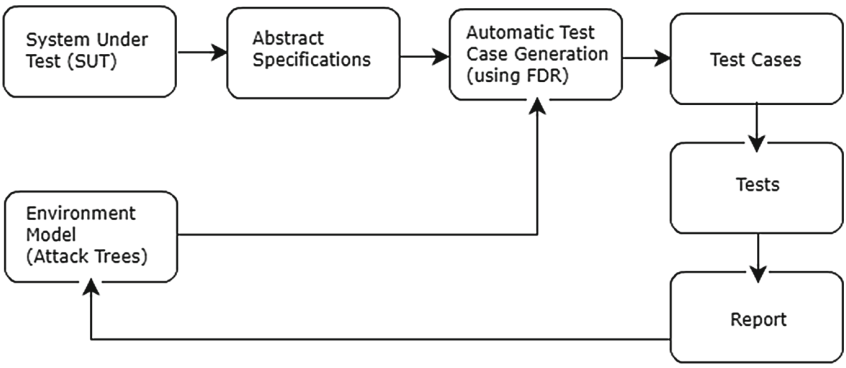

Fig. 1. Overview of our formal systematic security methodology

4.1 Transforming Attack Trees into CSP Processes

In this section we use the process algebra CSP to describe the attack tree. We choose CSP because as a process algebra it is able to represent and combine the actions of the attack tree into a set of processes that could subsequently be used for test case generation.

In principle, the logic gates of the attack tree can be considered CSP operators [11] as follows:

- Since the `AND` logic gate demands that all actions must be successful for the branch to be considered complete, the interleave operator ($|||$) is used. This operator joins processes that operate concurrently but without them necessarily interacting or synchronising.
- The sequential composition operator (;) is used for the `SAND` logic gate. The former echoes the `SAND` logic gate, in that the first process must terminate successfully before the next is allowed;
- The external choice operator ($\Box$) (where any process could be chosen dependent on the environment in which it operates) is used for the `OR` logic gate.

Formally, we define the following transformation function trans : $\mathbb{T}_{SAND} \rightarrow CSP$ where $\Sigma = \mathbb{A}$:

- $\text{trans}(a) = a \rightarrow \mathit{Skip}$ for $a \in \mathbb{A}$;
- $\text{trans}(OR(t_1, \ldots, t_n)) = \text{trans}(t_1) \,\Box\, \ldots \,\Box\, \text{trans}(t_n)$;
- $\text{trans}(AND(t_1, \ldots, t_n)) = \text{trans}(t_1) \,|||\, \ldots \,|||\, \text{trans}(t_n)$;
- $\text{trans}(SAND(t_1, \ldots, t_n)) = \text{trans}(t_1);\ \ldots;\ \text{trans}(t_n)$;

In order to show the correctness of the above transformation, it is necessary to make the two semantics in $\mathbb{G}_{\mathcal{SP}}$ and $\Sigma^{*\checkmark}$ compatible for comparison. Recall from [13] that each SP graph represents a possible way to carry out an attack. In such a graph, an AND vertex indicates that actions along its branches must be executed. However, there is no restriction on the order of their executions. In

other words, their executions are interleaving in general. Therefore, it is possible to serialise the actions from a SP graph where parallel compositions of graphs is considered as interleaving and sequential composition as concatenation. Given G in $\mathbb{G}_{\mathcal{SP}}$, let serials($G$) denote the set of all possible ways to serialise G, which is formally defined as follow:

- serials($\xrightarrow{a}$) = $\{\langle a \rangle\}$;
- serials($G_1 \parallel G_2$) = $\{tr \in tr_1 \,|||\, tr_2 \mid tr_1 \in \text{serials}(G_1) \wedge tr_2 \in \text{serials}(G_2)\}$;
- serials($G_1 \cdot G_2$) = $\{tr_1 \frown tr_2 \mid tr_1 \in \text{serials}(G_1) \wedge tr_2 \in \text{serials}(G_2)\}$.

For convenience, we denote the set of all prefixes from serials(G) by pserials(G), i.e., pserials(G) = $\{tr \mid \exists\, tr' \in \text{serials}(G) : tr \leq tr'\}$. We also denote pserials(t) = $\bigcup_{G \in [\![t]\!]_{\mathcal{SP}}} \text{pserials}(G)$ for all attack trees $t \in \mathbb{T}_{SAND}$.

The correctness of transforming attack trees into CSP processes is guaranteed by the following result:

Lemma 1. $\forall\, t \in \mathbb{T}_{SAND}$, pserials($t$) = *traces*(trans($t$)) $\setminus \checkmark$.

Proof. The proof is done by induction on the structure of t.

Base case: Consider $t = a$; then, $[\![t]\!]_{\mathcal{SP}} = \{\xrightarrow{a}\}$ and pserials(t) = $\{\langle\rangle, \langle a \rangle\}$. We also have trans($t$) = $a \rightarrow \mathit{Skip}$ and *traces*(trans(t)) = $\{\langle\rangle, \langle a \rangle, \langle a, \checkmark \rangle\}$. Hence, it is straightforward that pserials(t) = *traces*(trans(t)) $\setminus \checkmark$.

Induction Step:

Case $t = OR(t_1, \ldots, t_n)$**:** It is straightforward that
- $[\![t]\!]_{\mathcal{SP}} = \bigcup_{i=1,\ldots,n} [\![t_i]\!]_{\mathcal{SP}}$, and
- *traces*(trans(t)) = *traces*(trans(t_1) $\Box \ldots \Box$ trans(t_i)).

Then, we have

$$\begin{aligned} tr \in \text{pserials}(t) &\Leftrightarrow \exists\, G \in [\![t]\!]_{\mathcal{SP}} : tr \in \text{pserials}(G) \\ &\Leftrightarrow \exists\, i \in \{1, \ldots, n\}, G \in [\![t_i]\!]_{\mathcal{SP}} : tr \in \text{pserials}(G) \\ &\Leftrightarrow \exists\, i \in \{1, \ldots, n\} : tr \in \text{pserials}(t_i) \\ &\Leftrightarrow tr \in \mathit{traces}(\text{trans}(t_i)) \setminus \checkmark \text{ by induction hypothesis} \\ &\Leftrightarrow tr \in \mathit{traces}(\text{trans}(t)) \setminus \checkmark. \end{aligned}$$

Case $t = AND(t_1, \ldots, t_n)$**:** It is obvious that:
- $[\![t]\!]_{\mathcal{SP}} = \{G_1 \parallel \ldots \parallel G_n \mid G_i \in [\![t_i]\!]_{\mathcal{SP}}\, \forall\, i = 1, \ldots, n\}$, and
- *traces*(trans(t)) = *traces*(trans(t_1) $|||$ $\ldots$ $|||$ trans(t_i)).

Then, we have

$$\begin{aligned} tr \in \text{pserials}(t) &\Leftrightarrow \exists\, G \in [\![t]\!]_{\mathcal{SP}} : tr \in \text{pserials}(G) \\ &\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}} : \\ &\qquad tr \in \text{pserials}(G_1 \parallel \ldots \parallel G_n) \\ &\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}}, tr' \in \text{serials}(G_1 \parallel \ldots \parallel G_n) : \\ &\qquad tr \leq tr' \\ &\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}}, tr_i \in \text{serials}(G_i) : \\ &\qquad tr' \in tr_1 \,|||\, \ldots \,|||\, t_n \wedge tr \leq tr' \\ &\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, tr_i \in \mathit{traces}(\text{trans}(t_i)) \setminus \checkmark : \\ &\qquad tr' \in tr_1 \,|||\, \ldots \,|||\, t_n \wedge tr \leq tr' \text{ by induction hypothesis} \\ &\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, tr_i' \leq tr_i \in \mathit{traces}(\text{trans}(t_i)) \setminus \checkmark : \\ &\qquad tr \in tr_1' \,|||\, \ldots \,|||\, t_n' \\ &\Leftrightarrow tr \in \mathit{traces}(\text{trans}(t)) \setminus \checkmark. \end{aligned}$$

Case $t = SAND(t_1, \ldots, t_n)$**:** It is obvious that:
- $[\![t]\!]_{\mathcal{SP}} = \{ G_1 \cdot \ldots \cdot G_n \mid G_i \in [\![t_i]\!]_{\mathcal{SP}} \, \forall \, i = 1, \ldots, n \}$, and
- $traces(\text{trans}(t)) = traces(\text{trans}(t_1);\ \ldots;\ \text{trans}(t_n))$.

Then we have:

$$
\begin{aligned}
tr \in \text{pserials}(t) &\Leftrightarrow \exists\, G \in [\![t]\!]_{\mathcal{SP}} : tr \in \text{pserials}(G) \\
&\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}} : \\
&\qquad tr \in \text{pserials}(G_1 \cdot \ldots \cdot G_n) \\
&\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}}, tr' \in \text{serials}(G_1 \cdot \ldots \cdot G_n) : \\
&\qquad tr \leq tr' \\
&\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, G_i \in [\![t_i]\!]_{\mathcal{SP}}, tr_i \in \text{serials}(G_i) : \\
&\qquad tr' \in tr_1 \frown \ldots \frown t_n \wedge tr \leq tr' \\
&\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, tr_i \in traces(\text{trans}(t_i)) \setminus \checkmark : \\
&\qquad tr' \in tr_1 \frown \ldots \frown t_n \wedge tr \leq tr' \text{ by induction hypothesis} \\
&\Leftrightarrow \forall\, i \in \{1, \ldots, n\}, \exists\, tr'_i \leq tr_i \in traces(\text{trans}(t_i)) \setminus \checkmark : \\
&\qquad tr \in tr'_1 \frown \ldots \frown t'_n \\
&\Leftrightarrow tr \in traces(\text{trans}(t)) \setminus \checkmark .
\end{aligned}
$$

5 Implementation

We provide a prototype implementation of our proposed methodology. This implementation is built using Python 2.7 and requires as input an attack tree and a formal model of the SUT. It then automatically carries out three main tasks: (1) translates an input attack tree into a CSP process; (2) uses this process and the formal model of the SUT to generate test cases; and (3) executes all the generated test cases by associating each one with a sequence of predefined primitive test scripts. Task 1 is a straightforward implementation of function `trans` from Sect. 4.1. In the rest of this section, we discuss the implementation of tasks 2 and 3 in detail.

5.1 Test Case Generation

Let us assume that the formal model of the SUT is given as a CSP process *Sys*. Furthermore, the behaviours of the attacker are also given in terms of an attack tree t, which is then transformed into a CSP process $\text{trans}(t)$. We shall use trace refinement in CSP to extract test cases following [16]. To this end, $\text{trans}(t)$ acts as a filter criterion to select test cases among all possible runs of the system captured by *Sys*. As in [16], we define a fresh event *attackSucceed* to mark the end of an attack, which indicates that an attack is successfully executed. We form the following filter

$$TestPurpose = \text{trans}(t);\ (attackSucceed \rightarrow Stop)$$

which captures all attacks extended with the marking event *attackSucceed* at the end. Then, we establish the following trace refinement:

$$Sys \,\square\, TestCases \sqsubseteq_T Sys \underset{\Sigma \setminus \{attackSucceed\}}{\parallel} TestPurpose$$

In this refinement, *TestCases* encodes test cases that have previously been generated. By combining it with *Sys* using the external choice operator, a fresh test case, i.e., different from the generated ones, will be generated if one exists. $Sys \underset{\Sigma \setminus \{attackSucceed\}}{\parallel} TestPurpose$ encapsulates all attack traces that can be carried out with respect to the formal model *Sys*. These attack traces are ended with the marking event *attackSucceed*, which does not belong to *Sys*, hence, gives rise to counter examples of the refinement. Initially, $TestCases = TestCases_0 = Stop$, i.e., corresponding to an empty set of test cases. This refinement is checked by calling FDR [26]. If an attack trace exists, FDR will provides a counter example of the form $\langle a_1, \ldots, a_n, attackSucceed \rangle$ where $a_1, \ldots, a_n \in \Sigma \setminus \{attackSucceed\}$. We encode this trace as a test case $tc_1 = a_1 \rightarrow \ldots \rightarrow a_n \rightarrow attackSucceed \rightarrow Stop$. After *TestCases* is rebuilt as $TestCases = TestCases_1 = TestCases_0 \,\Box\, tc_1$, the above refinement check is called again and again to extract further test cases $tc_2, \ldots$ and to construct $TestCase_2, \ldots$ until no further counter example can be found. In this implementation, the calls to checking refinements and extracting counter examples are facilitated by API functions provided by FDR [26].

5.2 Test Case Execution

Test cases that are generated can now be assigned programmatic functions that would allow for execution. This is dependant on implementation of the system and so would necessarily be specific rather than abstract. Furthermore, as the attack tree in this case is based on penetration testing, not all actions (such as "social engineering") are scriptable, largely due to requiring manual intervention. All such actions are indicated in the implementation.

Given an attack tree t, let $scriptable(t)$ denote the set of its scriptable leaves. Then, each scriptable leaf $a \in scriptable(t)$ is associated with a primitive test script $script(a)$. A generated test case $tc = a_1 \rightarrow \ldots \rightarrow a_n \rightarrow attackSucceed \rightarrow Stop$ is automatically executable if $a_i \in scriptable(t)$ for all $i = 1, \ldots, n$. Then, executing an automatically executable test case means to execute all test scripts $script(a_1), \ldots, script(a_n)$ sequentially. If all such scripts are executed successfully, the test case is called *passed*, otherwise *failed*. Note that a passed test case means that the SUT is not secure with respect to the attack encoded by this test case. Conversely, the SUT is impervious to this attack.

In this paper, we use the example of an aftermarket on-board diagnostics (OBD-II) dongle attached to the vehicle (see Sect. 5). Executable test cases are written in Python 2.7 to enable compatibility with Bluetooth functions.

6 Case Study

We take here a case study of evaluating the intra-vehicular network with the attack goal of vehicle compromise. The attack tree (see Sect. 6.2) for this goal is based around access through a Bluetooth-enabled aftermarket device that attaches to the vehicle's on-board diagnostic (OBD-II) port. These devices were

originally created so that enthusiasts and hobbyists were able to read information from their own vehicles for diagnostic and maintenance purposes. The devices (or dongles) contain an ELM327 chip [7] which serves as an RS-232 interpreter. "Attention Modem" (`AT`) commands are used to configure the chip through any serial terminal. The device used for this case study was the OBDLINK MX, with an ELM chip version of 1.3, attached to a 2013 small hatchback from a major manufacturer. Through such a device, an attacker is able to gain access to the intra-vehicular CAN bus remotely and potentially push messages directly into the vehicle [1,2]. The risk of compromise is exacerbated by the fact that these OBD-II devices are usually highly insecure, being wireless, and with weak PINs that are made public (such as `0000` or `1234`) [17].

6.1 Vehicular Communications

Messages that are sent through the dongle on to the CAN bus takes the form of either a raw Controller Area Network (CAN) frame (Sect. 6.1) or a diagnostic message (Sect. 6.1) (that is translated into a CAN frame by the dongle).

CAN Messages. The CAN protocol is the primary mode of communication inside the vehicle. The latest version is CAN 2.0, first specified in 1991 [19] and embodied as an ISO standard (ISO11898) in 2003.

The standard CAN packet comprises (up to) 11 bits for the message ID, followed by (up to) 8 bytes of data, then a cyclic redundancy check (16 bits) for error detection. The full 8 bytes of data need not be used. Information for a door sensor, for example, may only require 1 bit. Conversely a message can be spread across many frames.

Arbitration, should nodes on the CAN network transmit simultaneously, is based on message prioritisation. This prioritisation is determined using the message ID, with the lowest ID being the highest priority; implementation usually means that mission-critical messages are the ones assigned lower IDs.

Assignment of IDs along with data payload is manufacturer specific, however, reuse is common to save on the cost of redesigning a network [18].

Reverse engineering of CAN messages is difficult considering volume and variety of content that is transmitted. This is especially the case without an Original Equipment Manufacturer's (OEM's) typically confidential CAN database, which contains definitions for every message and signal. However, specific CAN messages for discrete events (such as unlocking doors) can be obtained relatively easily through trial and error.

CAN data is transmitted in a bus configuration; any Electronic Control Unit (ECU) on the network has access to all messages. There is no addressing; each ECU listens to a set of IDs which then triggers pre-determined functionality.

Diagnostic Messages. Parameter IDs (PIDs) are used to perform diagnostic functions or request data from the vehicle specifically through OBD-II port; done through a query-response mechanism where a PID query comprises the CAN ID

7DF followed by 8 data bytes. The first byte is data length (usually 02) with the second byte the *mode* and the third byte typically the PID. The combination of modes and PIDs can then be transmitted and a response received from whatever in-vehicle module is responsible. The response CAN ID is typically 8 (in hex) higher than the message ID that the responding ECU answers to.

The first ten modes (01 to 0A), described in SAE J1979 (E/E Diagnostic Test Modes) [22], are standard to all compliant vehicles: PID is only the 2nd byte, with the 3rd to 8th byte unused. With non-standard modes, PIDs could extend to 3rd byte. Manufacturers, are not obliged to implement all standard commands, and additionally could also define functions for non-standard PIDs. There is much information that could be gathered through this port. For example, sending the mode 09 with PID 02 retrieves the Vehicle Identification Number (VIN), which is unique to vehicles, used for maintenance to recovery of stolen vehicles.

6.2 Attack Tree Translation

The attack tree used for this case study is shown in Fig. 2. Figure 2 also lists test scripts corresponding to leaves in this attack tree. If a leaf is not scriptable, it is denoted as a *manual leaf*.

The function trans(Vehicle Compromise) (see Sect. 4.1), gives the translation of this tree into CSP as below:

$$
\begin{aligned}
&Attacker = Vehicle\_Compromise\\
&Vehicle\_Compromise = Connect\_to\_device;\ Cause\_Vehicle\_Compromise\\
&Connect\_to\_device = Using\_legitimate\_device \ \Box\ Spoof\_previously\_paired\_device\\
&Using\_legitimate\_device = Determine\_pairing\_status;\ Connect\_to\_serial\_port\\
&Determine\_pairing\_status = action\_Determine\_pairing\_status \rightarrow Skip\\
&Connect\_to\_serial\_port = action\_Connect\_to\_serial\_port \rightarrow Skip\\
&Spoof\_previously\_paired\_device =\\
&\qquad Find\_the\_link\_key\_from\_local\_or\_remote\_device\\
&\quad |||\ Change\_address\_of\_local\_device\\
&Change\_address\_of\_local\_device =\\
&\quad action\_Change\_address\_of\_local\_device \rightarrow Skip\\
&Find\_the\_link\_key\_from\_local\_or\_remote\_device =\\
&\quad action\_Find\_the\_link\_key\_from\_local\_or\_remote\_device \rightarrow Skip\\
&Cause\_Vehicle\_Compromise =\\
&\qquad Using\_OBD\_messages\\
&\quad \Box\ Run\_through\_all\_messages\\
&\quad \Box\ Flooding\_with\_raw\_CAN\_messages\\
&Flooding\_with\_raw\_CAN\_messages =\\
&\quad Predetermine\_CAN\_messages;\ Send\_\_flood\_with\_\_CAN\_messages\\
&Predetermine\_CAN\_messages =\\
&\qquad Using\_passive\_monitoring\\
&\quad \Box\ Using\_OEM\_CAN\_database\\
&\quad \Box\ Using\_reverse\_engineering\\
&Send\_flood\_with\_CAN_{m}essages = action\_Send\_flood\_with\_CAN\_messages \rightarrow Skip
\end{aligned}
$$

$Using\_OEM\_CAN\_database = action\_Using\_OEM\_CAN\_database \rightarrow Skip$
$Using\_passive\_monitoring = action\_Using\_passive\_monitoring \rightarrow Skip$
$Using\_reverse\_engineering = action\_Using\_reverse\_engineering \rightarrow Skip$
$Run\_through\_all\_messages = Run\_through\_standard;\ Run\_through\_non\_standard$
$Run\_through\_standard = action\_Run\_through\_standard \rightarrow Skip$
$Run\_through\_non\_standard = action\_Run\_through\_non\_standard \rightarrow Skip$
$Using\_OBD\_messages = Flood\_with\_set\_OBD\_messages$
$Flood\_with\_set\_OBD\_messages = action\_Flood\_with\_set\_OBD\_messages \rightarrow Skip$

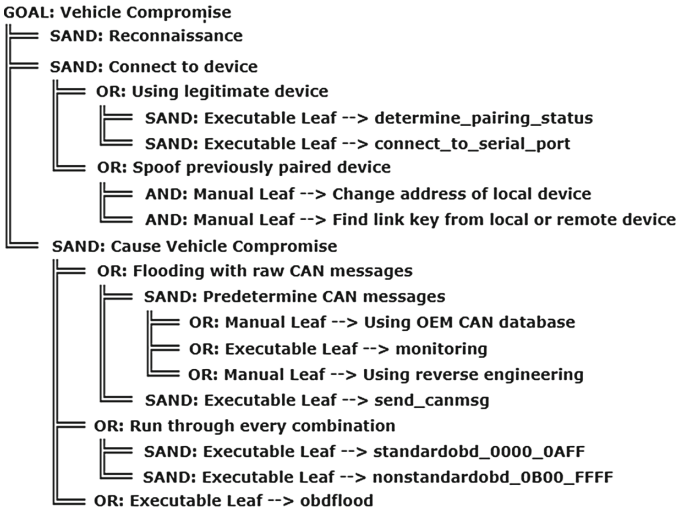

Fig. 2. Attack tree, with attack goal of compromising the vehicle through an aftermarket Bluetooth-enabled OBD-II device

Reconnaissance was defined as "to find as much information as possible" (meaning the subsequently generated formal attack tree would be much larger). Many of the steps were manual and non-sequential.

6.3 Results

We generate test cases using the implementation in Sect. 5.1. Given the small size of the attack tree, we use the most abstract model for the SUT where all behaviours are accepted (the most insecure model) to generate a total of 15 test cases. Results from the run of test cases against an actual implementation are given in Table 1. Three of the test cases passed (i.e. they were executed successfully), and we highlight test case 3 (TC3 in Table 1) for further analysis.

The action of flooding with a particular diagnostic message resulted in loss of function in the vehicle of both electronics and engine. This violates the security property of availability by causing a denial of service. Additionally, injection of

Table 1. Test cases that were run against a real world vehicle

TC#	Execution result
1	unexecutable action_Find_the_link_key_from_local_or_remote_device
2	unexecutable action_Change_address_of_local_device
3	Passed
4	unexecutable action_Find_the_link_key_from_local_or_remote_device
5	unexecutable action_Change_address_of_local_device
6	Passed
7	unexecutable action_Find_the_link_key_from_local_or_remote_device
8	Passed
9	unexecutable action_Using_OEM_CAN_database
10	unexecutable action action_Using_reverse_engineering
11	unexecutable action_Change_address_of_local_device
12	unexecutable action_Find_the_link_key_from_local_or_remote_device
13	unexecutable action_Change_address_of_local_device
14	unexecutable action_Change_address_of_local_device
15	unexecutable action_Find_the_link_key_from_local_or_remote_device

messages into the CAN bus also changes the stream of CAN bus signals that would normally be expected in vehicles. This violates the security property of integrity (in which no unauthorised modification should be allowed). Protecting against this could involve the addition of gateways in the SUT, which could either filter out floods of messages (by defining thresholds for the number of these messages that could be sent through at any given time). Alternatively, such messages (unless from an authorised source) could be disallowed completely.

The other test cases (all involving permutations of finding a link key, and changing the address) were not scripted because they required manual intervention. The former because it would need a remote device set to enable logging on the Host Controller Interface (not always possible) or to manually acquire data from a vehicle to find where the link key has been stored (which would have required hardware removal). The latter is automatable (for example, using a tool called *Spooftooph* [6]), however, either hardware removal or social manipulation is involved to find knowledge of an address that is already stored on the vehicle.

Other branches that were unscripted involves reverse engineering CAN messages to inject, which involves manual trial and error due to the sheer volume and variety of messages that are on the CAN bus at any one time. Using an OEM CAN database would enable automation, but availability is often non-existent due to commercial confidentiality. The branch that ended with successful test cases all involved using a legitimate device. That is, a device that was under our control, which we could use to test weaknesses in the vehicular implementation.

$TC(3) = action\_Determine\_pairing\_status \rightarrow$
$action\_Connect\_to\_serial\_port \rightarrow action\_Flood\_with\_set\_OBD\_messages \rightarrow$
$attack\_succeed \rightarrow Stop$
$TC(6) = action\_Determine\_pairing\_status \rightarrow$
$action\_Connect\_to\_serial\_port \rightarrow action\_Run\_through\_standard \rightarrow$
$action\_Run\_through\_non\_standard \rightarrow attack\_succeed \rightarrow Stop$
$TC(8) = action\_Determine\_pairing\_status \rightarrow$
$action\_Connect\_to\_serial\_port \rightarrow action\_Using\_passive\_monitoring \rightarrow$
$action\_Send\_flood\_with\_\_CAN\_messages \rightarrow attack\_succeed \rightarrow Stop$

Fig. 3. Test cases that succeeded

7 Conclusion and Future Work

We have demonstrated the translation of an informal attack tree into a formal structure using the process algebra CSP and proved equivalence. We use this tree to generate test cases automatically, and assign executions to scriptable test cases. We execute the test cases on a real-world vehicle (although this could be substituted with a testbed, with input from an OEM to reflect a real architecture, without the cost or risk to a test vehicle [9]). Thus, the full testing process is one step further to automation, and furthermore, the formal model of the attack tree could also be used for formal verification should the specifications of the system-under-test be available. Limitations are around how a tree is created (still largely manual) and certain actions within the attack tree requiring manual intervention. The aim is for the entire process (at an abstract level) to resemble Fig. 1. The work with testbeds (as in work done by [9]) as well as manual testing could continue to assist in further refinement of the models created.

References

1. Argus Cybersecurity: Argus Cyber Security Working with Bosch to Promote Public Safety and Mitigate Car Hacking (2017). http://bit.ly/2tNBLsm
2. Cheah, M., Bryans, J., Fowler, D.S., Shaikh, S.A.: Threat intelligence for bluetooth-enabled systems with automotive applications: an empirical study. In: Proceedings of the 47th IEEE/IFIP Dependable Systems and Networks Workshops: Security and Safety in Vehicles (SSIV). IEEE, Denver, June 2017
3. Cheah, M., Shaikh, S., Haas, O., Ruddle, A.: Towards a systematic security evaluation of the automotive bluetooth interface. J. Veh. Commun. **9**(7), 8–18 (2017)
4. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of 20th USENIX Security Symposium, pp. 77–92. USENIX Association, San Francisco, August 2011
5. Cho, K.T., Shin, K.G.: Error handling of in-vehicle networks makes them vulnerable. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security, pp. 1044–1055. ACM, New York, October 2016

6. Dunning, J.: SpoofTooph (2012). http://bit.ly/2tiOx5O
7. ELM Electronics: ELM Electronics: OBD. http://bit.ly/2s0yZPZ
8. Felderer, M., Zech, P., Breu, R., Buchler, M., Pretschner, A.: Model-based security testing: a taxonomy and systematic classification. Softw. Test. Verif. Reliab. **26**(2), 119–148 (2015)
9. Fowler, D.S., Cheah, M., Shaikh, S.A., Bryans, J.: Towards a testbed for automotive cyberecurity. In: Process of the 10th International Conference on Software Testing, Verification and Validation: Industry Track. IEEE, Tokyo, March 2017
10. Greenberg, A.: Hackers remotely kill a jeep on the highway-with me in it (2015). http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
11. Hoare, C.: Communicating Sequential Processes, Electronic edn. Prentice Hall International, Upper Saddle River (1985)
12. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks - practical examples and selected short-term countermeasures. Reliab. Eng. Syst. Saf. **96**(1), 11–25 (2011)
13. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 339–353. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_23
14. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental security analysis of a modern automobile. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 447–462. IEEE, Oakland, May 2010
15. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_17
16. Nogueira, S., Sampaio, A., Mota, A.: Test generation from state based use case models. Form. Asp. Comput. **26**(3), 441–490 (2014)
17. Oka, D.K., Furue, T., Langenhop, L., Nishimura, T.: Survey of vehicle IoT bluetooth devices. In: 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, pp. 260–264. IEEE, Matsue, November 2014
18. Pretschner, A., Broy, M., Kruger, I.H., Stauner, T.: Software engineering for automotive systems: a roadmap. In: Proceedings of the FOSE'07 2007 Future of Software Engineering, pp. 55–71. IEEE, Minneapolis, May 2007
19. Robert Bosch GmbH: CAN Specification Version 2.0 (1991). http://esd.cs.ucr.edu/webres/can20.pdf
20. Roscoe, A.: Understanding Concurrent Systems, 1st edn. Springer, London (2010). https://doi.org/10.1007/978-1-84882-258-0
21. Ruddle, A., Ward, D., Weyl, B., Idrees, S., Roudier, Y., Friedewald, M., Leimbach, T., Fuchs, A., Gurgens, S., Henniger, O., Rieke, R., Ritsscher, M., Broberg, H., Apvrille, L., Pacalet, R., Pedroza, G.: EVITA project: deliverable D2.3 - security requirements for automotive on-board networks based on dark-side scenarios. Technical report (2009). http://www.evita-project.org/Deliverables/EVITAD2.3.pdf
22. SAE International: SAE J1979 E/E Diagnostic Test Modes (2014). http://standards.sae.org/j1979_201408/
23. SAE International: J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems (2016). http://standards.sae.org/j3061_201601/
24. Salfer, M., Schweppe, H., Eckert, C.: Efficient attack forest construction for automotive on-board networks. In: Chow, S.S., Camenisch, J., Hui, L.C., Yiu, S.M. (eds.) 17th International Conference (ISC) on Information Security, October 2014

25. Schneier, B.: Attack trees: modeling security threats (1999). http://www.schneier.com/paper-attacktrees-ddj-ft.html
26. University of Oxford: FDR3 - The CSP Refinement Checker. https://www.cs.ox.ac.uk/projects/fdr/
27. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **22**(5), 297–312 (2012)
28. Vigo, R., Nielson, F., Nielson, H.R.: Automated generation of attack trees. In: Proceedings of the 27th Computer Security Foundations Symposium. IEEE, Vienna (2014)

Examination of a New Defense Mechanism: Honeywords

Ziya Alper Genç[1(✉)], Süleyman Kardaş[2], and Mehmet Sabir Kiraz[3]

[1] University of Luxembourg, Luxembourg City, Luxembourg
ziya.genc@uni.lu
[2] Batman University, Batman, Turkey
[3] TÜBİTAK BİLGEM, Gebze, Turkey

Abstract. Past experiences show us that password breach is still one of the main methods of attackers to obtain personal or sensitive user data. Basically, assuming they have access to list of hashed passwords, they apply guessing attacks, *i.e.,* attempt to guess a password by trying a large number of possibilities. We certainly need to change our way of thinking and use a novel and creative approach in order to protect our passwords. In fact, there are already novel attempts to provide password protection. The Honeywords system of Juels and Rivest is one of them which provides a detection mechanism for password breaches. Roughly speaking, they propose a method for password-based authentication systems where fake passwords, *i.e.,* "honeywords" are added into a password file, in order to detect impersonation. Their solution includes an auxiliary secure server called "honeychecker" which can distinguish a user's real password among her honeywords and immediately sets off an alarm whenever a honeyword is used. However, they also pointed out that their system needs to be improved in various ways by highlighting some open problems. In this paper, after revisiting the security of their proposal, we specifically focus on and aim to solve a highlighted open problem, *i.e.,* active attacks where the adversary modifies the code running on either the login server or the honeychecker.

Keywords: Passwords · Cracking · Honeywords · Code modification

1 Introduction

Password based authentication is a widely used technique throughout the Internet due to its simplicity and efficiency. However, this mechanism brings the potential risk of user credentials' being stolen in a server compromise event. In fact, there have been many incidents that confirm the significance of this threat where an adversary were able to obtain the database, which contains the usernames and the corresponding password hashes [1,5,7]. Once the breach occurs, it is at the mercy of authentication authority to disclose the details of the incident unless the adversary publishes the credentials.

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 130–139, 2018.
https://doi.org/10.1007/978-3-319-93524-9_8

Authentication systems employ cryptographic measures to protect the user credentials, however, many users have tendency to choose weak passwords *i.e.,* common words that can be easily guessed by a dictionary attack [3,4]. Due to the advancements in GPU technology, the hash value of a common password (*i.e.,* a word in a dictionary) can be cracked efficiently. These kinds of attacks may allow user credentials to be obtained by an adversary. Existing servers are capable of blocking any illegitimate login attempt when the authentication servers employ additional security mechanisms (*e.g.,* SMS that is used for 2 factor authentication) [10,11]. Even though such multi-factor authentication solutions improve the security against any illegitimate login attempt, these solutions do not provide any detection of password breaches.

In order to detect whether the password file has been stolen or not, Juels and Rivest [6] proposed the use of "honeywords", that is, a set of fake passwords that are mixed with the user's real password and the hash values of these passwords (real password and honeywords) are stored in the password file. Suppose that this file is compromised and all hash values in the file are cracked, the adversary still does not know which one is the real password. Note that the user or the adversary sends LS identity and password in order to request login. Then, LS checks whether a submitted password is among a user's honeywords but even when this check succeeds, LS needs to consult another secure component, HC, to know whether the index of the retrieved honeyword is that corresponding to the user's real password. HC alerts the administrator otherwise, since having observed an honeyword signals that the password file might have been compromised.

Our Contributions. In this paper, we first examine the Honeywords system of Juels and Rivest [6] and propose a practical improvement to solve a highlighted open problem. We enhanced the security of the Honeywords protocol against active code modification attacks where the adversary is assumed to modify the code running on either the login server or the honeychecker.

Roadmap. The outline of this paper is as follows. Section 2 describes the brute-force and dictionary attacks on passwords and their connection with the Honeywords system. Section 3 gives a detailed overview of Honeywords scheme. In Sect. 4, we present our improvements against active adversaries. Security analysis of our enhanced model is given in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Offline Brute-Force and Dictionary Attacks

Brute-force password cracking is one of the most popular attack types related to passwords. In a typical scenario, first, the adversary steals the password hash file. Next, a set which contains only the presumed characters that appear in a password is created. Then the adversary creates a combination of characters from this set, computes its hash and compares the hash value with the password hash. This process continues until a match is found [2].

There exist several techniques which increase the success rate of attackers while performing a brute-force attack. As an example, Weir *et al.* [12] developed

a state of the art password cracking algorithm which uses probabilistic, context-free grammars. Kelley *et al.* [8] showed that using Weir's attack, one billion guess is enough to crack % 40.3 of the passwords that comply with the "basic8" policy, *i.e.,* a password must have at least 8 characters. In the meantime, parallel processing capabilities of GPUs have been increased dramatically. For example, using `hashcat`[1], an open source password recovery software, cracking speed of hashes has reached 101.3×10^9 passwords per second for SHA1 on a single high computing cluster[2] which is commercially available [9].

Brute-force attacks sometimes can be applied efficiently depending on the password policy of the system. Assume that a user creates a password consisting of letters in the English alphabet which complies with the basic8 policy. The required time to span the password space of this password can be computed as follows:

$$\text{Time} = \frac{\text{Password space length}}{\text{Cracking speed}}$$

Considering the worst case and applying the above formula, we find that using the previously mentioned cluster, an adversary can crack the SHA-1 hash of that password in

$$\frac{(26)^8 \ \text{passwords}}{101.3 \times 10^9 \ \text{passwords per second}} \approx 2.06\,\text{s}.$$

Offline dictionary attack is similar to brute-force, with one difference. In this type of attack, an adversary computes the hash of words (possibly with salt) from a list that consists of strings which are typically derived from a dictionary. The adversary compares these hashes with the password hashes. The intention is to try dictionary words (which are more likely to be a user's password) rather than sequences of random characters. Most users unfortunately do not choose strong passwords for the sake of easy memorization. On the contrary, they choose weak passwords that are simple concatenations of dictionary words, common names, birthdays, city/street names, or easily guessable phrases.

3 Review of Honeywords System

Juels and Rivest proposed a novel authentication scheme called Honeywords system [6]. The central idea behind the Honeywords system is to change the structure of the password storage in such a way that each user is associated with a password and a set of fake passwords. The fake passwords are called honeywords. The union of all honeywords and the password is called sweetwords. As soon as a honeyword is submitted during the login process, it is automatically detected that the password database has been stolen. Hence, unlike conventional systems, honeywords based solutions can easily detect password database breaches.

The Honeywords system works as follows. As in the many conventional systems, users choose a *username* and a *password* during the registration phase.

[1] https://hashcat.net/hashcat/.
[2] https://gist.github.com/epixoip, Retrieved on June 22, 2017.

User ID	Username	Hashes
1	u_1	$H(sw_{1,1}), \cdots, H(sw_{1,n})$
2	u_2	$H(sw_{2,1}), \cdots, H(sw_{2,n})$
$\cdots$	$\cdots$	$\cdots$
m	u_m	$H(sw_{m,1}), \cdots, H(sw_{m,n})$

Fig. 1. Credentials database of a LS in the Honeywords system

User ID	Password Index
1	c_1
2	c_2
$\cdots$	$\cdots$
m	c_m

Fig. 2. Data stored on a HC

Next, the Login Server (LS) generates honeywords for the password and creates a record in credentials database. In each record, the ordering of the sweetwords is randomly chosen by the LS. Furthermore, LS sends the corresponding user ID and the index of the real password to Honeychecker (HC), which is an auxiliary server designed to store the index of the password. Let u_i and $H()$ denote the username of user i and the hash function used in the system, respectively. $H(sw_{i,j})$ denotes the hash of j^{th} sweetword of user i. A typical example of credentials table is demonstrated in Fig. 1.

HC stores the user IDs and the index of the passwords among the honeywords. Neither username nor password itself is submitted to HC during the authentication. Moreover, HC is designed as an *hardened* server which can only be accessed by LS. A typical structure of the data stored in HC is demonstrated in Fig. 2.

Note that HC accepts only two types of messages: Check and Set.

- Check(i, j) means to confirm whether $j = c_i$. If $j = c_i$, HC returns True, otherwise it returns False and triggers the alarm.
- Set(i, j) means to set $c_i = j$.

During the authentication phase, user submits her username and password. LS tries to find the corresponding record for that username in the credentials database. If a record exists, LS computes the hash of submitted password and tries to find a match in the hashes of sweetwords. If there is no match, then the submitted password is wrong and the access is denied. If there is a match, LS sends the corresponding user ID and the matching index to HC. Next, HC finds the record which corresponds to the user ID and compares the received index value with the one stored in its database. If the result is true, then the access is granted. Otherwise, the HC returns false and follows the system policy, *e.g.,* creates an alert and notifies the administrators. Authentication phase of the Honeywords system is depicted in Fig. 3.

The Honeywords system is originally designed with the assumption that the adversary can steal the hashed passwords and can invert the hashes to obtain the passwords. Also, it is assumed that the adversary cannot compromise both LS and HC in the same time period. Under this assumption, the Honeywords system protect passwords against brute-force and dictionary attacks described in Sect. 2. The Honeywords system aims at detecting password database breaches and helps deterring only offline dictionary attacks where it is assumed that the

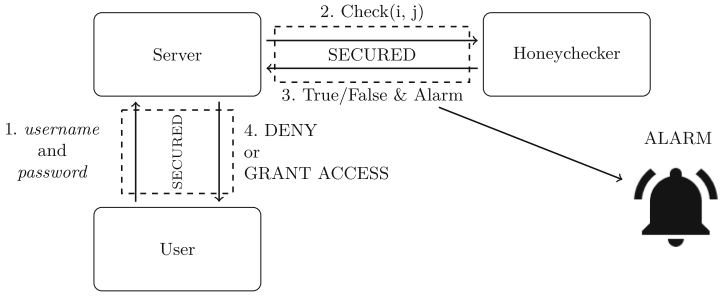

Fig. 3. Login scheme of a system using honeywords

adversary has stolen the password hashes and left the system. As also pointed out by Juels and Rivest, there are multiple open problems to solve in order to withstand active attacks.

4 Our Proposed Solutions

In this section, we focus on the following open problem which is highlighted in the original paper [6]:

> How can a honeyword system best be designed to withstand active attacks, e.g., malicious messages issued by a compromised computer system or code modification of the computer system (or the honeychecker)?

For this scenario, we deal with the active attacks in which the adversary makes code modifications on the Honeywords system where the adversary basically executes a malicious code on LS and set the index of the password to a new value that corresponds to a recovered honeyword. In the lights of these circumstances, the Honeywords system needs to be improved in order to withstand these advanced types of attacks.

4.1 Defending Against Malicious Code Modifications

An adversary may gain privileges to modify the running code on components of the Honeywords system. We only consider the cases where an adversary corrupts the component of LS that performs Set and Check commands. In that case, the parameters of these commands can be altered by the adversary. Similarly, HC can also be corrupted and send maliciously modified responses to LS. To mitigate these attacks, a reliable auditing mechanism is needed to check and verify the correctness of LS and HC. We classify the attack scenarios into two cases as follows.

Preventing Malicious Modifications on Set and Check Commands: A malicious adversary may target on modifying Set and Check commands that are run on the HC in order to gain more advantages for her attacks. He could call these command from the corrupted LS. In this context, in order to detect a malicious activity of a corrupted or legal LS, HC can verify whether the user requests a password change. If confirmed, HC will process Set command. Otherwise, the request would not be valid which detects the malicious activity by LS. In order to validate the origin of the Set request, HC may use some helper data. Following the design principles of the original protocol and keep the amount of data at HC at a minimum level it is possible to ask a security question or send a validation code to the mobile phone of the user. We follow the latter approach since it is in wide spread use and a practical way of adding another factor of authentication. The system roughly works as follows: during the registration, LS asks the user to enter the registration information including mobile number. The mobile number will be stored by HC. In order to accomplish this task, we overload the Set function as follows:

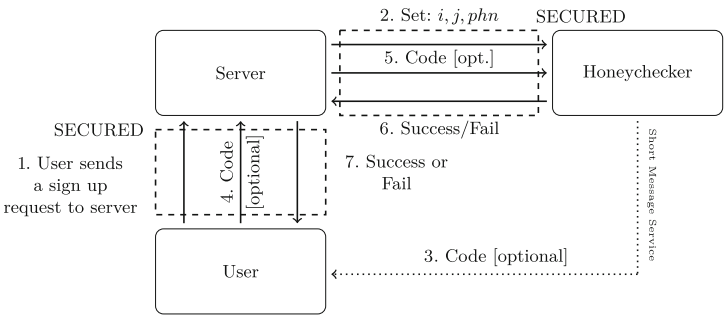

Fig. 4. Sign up scheme to mitigate against malicious code modification of LS.

- $\mathsf{Set}(i, j)$
- $\mathsf{Set}(i, j, phn)$

where phn denotes the phone number. Note that $\mathsf{Set}(i, j)$ function is the same as the Set function in the original Honeywords system. $\mathsf{Set}(i, j, phn)$ is invoked whenever a user registers to the system. $\mathsf{Set}(i, j)$ is invoked whenever a user changes her password. While the login procedure depicted in Fig. 3 did not change, the sign up of our enhanced Honeywords system is depicted in Fig. 4.

If the password change request is received, LS will generate new honeywords and randomly permute the sweetwords. Next, LS will send $\mathsf{Set}(i, c_i)$ to HC. HC generates a one time pad *nonce* and using *phn* sends it to user via SMS. The user submits *nonce* to LS which forwards it to HC for validation. The password change scheme is depicted in Fig. 5.

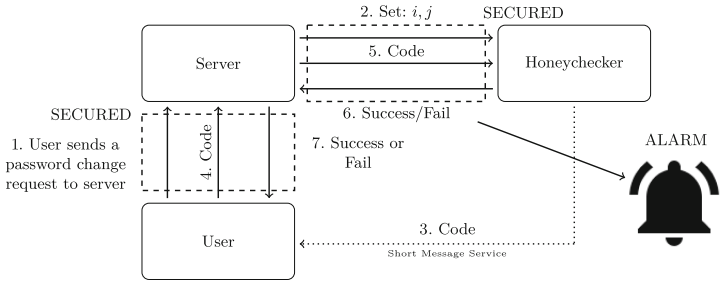

Fig. 5. Password Change scheme to mitigate against malicious code modification of HC.

Preventing Malicious Code Modification on HC: In this scenario, an adversary can modify HC and can send illegitimate responses to LS. In order to verify the correctness of HC, we propose an efficient probabilistic method which audits HC periodically. More concretely, there will be built-in user accounts in LS whose passwords will be known to LS. Since LS knows the result of these Check messages by itself, it can easily verify them with the ones coming from HC. More formally, let b be the user ID of a built-in account and c_b be the index of the password which is known to LS. A scheduled service at LS takes the following actions to test the correctness of HC.

Algorithm 1. Test Correctness of Honeychecker

```
1: function IsCorrupted(bID, c_bID)    ▷ bID: built-in user ID, c_bID: built-in index
2:     response ← false
3:     isCorrupt ← true
4:     response ← Check(bID, c_bID)
5:     if response = true then:
6:         isCorrupt = false
7:     return isCorrupt
```

This test will be repeated periodically for all built-in accounts to increase the probability of detection. For simplicity, let the probability of receiving a legitimate response from a compromised HC be fixed (p). After t number of tests, the probability of detecting a malicious activity P is computed as

$$P = 1 - \prod_{i=1}^{t} p_i$$

where t denotes the number of tests and p_i denotes the probability of receiving an legitimate response from the malicious adversary and $\forall i, p_i = p$.

If HC responds uniformly at random, then the equality yields

$$P = 1 - \prod_{i}^{t} p = 1 - \prod_{i}^{t} \frac{1}{n} = 1 - \frac{1}{n^t}$$

where n denotes the number of honeywords per user. Given $n = 20$ as suggested by [6], only two test is enough to achieve the correctness check with probability $P > 0.997$.

5 Security Analysis

In this section, we analyze the security of our enhanced system. We follow the case by case approach to perform the security analysis. We begin with the case that the adversary has compromised the LS.

Theorem 1. *Under the assumption that the honeywords are indistinguishable from passwords, if the LS is compromised, then the enhanced system depicted in Fig. 4 detects illegitimate login attempts and the incident of password breaches with probability* $1 - \frac{1}{n}$ *where* n *is the number of sweetwords.*

Proof. Assume that the adversary compromised the LS and obtained the user IDs and sweetwords, *i.e.,* $sw_1, sw_2, \ldots, sw_n$. Since the index of the password, c_i is stored securely in the HC, the adversary will try to guess the password among honeywords. There are $n-1$ honeywords per user and the probability of guessing the password is $\frac{1}{n}$. In other words, the adversary will fail and the system will detect the credentials breach with probability $1 - \frac{1}{n}$. □

Next we continue with the case that the adversary steals the information from HC.

Theorem 2. *Our enhanced system does not disclose any information about the passwords if an adversary steals only HC database.*

Proof. Assume that an adversary steals HC database. According to the design principles of the Honeywords system LS does not send any information about passwords to HC. Therefore, the adversary cannot not obtain any information about the passwords as only i, c_i and *phn* records are stored in HC. □

Next, we prove the security of our enhanced system in the case that the adversary modifies the code running on LS. We assume that any abnormal deviation from the protocol would be noticed by the system administrators (*e.g.,* by auditing network logs), and therefore, adversary would have to restrict itself to modify existing functions for attacking the system. Thus, we consider abuse of only existing functions of LS, *i.e.,* Set and Check commands.

Theorem 3. *Our enhanced system does not disclose the passwords with probability* $1 - \frac{1}{n}$ *where* n *is the number of sweetwords if an adversary maliciously modifies the running code on* LS*. Also, the probability of unauthorized password change is negligible.*

Proof. Assume that an adversary maliciously modifies the running code on LS. A corrupted LS would send only Set or Check messages to HC (other abnormal behaviors would be detected by the administrators). However, a malicious Check message would be detected and the alarm would be set off by HC with probability $1 - \frac{1}{k}$ where k is the number of honeywords. Similarly, in the case that adversary sends malicious Set messages, HC will ask a validation code which is sent to the user's mobile phone. However, the probability of sending a valid combination without possessing the mobile phone is negligible. Thus, the adversary will also fail to send a malicious Set request. □

Finally, we analyze the case that the adversary modifies the code running on HC.

Theorem 4. *Our enhanced system does not disclose any information about the passwords if an adversary maliciously modifies the running code on* HC*.*

Proof. Consider that the adversary modifies the code running on HC. The TestHoneychecker routine will check the correctness of the Check messages and detect malicious responses with probability $P = 1 - \prod_{i=1}^{t} p_i$, where t denotes the number of tests and p_i denotes the probability of receiving an illegitimate response. The HC does not contain any useful information that would help to adversary developing a meaningful attack strategy. Hence, if the HC maliciously responds to the Check messages, LS will detect with probability P. □

Hence, our scheme has all the security properties of the primitive system designed in [6], plus it is more robust to code modification attacks.

6 Conclusion

Juels and Rivest [6] propose an interesting defense mechanism under a common attack scenario where an adversary steals the file of password hashes and inverts most or many of the hashes. The Honeywords system provides a powerful defense against this attack. Namely, even if the adversary recovers all of the hashes in the password file, he cannot try to login to the system without a high risk of being detected. On the other hand, the original Honeywords system is not a complete solution for the password management problem. The scenarios in which an adversary modifies running code on LS or HC is left as open problem. In this work, we review the original Honeywords system and focused on solving that problem. Namely, we enhanced the Honeywords system through adding additional security checks. Our additions are inexpensive and practical, and can be easily integrated into the primitive scheme. Moreover, we discussed the security of our enhanced protocol and showed that it is robust against code modification attacks.

Acknowledgments. Ziya Alper Genç's research is supported by a partnership between SnT/University of Luxembourg and pEp Security S.A. Mehmet Sabır Kiraz's work is supported by a grant from Ministry of Development of Turkey provided to the Cloud Computing and Big Data Research Lab Project (project ID: 2014K121030).

References

1. Burgess, M.: How to Check if Your Linkedin Account was Hacked, May 2016. http://www.wired.co.uk/article/linkedin-data-breach-find-out-included
2. Conklin, A., Dietrich, G., Walz, D.: Password-based authentication: a system perspective. In: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS 2004) - Track 7, HICSS 2004 - vol. 7, pp. 701–702. IEEE Computer Society, Washington, DC (2004)
3. Florencio, D., Herley, C.: A large-scale study of web password habits. In: Proceedings of the 16th International Conference on the World Wide Web. Association for Computing Machinery Inc. (2007)
4. Furnell, S., Dowland, P., Illingworth, H., Reynolds, P.: Authentication and supervision: a survey of user attitudes. Comput. Secur. **19**, 529–539 (2000)
5. Gallagher, S.: Yahoo Admits It's been Hacked Again, and 1 Billion Accounts were Exposed, December 2016. https://arstechnica.com/security/2016/12/yahoo-reveals-1-billion-more-accounts-exposed-and-some-code-may-have-been-stolen/
6. Juels, A., Rivest, R.L.: Honeywords: making password-cracking detectable. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS 2013, vol. 38, pp. 145–160. ACM (2013)
7. Keane, J.: Security Researcher Dumps 427 Million Hacked Myspace Passwords Online, July 2016. https://www.digitaltrends.com/social-media/myspace-hack-password-dump/
8. Kelley, P.G., Komanduri, S., Mazurek, M.L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., Lopez, J.: Guess again (and again and again): measuring password strength by simulating password-cracking algorithms. In: IEEE Symposium on Security and Privacy, pp. 523–537 (2012)
9. Sagitta: Brutalis - GPU Compute Nodes. https://sagitta.pw/hardware/gpu-compute-nodes/brutalis/
10. Wang, D., Wang, P.: Two birds with one stone: two-factor authentication with security beyond conventional bound. IEEE Trans. Depend. Secur. Comput. (2017). https://ieeexplore.ieee.org/document/7558124/
11. Wang, D., Gu, Q., Cheng, H., Wang, P.: The request for better measurement: a comparative evaluation of two-factor authentication schemes. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS 2016, pp. 475–486. ACM, New York (2016)
12. Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, SP 2009, pp. 391–405. IEEE Computer Society, Washington, DC (2009)

AndroNeo: Hardening Android Malware Sandboxes by Predicting Evasion Heuristics

Yonas Leguesse(✉), Mark Vella, and Joshua Ellul

University of Malta, Msida 2080, Malta
{yonas.leguesse.05,mark.vella,joshua.ellul}@um.edu.mt

Abstract. Sophisticated Android malware families often implement techniques aimed at avoiding detection. Split personality malware for example, behaves benignly when it detects that it is running on an analysis environment such as a malware sandbox, and maliciously when running on a real user's device. These kind of techniques are problematic for malware analysts, often rendering them unable to detect or understand the malicious behaviour. This is where sandbox hardening comes into play. In our work, we exploit sandbox detecting heuristic prediction to predict and automatically generate bytecode patches, in order to disable the malware's ability to detect a malware sandbox. Through the development of AndroNeo, we demonstrate the feasibility of our approach by showing that the heuristic prediction basis is a solid starting point to build upon, and demonstrating that when heuristic prediction is followed by bytecode patch generation, split personality can be defeated.

Keywords: Android · Malware sandbox hardening
Sandbox evasion heuristics · Bytecode patching

1 Introduction

Android powers most mobile devices, and has recently surpassed Windows to become the Internet's most used operating system [1]. AntiVirus company McAfee reported [2] that in Q4 (2016) they witnessed a 72% increase of unique mobile malware samples collected in Q3, with over 2.4 million detections in Q4 alone. These staggering numbers are rendering automated malware analysis tools essential for analysts. A popular approach for automated malware analysis involves the use of malware analysis sandboxes, where the analyst sets up an environment in which a malware sample can run whilst its relevant operations and behaviour is collected for analysis. Neuner et al. [3] provided an interesting comparison of available Android malware sandboxes, where system emulation plays a central role in the provision of a safe inspection environment. Sophisticated malware often use techniques that allow them to detect and thwart the sandbox's analysis. One popular approach is commonly referred to as split-personality malware [4], where the generic evasion approach is as follows:

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 140–152, 2018.
https://doi.org/10.1007/978-3-319-93524-9_9

```
if(isSandbox()){
    System.exit();
} else{
    continueMaliciousOperations();
}
```

This code pattern allows malware to behave in two different manners, i.e. executing `System.exit()` or `continueMaliciousOperations()`, depending on whether the environment is a sandbox or not. The distinguishing factors are established within `isSandbox()`, which is where the sandbox detection techniques are implemented. These checks can come in the form of a simple one-line verification or more complex checks such as performance analysis. Further still, malware can shift malicious operations to event handlers that are only likely to be triggered on real devices (e.g. SMS received events).

In this work, through the development of AndroNeo, we demonstrate the ability to automatically generate emulator based sandbox detection techniques, whilst providing the sandbox with the means to avert and disable the malware's split personality capabilities. In other words, we managed to predict the checks being made within `isSandbox()`, and ensure that the heuristic checks fail even when running in a sandboxed environment. This ensures that the behaviour of `continueMaliciousOperations()` is exposed to the sandbox probes. Moreover, we demonstrate that Jing's [5] heuristic prediction basis is a solid starting point for our work. We analysed a set of a number of real malware samples, exposing the presence of the identified heuristic data. The results demonstrated that numerous instances of the malware samples exhibit the use of the discovered heuristic, indicating the potential use of some sort of emulator or sandbox detection. We also developed a prototype implementation that takes advantage of data collected throughout the heuristic prediction process in order to automatically generate sandbox hardening capabilities. Our evaluation demonstrates that our proposed technique is able to predict and disable sandbox detecting capabilities without prior knowledge of the employed heuristic checks.

2 Background and Related Work

2.1 Sandbox Detection

The use of emulation underpins sandbox construction (e.g. Ananas [6] and Mobile-sandbox [7]), since it easily provides isolation and efficient system state restoration. However, emulation provides an easy target for malware to evade sandbox analysis through sandbox detection. For example, Jing et al. [5] developed a tool (Morpheus) that is able to automatically generate heuristics that can detect Android emulators. Pestas et al. [8] outlined three categories of evasion techniques based on static properties, dynamic sensor information, and VM-related intricacies of the Android emulator that can be used for sandbox detection. Maier et al. [9] developed a tool called Sand-Finger that uses a fingerprinting approach which gathers information on several sandboxes and uses

this information to identify which particular sandbox is being used. Vidas and Christin [10] categorized four classes of sandbox detection techniques based on differences in behaviour, performance, hardware components, and software components. These sandbox detecting techniques are problematic for malware analysts, since they expose potential weaknesses in their sandboxes, and that is where sandbox hardening comes into play.

2.2 Sandbox Hardening

Hardened sandboxes are ones that are not so easily bypassed. Gajrani et al. [11] took this threat into account, and set out to develop techniques that help malware analysts build a hardened sandbox analysis environment by identifying commonly used sandbox detecting techniques and patching them by applying emulator modifications, system image modifications, and by applying runtime hooks. Another sandbox hardening approach involves the use of bare-metal devices which drops the need of emulation altogether. The bare-metal approach was applied by Mutti et al. with the tool BareDroid [12]. Kirat et al. [13] mention how there is a constant tension between the quality, stealthiness, and efficiency of a malware sandbox. Trying to improve one of the factors often results in compromises in the remaining two. A bare-metal approach does bring about many negative efficiency implications in terms of cost and time. For example, in the experimentation of BareDroid, the authors mention that a full restore which is required after every analysis takes 141 s, thus causing a scalability issue. On the other hand, system restore time on an emulator is almost negligible. They also mentioned that the quality of the analysis is often adversely affected since probes that rely on system emulation, such as taint analysis on native code, cannot be used. Moreover, the use of bare-metal sandboxes can create new sandbox detecting heuristics when attempting to address these issues. It is evident that because of the advantages provided by system emulation, bare-metal is not a fix-all approach, and emulator based sandboxes will not be fully replaced by bare-metal ones. In our work, we will be focusing on hardening those that are still based on emulation.

Sandbox hardening can be applied to different components of a sandbox. One approach involves the modification of emulator properties [11]. Certain properties are easy to modify, however others are not modifiable out of the box and require hardware emulation tweaking. The modifications are sensitive in nature since any misconfiguration could easily corrupt the system. Besides modifying the emulator, one may also choose to directly modify the application's bytecode. By modifying the sandbox detecting parts of the code directly in the application, it is possible to force the application into failing sandbox detecting checks, even when running in a sandbox. Another hardening technique involves patching the Android Framework. By modifying the Android Framework APIs, one can control the results of the class method invocations, thus manipulating the sandbox detection verification checks. Even though this approach can be very effective, the downside is that it would require custom modifications for every Android OS version.

3 AndroNeo

AndroNeo builds upon an evasion heuristic generation technique, Morpheus [5]. While Morpheus focuses on the automatic generation of sandbox detecting heuristics, AndroNeo aims to provide automated sandbox hardening capabilities. The proposed technique involves utilising the generated heuristics, as well as properties obtained throughout the generation process, in order to produce sandbox hardening patches for the malware's Dalvik bytecode.

The notion of a distinguisher is a central component: A **Distinguisher** refers to a distinctive characteristic that can be used to classify an environment as a sandbox or an actual device. These distinguishers can fall under one of two categories: Sandbox Profile Distinguishers, or Device Profile Distinguishers. **Sandbox Profile Distinguishers** refer to a set of distinguishers that contain properties found in most sandboxes environments. In other words, if the system on which the malware is being analysed contains a property within the Sandbox Profile, then it is likely that it is in fact a sandbox. On the other hand **Device Profile Distinguishers** refer to a set of distinguishers that contain properties found in most mobile devices. In other words, if the system on which the malware is being analysed contains a property within the Device Profile, then it is likely that it is an actual device and not an emulator. As Fig. 1 depicts, the distinguisher generation stage is split in two phases. The first phase starts off with a recon task that operates upon samples of real devices and sandboxes in order to generate recon datasets that contain candidate distinguishers. The second step of this phase produces the device and sandbox profiles as characterized by the computed profile distinguishers that identify them as such. The resulting distinguisher profiles are used during a second stage to automatically harden the sandbox. The distinguisher profiles themselves provide the required information to locate the patch points and to generate the code patches that deactivate evasion. The following sections present the individual steps in detail.

3.1 Reconnaissance

Let $\mathbf{S}$ be the set of n sample sandboxes, and $\mathbf{D}$ be the set of m sample devices, $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$ and $\mathbf{D} = \{d_1, d_2, \ldots, d_m\}$. Let $\mathbf{X}$ represent the set of all sample sandboxes and devices, i.e. $\mathbf{X} = \mathbf{S} \cup \mathbf{D}$.

The reconnaissance (or recon) phase involves the extraction of data from the sandboxes and devices, that can potentially be used to identify sandboxes. The recon works by parsing all of the Android API classes [14] in each element in $\mathbf{S}$ and $\mathbf{D}$, and invoking all of their available methods and reading of class constants. Every recon execution on a sandbox or device produces what we refer to as a recon dataset. Let $\mathbf{M}_x$ be the set of class constants and methods of all Android API classes accessible from sandbox or device x: $\mathbf{M}_x = \{m | m \in (ApiMethodCalls \cup ApiClassConstants), \exists x \in \mathbf{X}\}$, e.g.: $\mathbf{M}_x = \{getDeviceId(), \ldots, getLineNumber(), Build.DEVICE, \ldots, Build.SERIAL\}$. The function $v(m, x)$ denotes the value returned when calling/reading m within the context of $x \in \mathbf{X}$, e.g.: $v(\texttt{getDeviceID()}, s_1) =$

"000000000000000". The set $\mathbf{R}_x$ represents the resulting set of key value pairs obtained by invoking all methods and constants available for the device or sandbox x i.e.: $\mathbf{R}_x = \{(m, v(m, x)), \forall m \in \mathbf{M}_x, \exists x \in \mathbf{X}\}$ e.g.: $\mathbf{R}_{s_1} = \{$(`getDeviceId()`, `"000000000000000"`), ..., (`getLineNumber()`, `"15555215554"`)$\}$. The set $\mathbf{R_X}$ represents the set of all recons from the set of devices and/or sandboxes in $\mathbf{X}$. i.e.: $\mathbf{R_X} = \{\mathbf{R}_x | \forall x \in \mathbf{X}\}$.

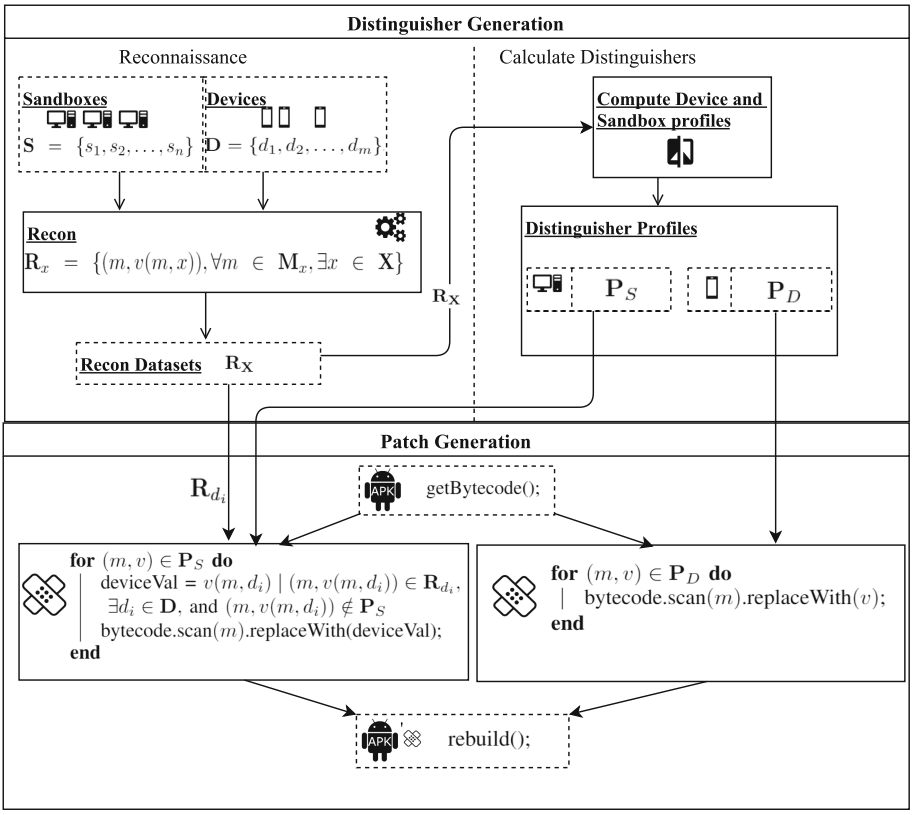

Fig. 1. AndroNeo

3.2 Calculating Distinguishers

Let $r = (m, v(m, x)) \in \mathbf{R}_x$ where $m \in \mathbf{M}_x$ and $x \in \mathbf{X}$. We are interested in the ratios $\frac{|r_\mathbf{S}|}{|\mathbf{S}|}$ and $\frac{|r_\mathbf{D}|}{|\mathbf{D}|}$ where $|r_\mathbf{S}|$ and $|r_\mathbf{D}|$ represent the number of times r is present in $\mathbf{S}$ and $\mathbf{D}$ respectively. For example, if `getDeviceId()` returns `"000000000000000"` for every sandbox in our set of sample sandboxes then we are interested in the ratio $\frac{|(getDeviceId(), \text{“000000000000000”})_\mathbf{S}|}{|\mathbf{S}|} = 1$. This ratio tells us that all sandboxes returned the value of `"000000000000000"`, and therefore

points towards a property that could be used to identify a sandbox, i.e. a sandbox profile distinguisher.

On the other hand, if for example the value of the class constant `Build.TAGS` is `"release-keys"` for every device in our set of sample devices then we are interested in the ratio $\frac{|(Build.TAGS, \text{“}release-keys\text{”})_{\mathbf{D}}|}{|\mathbf{D}|} = 1$. This ratio tells us that all devices returned the value of `"release-keys"`, and therefore points towards a property that could be used to identify a device, i.e. a device profile distinguisher.

Sandbox Profile. The set $\mathbf{P}_S$ represents all sandbox profile distinguishers: $\mathbf{P}_S = \{(m, v) \mid (m, v) = (m, v(m, s_i)) \in \mathbf{R}_{s_i}, \frac{|r_{\mathbf{S}}|}{|\mathbf{S}|} > \tau > \frac{|r_{\mathbf{D}}|}{|\mathbf{D}|}, \exists s_i \in \mathbf{S}\}$. The elements in $\mathbf{P}_S$ provide us with a list of properties that can be used to identify a sandbox. Let us assume that we have a device or sandbox x_0, and we want to determine whether or not x_0 is a sandbox. For every $(m, v(m, x_0)) \in \mathbf{R}_{x_0}$ if $(m, v(m, x_0)) \in \mathbf{P}_S$ then this indicates that x_0 is a sandbox.

Device Profile. The set $\mathbf{P}_D$ represents all device profile distinguishers: $\mathbf{P}_D = \{(m, v) \mid (m, v) = (m, v(m, d_j)) \in \mathbf{R}_{d_j}, \frac{|r_{\mathbf{D}}|}{|\mathbf{D}|} > \tau > \frac{|r_{\mathbf{S}}|}{|\mathbf{S}|}, \exists d_j \in \mathbf{D}\}$. The elements in $\mathbf{P}_D$ provide us with a list of properties that can be used to identify a sandbox, or rather the lack of a device. Let us assume that we have a device or sandbox x_0, and we want to determine whether or not x_0 is a sandbox. Then for every $(m, v(m, x_0)) \in \mathbf{R}_{x_0}$ if $(m, v(m, x_0)) \notin \mathbf{P}_D$ then this indicates that x_0 is not a device, and therefore a sandbox. The Sandbox and Device profiles correspond to what Morpheus [5] refers to as S-pool and D-pool respectively. Additionally, our profiles make use of a tunable threshold (τ), and retain the obtained Device Dataset values for the patch generation phase.

3.3 Patch Generation

Let s_0 be the sandbox that requires hardening. Since s_0 is in fact a sandbox, then there is a good chance that for every $(m, v) \in \mathbf{P}_S$, it is also the case that $(m, v(m, s_0)) \in \mathbf{P}_S$. These occurrences can provide malware with sandbox detecting capabilities. In order to hide the presence of these values, we will re-use data that was collected during the recon phase. For every $(m, v) \in \mathbf{P}_S$, we need to identify a corresponding $(m, v(m, d_i)) \in \mathbf{R}_{d_i}$, where $d_i \in \mathbf{D}$ and $(m, v(m, d_i)) \notin \mathbf{P}_S$. The bytecode can then be modified to ensure that for every $(m, v) \in \mathbf{P}_S$, $v(m, s_0)$ returns $v(m, d_i)$, thus emulating the value of a real device.

In the case of device profile distinguishers since s_0 is a sandbox, then there is a good chance that for every $(m, v) \in \mathbf{P}_D$, $(m, v(m, s_0)) \notin \mathbf{P}_D$. This time, the bytecode needs to be modified to in such a way that for every $(m, v) \in \mathbf{P}_D$, $v(m, s_0)$ returns v, thus emulating the value of a real device (since this time v indicates a value that is commonly found on devices). The following process is used to patch the malware's bytecode:

Input: sandbox detecting malware apk
Output: patched malware apk′
bytecode = apk.getBytecode();
for $(m, v) \in \mathbf{P}_S$ **do**
  deviceVal = $v(m, d_i) \mid (m, v(m, d_i)) \in \mathbf{R}_{d_i}$,
   $\exists d_i \in \mathbf{D}$, and $(m, v(m, d_i)) \notin \mathbf{P}_S$
  bytecode.scan(m).replaceWith(deviceVal);
end
for $(m, v) \in \mathbf{P}_D$ **do**
  bytecode.scan(m).replaceWith(v);
end
return apk.rebuild() -> apk′;

4 Experimentation

A number of experiments were carried out in order to evaluate the capabilities of automatically generated sandbox detection heuristics, whilst demonstrating the effectiveness of the hardening process just presented.

4.1 Experiment Setup

We developed an Android application that allowed us to collect data by invoking all possible Android API class methods and constant values through the use of the Reflection API. Apktool was used to decode and re-compile the applications, whilst a number of bash scripts allowed us to patch the smali code generated through the apktool, according to the identified distinguishers. The environment in which the experimentation was conducted, consisted of a number of Android sandboxes and emulators (Sanddroid [15], NVISO ApkScan [16], Droidbox [17], Android 7.0 Emulator, Android 6.0 Emulator), and a set of Android devices (Samsung Galaxy S4, Nexus 5x, Nexus 5, Nexus 6P, OnePlus X). Moreover, we used a set of 7160 real malware samples from VirusShare [18]. In our case studies, we made use of the popular DroidBox sandbox, which is also the underlying dynamic analysis tool of several Android sandboxes [3].

4.2 Distinguisher Profiles

In order to validate the relevance of the identified profile distinguishers and test their potential in identifying evasion checks, we cross-checked our findings against the malware samples. Table 1 enlists the results of the top 10 sandbox profile distinguishers. The results were calculated by identifying the number of malware samples that contained instances of $(m, v) \in \mathbf{P}_S$, where both m and v are found in the same class. For example, the results show that out of the 7160 malware samples, we found 2395 applications that invoked the method `getDeviceId()`. The invocation of this method on its own is not necessarily suspicious, however when we see that 794 of these applications also looked for the string `"000000000000000"` within the same class, then this fact increases the

likelihood that this call is made for emulator detection purposes. The numbers in Table 1, are somewhat conservative since they only represent cases where both instances of the invocation (e.g.: `getDeviceId()`) and the corresponding distinguisher value (e.g.: `"000000000000000"`) were found to be in the same class. There may very well be a few additional cases where the distinguisher values are defined in a class different to the distinguisher method/field invocation. Moreover, there is also the possibility that the distinguisher values are encrypted or hashed, and are therefore not identified during the crosscheck against the malware samples. Nevertheless, when it comes to the actual patching, these cases will still be patched using the bytecode modification approach, since it is the field or method call (e.g.: `getDeviceId()`) that is being modified. These results show that it is very likely that the automatically generated heuristics are being used to detect sandboxes or emulators by malware families in the wild, providing validity to Jing's [5] assumptions.

Table 1. Sandbox profile distinguishers in malware samples

Method/field	Count	Value	Count
`Build.MODEL`	2478	`sdk`	2352
`getDeviceId`	2395	`000000000000000`	794
`getNetworkOperatorName`	1522	`Android`	665
`Build.DEVICE`	1150	`generic`	594
`Build.BOARD`	992	`unknown`	561
`Build.MANUFACTURER`	1404	`unknown`	255
`Build.CPU_ABI`	306	`x86`	95
`getSubscriberId`	1830	`310260000000000`	53
`getSimOperatorName`	432	`Android`	23
`Build.TAGS`	378	`test-keys`	17

4.3 Case Studies

In order to verify that the detected profile distinguishers contain actual evasion checks, and that the proposed patch generation step effectively deactivates them, we chose two representative samples and conducted a more in-depth investigation. We chose the samples on the basis that they form part of two widespread malware families, Crosate and Pincer, as well as the availability of thorough documentation [19] of their behaviour. This information provides the ground truth with which to compare the results obtained by AndroNeo. Through Droidbox, we proceeded with analyzing both the original and patched samples, with the resulting behaviour observed in both instances being compared to the ground truth. We generated a bytecode hardening class, containing the data necessary for AndroNeo to spoof the return values of the identified distinguishers

(i.e. `bytecode.scan`(m)`.replaceWith`(v) step in Sect. 3 C). The code below is a snippet from the hardening class.

```
# Field Declarations
const-string v0, "release-keys"
sput-object v0, Lharden/Harden;->FIELD18:Ljava/lang/String;
....
# Method Declarations
.method public static method14()Ljava/lang/String;
....
   const-string v0, "353627074120224"
   return-object v0
.end method
```

Crosate. Crosate is a bot with the ability to steal SMSs, call logs, contact information, send SMS, record a call, and makes a phone call. However, when executing in DroidBox, it terminates itself, thus hiding all bot to Command and Control (CNC) communication. The listing below contains Crosate's code where the sandbox detection check is made:

```
public void onCreate() {
....
    String BotID = tm.getDeviceId();
....
if (BotID.indexOf("000000000000000") != -1) {
    System.exit(0);
  }
```

In the last three lines the application checks the value of `BotID`, which returns the value of `getDeviceId()` (i.e. the phone's IMEI). If it finds that the IMEI contains `"000000000000000"` then it calls `System.exit(0)`. The listing below contains the code of the patched version of Crosate:

```
public void onCreate() {
....
  String BotID =  AGHardening.method14();
....
  if (BotID.indexOf("000000000000000") != -1) {
    System.exit(0);
  }
```

Here one can see that again the code checks the value of `BotID` and compares it with `"000000000000000"`. However, this time `BotID` is not returning `getDeviceId()`, but is instead invoking `method14()` from our hardening class. As we saw earlier, `method14()` now returns `"353627074120224"` instead of the sandbox's IMEI. Therefore, in our patched version the check will fail and the `System.exit(0)` method will not be invoked, thus performing all of the malicious operations as it would on a actual device. The application also calls several other methods such as `getLineNumber1()` and `getNetworkOperatorName()`, which can also be used to identify a sandbox. All of these instances were replaced with

their corresponding device values found in the hardening class. Comparing the analysis report of the original Crosate sample and the hardened version exposed the difference in their behaviour. The original version showed very little activity, and the entire report only produced 21 log entries. On the other hand, the modified version produced 191 log entries, which clearly showed additional activities and services being started, as well as device data being exfiltrated. As confirmed by a review of the malware's code, the results demonstrated that the additional 170 log entries were generated by the services and activities that were only launched if the sandbox detecting code failed, and thus the `System.exit(0)` was not invoked. This also corresponded to the expected behaviour as documented by Nigam [19]. This demonstrates the successful automatic patching of the malware's sandbox detecting capabilities, exposing the malware's split personalities.

Pincer. Pincer is another bot that only communicates with the CNC server when it detects that the device is an actual device and not a sandbox. Therefore, our assumption was that we would only see network communication with the CNC once we patched the malware sample. The listing below outlines the several checks that Pincer invokes in order to distinguish between a sandbox and a device. Fortunately, our tool was able to predict these checks and patch them accordingly.

```
if ( C0024b.m108d(context).toLowerCase().equals("android")
  || C0024b.m106b(context).equals("000000000000000")
  || C0024b.m107c(context).equals("15555215554")
  || AGHardening.FIELD21.toLowerCase().equals("sdk")
  || AGHardening.FIELD21.toLowerCase().equals("generic")) {
    C0018a.m68a(context, true);
} else {C0014a.m50a(context, jSONObject, new C0023d());}
```

The listing below shows the methods from where the distinguishers are invoked, and how they were identified and modified.

```
public static String m106b(Context context) {
  return AGHardening.method14();
}
public static String m107c(Context context) {
  return AGHardening.method6();
}
public static String m108d(Context context) {
  return AGHardening.method8();
}
```

Therefore, all of the methods and constants involved in the sandbox detecting checks were replaced with our methods and fields in the hardening class, forcing them to return actual device values instead of the sandbox's values. Pincer was an interesting case study since initially both the patched and original versions seemed to return the same logs. The patching itself did not fail. However, the parts of the code where the split personality occur are only executed once a

command is received from the CNC. Only once we simulated the receipt of a CNC command could we observe the differences in behaviour. This issue falls outside the scope of AndroNeo, however it is included in the scope extension. Comparing the analysis report of the original Pincer sample and the hardened version exposed the difference in behaviour. The original version did not show any network communication corresponding to our simulated CNC commands. The modified version, on the other hand, reported the network connections and the IP addresses of the CNC, and responded to the commands that we requested. Moreover, the modified version also attempted to access image files on the SD card, a behaviour that was not present in the original version. Once again this case study clearly demonstrates the successful patching of the malware's sandbox detecting capabilities, exposing the malware's split personalities. The patched malware's behaviour corresponded to the expected behaviour as documented by Nigam [19].

It is important to note that in our tests the detection and patching of distinguishers was done in an automated manner. The identified distinguishers retrieved from our sample sandboxes and devices were able to predict and patch the sandbox detection techniques used by these malware samples without being explicitly told what to look for.

5 Scope Extension

5.1 Limitations

Even though the evaluation produced promising results, and the prototype successfully patched well known malware samples in an automated manner, one must bear in mind that heavily obfuscated malware within scope is not currently handled. There still exist evasion techniques that may not be identified and patched in the current implementation. For example, malware obfuscation through the use of native code, direct Binder IPC invocations, or malware packers will not be handled by the prototype. Another limitation is that currently our recon implementation only utilises Java reflection in order to invoke the methods and class fields and collect system information. Morpheus's artefact collector on the other hand implements additional techniques, such as the use of a directory walker that identifies the presence of emulator specific files and folders, in order to generate heuristics based on these artefacts.

5.2 Proposed Extensions

AndroNeo is planned to be extended subject to further experimentation in the following ways:

Improve Bytecode Patching. In its current state AndroNeo is not able to tackle self-modifying malware that implement techniques such as runtime class loading to hide its malicious code. To overcome this, AndroNeo would need to implement a form of dynamic patching.

Implement All of Morpheus' Artefact Retrievers. Extending AndroNeo to include other such sources of system information, such as the ones implemented by Morpheus, could generate additional heuristics.

Extend to All Sandbox Detecting Techniques. The current scope is limited to hardening sandbox detecting techniques based on static emulator properties. However, the proposed methodology can be evolved to tackle event-based, or user presence based techniques. Whether or not heuristic prediction can tackle these types of evasion techniques requires further investigation. Alternatively, an interesting avenue could involve merging our techniques with other techniques, such as the one proposed by Pooryousef and Amini [20], that are aimed at tackling the exposure of event-driven actions. Moreover, one could extend AndroNeo to build upon alternative heuristic generation techniques [21] in order to patch their corresponding generated heuristics.

6 Conclusion

Android malware families demonstrate the ability of detecting malware analysis sandboxes using detection heuristics. To tackle this problem we presented AndroNeo, a tool that automatically hardens malware analysis sandboxes to disable the malware's sandbox detection capabilities. AndroNeo identifies sandbox detecting capabilities within an Android application and alters and disables its functionality. Moreover, we presented a prototype implementation of AndroNeo demonstrating its capabilities on real malware families. AndroNeo would benefit from straightforward extensions in terms of complete reuse of Morpheus, and other extensions requiring further thought, such as the ability to patch event-based, or user presence based sandbox detecting techniques.

References

1. Techcrunch: Android overtakes windows as the internet's most used operating system. https://techcrunch.com/2017/04/03/statcounter-android-windows. Accessed 01 May 2017
2. Mcafee. 2016 mobile threat report. https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf. Accessed 02 May 2017
3. Neuner, S., van der Veen, V., Lindorfer, M., Huber, M., Merzdovnik, G., Mulazzani, M., Weippl, E.R.: Enter sandbox: Android sandbox comparison. CoRR, abs/1410.7749 (2014)
4. Maier, D., Protsenko, M., Muller, T.: A game of droid and mouse: the threat of split-personality malware on android. Comput. Secur. **54**, 2–15 (2015)
5. Jing, Y., Zhao, Z., Ahn, G.-J., Hu, H.: Morpheus: automatically generating heuristics to detect android emulators. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp. 216–225. ACM (2014)
6. Eder, T., Rodler, M., Vymazal, D., Zeilinger, M.: Ananas-a framework for analyzing android applications. In: 2013 Eighth International Conference on Availability, Reliability and Security (ARES), pp. 711–719. IEEE (2013)

7. Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J.: Mobile-sandbox: having a deeper look into android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1808–1815. ACM (2013)
8. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: hindering dynamic analysis of android malware. In: Proceedings of the Seventh European Workshop on System Security, p. 5. ACM (2014)
9. Maier, D., Muller, T., Protsenko, M.: Divide-and-conquer: why android malware cannot be stopped. In: 2014 Ninth International Conference on Availability, Reliability and Security (ARES), pp. 30–39. IEEE (2014)
10. Vidas, T., Christin, N.: Evading android runtime analysis via sandbox detection. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, pp. 447–458. ACM (2014)
11. Gajrani, J., Sarswat, J., Tripathi, M., Laxmi, V., Gaur, M.S., Conti, M.: A robust dynamic analysis system preventing sandbox detection by android malware. In: Proceedings of the 8th International Conference on Security of Information and Networks, pp. 290–295. ACM (2015)
12. Mutti, S., Fratantonio, Y., Bianchi, A., Invernizzi, L., Corbetta, J., Kirat, D., Kruegel, C., Vigna, G.: Baredroid: large-scale analysis of android apps on real devices. In: Proceedings of the 31st Annual Computer Security Applications Conference, pp. 71–80. ACM (2015)
13. Kirat, D., Vigna, G., Kruegel, C.: Barebox: efficient malware analysis on bare-metal. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 403–412. ACM (2011)
14. Anrdroid API classes. https://developer.android.com/reference/classes.html. Accessed 01 May 2017
15. Sandroid: Android malware sandbox. http://sanddroid.xjtu.edu.cn. Accessed 01 Mar 2017
16. Nviso apkscan, scan android applications for malware. https://apkscan.nviso.be/. Accessed 01 May 2017
17. Droidbox: Dynamic analysis of android applications. https://github.com/pjlantz/droidbox. Accessed 01 May 2017
18. Virusshare.com - because sharing is caring. https://virusshare.com/. Accessed 01 May 2017
19. Nigam, R.: A timeline of mobile botnets. Virus Bulletin, March 2015
20. Pooryousef, S., Amini, M.: Enhancing accuracy of android malware detection using intent instrumentation. In: Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1, ICISSP, pp. 380–388. INSTICC, ScitePress (2017)
21. Paleari, R., Martignoni, L., Roglia, G.F., Bruschi, D.: A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In: USENIX Workshop on Offensive Technologies (WOOT) (2009)

Protocols and Algorithms

A More Efficient 1–Checkable Secure Outsourcing Algorithm for Bilinear Maps

Öznur Kalkar[1(✉)], Mehmet Sabir Kiraz[1], İsa Sertkaya[1], and Osmanbey Uzunkol[2]

[1] Mathematical and Computational Sciences, TÜBİTAK BİLGEM, Kocaeli, Turkey
{oznur.arabaci,mehmet.kiraz,isa.sertkaya}@tubitak.gov.tr
[2] Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Hagen, Germany
osmanbey.uzunkol@gmail.com

Abstract. With the rapid advancements in innovative technologies like cloud computing, internet of things, and mobile computing, the paradigm to delegate the heavy computational tasks from trusted and resource-constrained devices to potentially untrusted and more powerful services has gained a lot of attention. Ensuring the verifiability of the outsourced computation along with the security and privacy requirements is an active research area. Several cryptographic protocols have been proposed by using pairing-based cryptographic techniques based on bilinear maps of suitable elliptic curves. However, the computational overhead of bilinear maps forms the most expensive part of those protocols. In this paper, we propose a new 1–checkable algorithm under the one-malicious version of a two-untrusted-program model. Our solution is approximately twice as efficient as the single comparably efficient 1–checkable solution in the literature, and requires only 4 elliptic curve point additions in the preimage and 6 field multiplications in the image of the bilinear map.

Keywords: Outsourcing computation · Bilinear maps · Security Privacy

1 Introduction

The flexibility of cloud computing comes with innovative and cost-effective solutions for both individuals and enterprises. The main advantages include ubiquitous and on-demand network access, using resources in a pay-per manner, scalability of the underlying services, location independence, and rapid elasticity [21]. Accordingly, the proposals to delegate the mostly expensive and sometimes energy consuming computations from trustful resource-constrained devices (e.g., sensors, RFID cards, SIM cards) to untrusted or even malicious external applications and services (e.g., cloudlets, large scale cloud service providers) have been increasing dramatically. The demand of weak clients to outsource costly computational tasks to external powerful services while preserving security and

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 155–164, 2018.
https://doi.org/10.1007/978-3-319-93524-9_10

privacy becomes more and more crucial in accordance with the interplay of cloud computing paradigm with the new advancements and novel solutions in internet of things (IoT) and mobile technologies. In order to assure security and privacy in such scenarios, the most crucial part of the underlying cryptographic techniques is not to get the private cryptographic values out of the secure memory. Therefore, one can only apply masking techniques in such a way that it can be outsourced to a (potentially) untrusted server to perform further computations. However, one also has to verify simultaneously that the computations are indeed correct without having an additional significant overhead. Hence, efficiently verifiable outsourcing to untrusted or even malicious services while ensuring the desired level of security and privacy has become a subject of many recent research activities, and forms a new challenge especially in the presence of malicious services [9,18] (Fig. 1).

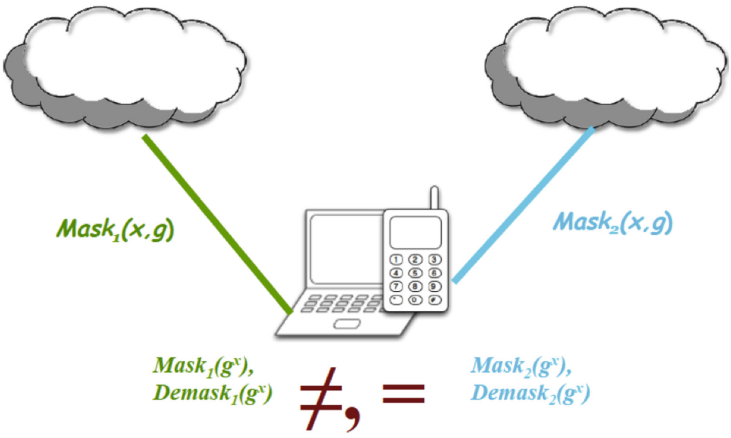

Fig. 1. A typical verifiable outsourcing mechanism with two servers

Many cryptographic applications realizing novel security and privacy solutions demand to use pairing-based primitives in order to overcome different challenges in the realm of cloud computing, internet of things and mobile technologies. Pairing-based cryptography is realized by using bilinear maps of carefully chosen elliptic curves [6,7,15]. Computing such bilinear maps is the most inefficient and expensive part of those solutions. Although, there are plenty of research results aiming to reduce the computational cost of the computation of bilinear maps [3,4,7,13,17], these computations are still far away of being feasible for resource-constrained environments. Even worse, new research results imply that the key sizes must be increased to achieve the desired security level [2].

1.1 Related Work

Hohenberger and Lysyanskaya gave the first formal security model for secure and verifiable outsourcing of cryptographic computations in the presence of

malicious powerful devices or services [14]. Subsequently, Chevallier-Mames *et al.* proposed the first cryptographic protocol for outsourcing the computation of bilinear maps [11]. However, this delegation process brought some additional challenges. Firstly, no sensitive informations has to be revealed to the untrusted parties. However, one needs to establish a mechanism in which the validity of the outsourced computation can effectively be verified by the client, since the external device can otherwise cheat the client by deliberately manipulating the desired result.

Chen *et al.* classified the security models based on the number of servers being involved in the outsourcing process [10]:

- One-Untrusted Program (OUP): There exists a single malicious server performing the delegated computation.
- One-Malicious version of a Two-Untrusted Program (OMTUP): There exist two untrusted servers performing the delegated computation but exactly one of them may also behave maliciously.
- Two-Untrusted Program (TUP): There exist two untrusted servers performing the delegated computation and both of them may simultaneously behave maliciously, but they do not maliciously collude.

Tian *et al.* improves the checkability result of the work of Chen *et al.* [10] with some additional computations under the TUP assumption in [25]. Both studies have two main phases; offline computation and online computation. In the offline phase, the client prepares some precomputed values. The online phase consists of masking the sensitive values and using the precomputed values from the offline phase to prepare the necessary requests. Both algorithms suffer from being only probabilistically checkable implementations (both are 1/2–checkable implementations). However, as outlined by Canard *et al.* [8], an outsourcing algorithm not ensuring sufficient checkability property could lead to severe security issues especially in the case of authentication protocols. Hence, it is highly desirable to have an efficient outsourcing mechanism having 1–checkable property, i.e. potentially malicious external services can never cheat the clients since the verification of the outsourced computation works with probabilty 1.

Chevallier-Mames *et al.* proposed the first 1–checkable outsourcing mechanism for bilinear maps in the OUP setting [11] which subsequently was improved by Kang *et al.* [16] and Canard *et al.* [8]. These mechanisms suffer unfortunately from being inefficient since directly embedding the computation of bilinear maps inside the trusted devices are more efficient than utilizing their outsourcing mechanism.

Chen *et al.* used two servers under the OMTUP assumption with an offline phase to propose a 1/2–checkable outsourcing mechanism using 5 point additions in the preimage of the bilinear map and 4 multiplications in the image of the bilinear map. A similar 1/2–checkable result was later proposed by Tian *et al.* [25] improving the online phase with 4 point additions in the preimage of the bilinear map and 3 multiplications in the image of the bilinear map. Subsequently, Lin *et al.* also proposed another 1/2–checkable outsourcing mechanism improving the tuple-sizes and offline phase of the previous mechanisms [19].

Recently, Luo *et al.* proposed the first efficient 1–checkable outsourcing mechanism for bilinear maps under the OMTUP assumption requiring only 8 point additions in the preimage of the bilinear map and 6 multiplications in the image of the bilinear map [20].

1.2 Our Contributions

In this paper, we propose a new and highly efficient outsourcing mechanism for bilinear maps following the steps of Chen *et al.* and Tian *et al.*'s work. The main advantage our mechanism is to provide 1–checkability property which ensures that external servers never can cheat the users. Our 1–checkable outsourcing mechanism is given under the OMTUP assumption for bilinear maps. Compared to the only efficient solution of Luo *et al.* [20], our mechanism is approximately twice more efficient requiring only 4 point additions in the preimage of the bilinear map and 6 multiplications in the image of the bilinear map. Furthermore, our mechanism is almost as efficient as existing 1/2–checkable solutions. In particular, our mechanism only needs 3 more multiplications in the image of the bilinear map in order to achieve a 1–checkable outsourcing algorithm.

2 Security Model

We follow the security model proposed by Hohenberger and Lysyanskaya [14], and we use one-malicious version of a two-untrusted program (OMTUP) model. More concretely, there are two untrusted cloud servers in this model performing the outsourced computation where only one of them is assumed to be malicious.

Definition 1. *A pair of algorithms* (T, U_1, U_2) *are an* α*-efficient implementation of an algorithm* Alg *if* (1) *they are an outsource-secure implementation of* Alg*, and* (2) $\forall$ *inputs* x*, the running time of* T *is* $\leq$ *an* α*-multiplicative factor of the running time of* $\mathsf{Alg}(x)$*.*

Definition 2. *A pair of algorithms* (T, U_1, U_2) *are an* β*-checkable implementation of an algorithm* Alg *if* (1) *they are an outsource-secure implementation of* Alg*, and* (2) $\forall$ *inputs* x*, if* $U_i', i = 1, 2$ *deviates from its advertised functionality during the execution of* $T^{(U_1', U_2')}(x), T$ *will detect the error with probability* $\geq \beta$*.*

Definition 3. *A pair of algorithms* (T, U_1, U_2) *are an* (α, β)*-outsource secure implementation of an algorithm* Alg *if they are both* α*-efficient and* β*-checkable.*

3 Verifiable Secure Outsourcing of Bilinear Maps

3.1 Bilinear Maps

Definition 4. *Let* $(\mathbb{G}_1, +)$ *and* $(\mathbb{G}_2, +)$ *be two additive cyclic groups of order* q *and* $(\mathbb{G}_3, \cdot)$ *be a multiplicative cyclic group of order* q*, where* q *is a prime number. Let also* P_1 *and* P_2 *be generators of* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*, respectively. Assume that Discrete Logarithm Problem (DLP) is hard in both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*. A bilinear map is a map* $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ *satisfying the following properties* [5,12,24]:

- ***Bilinearity:*** *For all* $P, P' \in \mathbb{G}_1, Q, Q' \in \mathbb{G}_2$, *e is a group homomorphism in each component, i.e.*
  1. $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$,
  2. $e(P, Q + Q') = e(P, Q) \cdot e(P, Q')$.
- ***Non-degeneracy:*** *e is non-degenerate in each component, i.e.,*
  1. *For all* $P \in \mathbb{G}_1^*$, *there is an element* $Q \in \mathbb{G}_2$ *such that* $e(P, Q) \neq 1$,
  2. *For all* $Q \in \mathbb{G}_2^*$, *there is an element* $P \in \mathbb{G}_1$ *such that* $e(P, Q) \neq 1$.
- ***Computability:*** *There exists an algorithm which computes the bilinear map e efficiently.*

3.2 Precomputations

In order to speed up the algorithms, some computations are required to be performed offline. For this, we will use BPV+ method proposed by Wang et al. [26] and applied by Tian et al. [25]. BPV+ uses two tables; a static table ST, and a dynamic table DT. Values in the static table are used to construct the values in the dynamic table. After each invocation of $Rand()$, values in the dynamic table needs to be refreshed in an idle time of the device. Dynamic table consists of the following tuple:

$$(\alpha P_1, xP_1, yP_1, \alpha P_1 - xP_1, \alpha P_1 - yP_1, mP_2, nP_2, \beta P_2, e(P_1, P_2)^{\alpha\beta}).$$

Rand. We now describe the precomputation algorithm called $Rand()$. At each invocation of $Rand()$, a tuple is returned and removed from the dynamic table. Then, at some convenient time, a new tuple is created from the values in the static table and stored in the dynamic table. Below, we describe the creation process of static and dynamic tables.

- `Preprocessing`:
  1. Generate n random elements $\alpha_1, \cdots, \alpha_n \in \mathbb{Z}_q$.
  2. For $j = 1, \cdots, n$ compute $\beta_{j_1} = \alpha_j P_1$, $_{j_2} = \alpha_j P_2$.
  3. Compute $e(P_1, P_2)$.
  4. Store the values of α_j, β_{j_1}, β_{j_2} and, $e(P_1, P_2)$ in the static table ST.
- `Point generation`:
  1. Randomly generate $S \subset \{1, \cdots, n\}$ such that $|S| = k$.
  2. For each $j \in S$, randomly select $K_j \in \{1, \cdots, h-1\}$, where $h > 1$ is a small integer.
  3. Compute
  $$\alpha = \sum_{j \in S} \alpha_j K_j \bmod q.$$
  If $\alpha = 0 \bmod q$, start again. Otherwise, compute
  $$\alpha P_1 \equiv \sum_{j \in S} K_j \cdot \beta_{j_1} \bmod q.$$

4. Following the above procedure, compute similarly the elements $xP_1, yP_1, mP_2, nP_2, \beta P_2$.
5. Compute $\alpha P_1 - xP_1, \alpha P_1 - yP_1$.
6. Calculate $e(P_1, P_2)^{\alpha\beta}$.
7. Store $(\alpha P_1, xP_1, yP_1, \alpha P_1 - xP_1, \alpha P_1 - yP_1, mP_2, nP_2, \beta P_2, e(P_1, P_2)^{\alpha\beta})$ in the dynamic table DT.

3.3 Our Algorithm: `OutPair`

Let T denote the trusted device with limited computational power which wants to outsource the pairing computation, $U_i(M, N) \rightarrow e(M, N)$, $i \in \{1, 2\}$ denotes untrusted party U_i taking (M, N) as input and returning $e(M, N)$.

Our algorithm `OutPair` takes $A \in \mathbb{G}_1, B \in \mathbb{G}_2$ as inputs and produces $e(A, B)$ as output.

1. T runs $Rand()$ in order to get the following values:

$$(\alpha P_1, xP_1, yP_1, \alpha P_1 - xP_1, \alpha P_1 - yP_1, mP_2, nP_2, \beta P_2, e(\alpha P_1, \beta P_2)).$$

2. T computes

$$A - \alpha P_1, B - nP_2, B - mP_2, B - \beta P_2,$$

and queries U_1 in random order as follows:

- $U_1(A - \alpha P_1, mP_2) \rightarrow A_{11}$,
- $U_1(A - \alpha P_1, B - nP_2) \rightarrow A_{12}$,
- $U_1(\alpha P_1 - xP_1, B - \beta P_2) \rightarrow A_{13}$,
- $U_1(yP_1, B - \beta P_2) \rightarrow A_{14}$.

Then similarly, T queries U_2 in random order as follows:

- $U_2(A - \alpha P_1, nP_2) \rightarrow A_{21}$,
- $U_2(A - \alpha P_1, B - mP_2) \rightarrow A_{22}$,
- $U_2(\alpha P_1 - yP_1, B - \beta P_2) \rightarrow A_{23}$,
- $U_2(xP_1, B - \beta P_2) \rightarrow A_{24}$.

3. Upon receiving computation results from both servers, T checks if
   - $A_{11}A_{22} \stackrel{?}{=} A_{21}A_{12}$,
   - $A_{13}A_{24} \stackrel{?}{=} A_{23}A_{14}$.

   If the check is not successful, then T outputs an "error". Otherwise, T computes and outputs

$$A_{11}A_{22}A_{13}A_{24}e(P_1, P_2)^{\alpha\beta} \ .$$

3.4 Security Analysis

Theorem 1. *Under the OMTUP assumption, the algorithms* $(T, (U_1, U_2))$ *are an outsource-secure 1–checkable implementation of* `OutPair` *for calculating the bilinear map* $e(A, B) = OutPair(A, B)$*, where the input* $(A \in G_1, B \in G_2)$ *may be honest secret, honest protected, or adversarial protected.*

Proof. The correctness of $(T, (U_1, U_2))$ is straight forward. If T wants to evaluate $e(A, B)$, at the last step of `OutPair` T needs to compute $D_1 D_2 e(P_1, P_2)^{\alpha\beta}$. Taking D_1, D_2, α, β as defined in OutPair:

$$\begin{aligned} D_1 D_2 e(P_1, P_2)^{\alpha\beta} &= A_{11} A_{22} A_{13} A_{24} e(P_1, P_2)^{\alpha\beta} \\ &= e(A - \alpha P_1, mP_2) e(A - \alpha P_1, B - mP_2) e(\alpha P_1 - xP_1, B - \beta P_2) \\ &\quad e(xP_1, B - \beta P_2) e(P_1, P_2)^{\alpha\beta} \\ &= e(A - \alpha P_1, B) e(\alpha P_1, B - \beta P_2) e(P_1, P_2)^{\alpha\beta} \\ &= e(A, B) e(\alpha P_1, -\beta P_2) e(P_1, P_2)^{\alpha\beta} \\ &= e(A, B). \end{aligned}$$

Next, we prove the security of the algorithm as follows. First of all, let (E, U_1', U_2') be a probabilistic polynomial-time (PPT) adversary that interacts with a PPT algorithm T in the OMTUP model, and S_1, S_2 be the simulators in Pair One, Pair Two, respectively.

– **Pair One:** We below show that $EVIEW^i_{real} \sim EVIEW_{ideal}$ to ensure that the external adversary E learns nothing.
If the input (A, B) is honest protected, or adversarial protected, then the simulator S_1 behaves the same way as in the real execution. There is no secret input, so S_1 does not require to access secret inputs. Hence, suppose that (A, B) is an hones secret input. Then the simulator S_1 behaves as follows: Upon receiving the input in round i, S_1 ignores it, selects a series of random points of the form $((a - a_1)P_1, b_1P_2, (b - b_2)P_2, (a_1 - a_2)P_1, b_3P_2, a_3P_1, b_2P_2, (b - b_1)P_2, (a_1 - a_3)P_1, a_2P_1)$, and makes the following queries to U_1' and U_2'.
  - $U_1((a - a_1)P_1, b_1P_2) \to A_{11}$,
  - $U_1((a - a_1)P_1, (b - b_2)P_2) \to A_{12}$,
  - $U_1((a_1 - a_2)P_1, b_3P_2) \to A_{13}$,
  - $U_1(a_3P_1, b_3P_2) \to A_{14}$.
  - $U_2((a - a_1)P_1, b_2P_2) \to A_{21}$,
  - $U_2((a - a_1)P_1, (b - b_1)P_2) \to A_{22}$,
  - $U_2((a_1 - a_3)P_1, b_3P_2) \to A_{23}$,
  - $U_2(a_2P_1, b_3P_2) \to A_{24}$.

  If $A_{11}A_{22} = A_{21}A_{12}$ and $A_{13}A_{24} = A_{23}A_{14}$, S_1 sets $Y_p^i = \emptyset, Y_u^i = \emptyset, rep^i = 0$; otherwise S_1 sets $Y_p^i = error, Y_u^i = \emptyset, rep^i = 1$. For both cases, S_1 saves the appropriate states. In the real and ideal experiments, the input distributions are computationally indistinguishable for U_1' and U_2'. In the ideal experiment, inputs are chosen uniformly random. In the real experiment, inputs that are generated by *Rand* are re-randomized. In the $i-$th round, there are two possibilities:

- U_1' and U_2' performs honestly: S_1 gives the correct output which is the same as the output of $(T, (U_1, U_2))$.
- One of U_1' and U_2' gives an incorrect output: In the ideal experiment, S_1 detects the mistake and result in "error". In the real experiment, T does the same thing.

Thus, $EVIEW^i_{real} \sim EVIEW^i_{ideal}$ and by the hybrid argument, we conclude that $EVIEW_{real} \sim EVIEW_{ideal}$.

– **Pair Two:** We now show that $UVIEW_{real} \sim UVIEW_{ideal}$ which ensures that the untrusted software (U_1, U_2) learns no information.
Upon receiving an input on round i, S_2 behaves exactly like S_1, generates random inputs and saves the appropriate states. Also, In the ith round of the real experiment, T re-randomizes the inputs to (U_1', U_2'). Hence, for each round i, we have $UVIEW^i_{real} \sim UVIEW^i_{ideal}$. Hence, by the hybrid argument, we conclude that $UVIEW_{real} \sim UVIEW_{ideal}$. □

4 Comparison

For the comparison table in Fig. 2, we use the following abbreviations:

– **SM**: Scalar multiplication in $\mathbb{G}_1, \mathbb{G}_2$
– **ME**: Modular exponentiation in $\mathbb{G}_3$
– **PA**: Point addition in $\mathbb{G}_1, \mathbb{G}_2$
– **PC**: Pairing computation
– **FM**: Field multiplication in $\mathbb{G}_3$

Also, k is the size of the set used at step 1 of point generation, and h is the size of the set used at step 2 of point generation. For a more detailed explanation of k, h we refer to [22].

	Chen[10]	**Tian**[25]	**AKSU**[1]	**Lin**[19]	**Luo**[20]	**Ren**[23]	`OutPair`
Checkability							
	1/2	1/2	1/2	1/2	1	1	1
Client's Workload							
PA	5	4	4	4	8	8	4
FM	4	3	3	3	6	14	6
Servers's Workload							
PC	8	6	4	6	4	6	8
ME	0	0	0	0	0	4	0
Precomputations							
SM	9	3	2	2	0	8	0
ME	0	2	2	2	1	0	1
PA	0	5(k+h-3)	4(k+h-3)	4(k+h-3)	2(k+h-3) + 2	0	6(k+h-3) + 2
PC	3	0	0	0	0	6	0

Fig. 2. Comparison of `OutPair` with the state-of-the-art schemes

As one can see from the comparisons, we achieve the 1–checkability property by reducing the client's workload in the online phase at the expense of adding small computational overhead to the offline phase.

5 Conclusion

In this paper, we propose a highly efficient outsourcing algorithm for bilinear maps with 1–checkability property. Our 1–checkable outsourcing mechanism basically ensures that external servers never can cheat the honest clients. Compared to the state-of-the-art, our mechanism is approximately twice more efficient requiring only 4 elliptic curve point additions in the preimage of the bilinear map and 6 field multiplications in the image of the bilinear map. Furthermore, our mechanism is almost as efficient as existing 1/2–checkable solutions. In particular, our mechanism only needs 3 more multiplications in the image of the bilinear map. For future work, it is interesting and highly desirable to propose such efficient algorithms in the presence of only one untrusted server, i.e. in the One-Trusted Program (OUP) assumption.

References

1. Arabacı, O., Kiraz, M.S., Sertkaya, I., Uzunkol, O.: More efficient secure outsourcing methods for bilinear maps. Cryptology ePrint Archive, Report 2015/960 (2015). http://eprint.iacr.org/2015/960
2. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Cryptology ePrint Archive, Report 2017/334 (2017). http://eprint.iacr.org/2017/334
3. Barreto, P., Galbraith, S., Higeartaigh, C., Scott, M.: Efficient pairing computation on supersingular abelian varieties. Des. Codes Cryptogr. **42**(3), 239–271 (2007). https://doi.org/10.1007/s10623-006-9033-6
4. Beuchat, J.-L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17455-1_2
5. Blake, I., Seroussi, G., Smart, N.: Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series. Cambridge University Press, New York (2005)
6. 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
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptol. **17**(4), 297–319 (2004). https://doi.org/10.1007/s00145-004-0314-9
8. Canard, S., Devigne, J., Sanders, O.: Delegating a pairing can be both secure and efficient. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 549–565. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07536-5_32
9. Chen, X.: Introduction to secure outsourcing computation. Synth. Lect. Inf. Secur. Priv. Trust **8**(2), 1–93 (2016)

10. Chen, X., Susilo, W., Li, J., Wong, D., Ma, J., Tang, S., Tang, Q.: Efficient algorithms for secure outsourcing of bilinear pairings. Theor. Comput. Sci. **562**, 112–121 (2015). http://dblp.uni-trier.de/db/journals/tcs/tcs562.html#ChenSLWMTT15
11. Chevallier-Mames, B., Coron, J.S., McCullagh, N., Naccache, D., Scott, M.: Secure delegation of elliptic-curve pairing. Cryptology ePrint Archive, Report 2005/150 (2005)
12. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
13. Hess, F., Smart, N., Vercauteren, F.: The eta pairing revisited. IEEE Trans. Inf. Theory **52**(10), 4595–4602 (2006)
14. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_15
15. Joux, A.: A one round protocol for tripartite Diffie-Hellman. J. Cryptol. **17**(4), 263–276 (2004). https://doi.org/10.1007/s00145-004-0312-y
16. Kang, B.G., Lee, M.S., Park, J.H.: Efficient delegation of pairing computation (2005)
17. Koblitz, N., Menezes, A.: Pairing-based cryptography at high security levels. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 13–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_2
18. Kumar, K., Liu, J., Lu, Y.H., Bhargava, B.: A survey of computation offloading for mobile systems. Mob. Netw. Appl. **18**(1), 129–140 (2013). https://doi.org/10.1007/s11036-012-0368-0
19. Lin, X.J., Qu, H., Zhang, X.: New efficient and flexible algorithms for secure outsourcing of bilinear pairings. Cryptology ePrint Archive, Report 2016/076 (2016). http://eprint.iacr.org/2016/076
20. Luo, Y., Fu, S., Huang, K., Wang, D., Xu, M.: Securely outsourcing of bilinear pairings with untrusted servers for cloud storage. In: Trustcom/BigDataSE/ISPA (2016)
21. Mell, P., Grance, T.: The NIST definition of cloud computing. NIST Special Publication, pp. 800–145 (2011)
22. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation (2000)
23. Ren, Y., Ding, N., Wang, T., Lu, H., Gu, D.: New algorithms for verifiable outsourcing of bilinear pairings. Sci. China Inf. Sci. **59**(9), 99103 (2016)
24. Shacham, H.: New paradigms in signature schemes. Ph.D. thesis, Stanford, CA, USA (2006)
25. Tian, H., Zhang, F., Ren, K.: Secure bilinear pairing outsourcing made more efficient and flexible. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS 2015, pp. 417–426. ACM, New York (2015). https://doi.org/10.1145/2714576.2714615
26. Wang, Y., Wu, Q., Wong, D.S., Qin, B., Chow, S.S.M., Liu, Z., Tan, X.: Securely outsourcing exponentiations with single untrusted program for cloud storage. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 326–343. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_19

A Selective Privacy-Preserving Identity Attributes Protocol for Electronic Coupons

Pau Conejero-Alberola(✉), M. Francisca Hinarejos, and Josep-Lluís Ferrer-Gomila

University of the Balearic Islands, Ctra. de Valldemossa, km 7,5, 07120 Palma, Spain
{pau.conejero,xisca.hinarejos,jlferrer}@uib.es

Abstract. Electronic coupons (e-coupons) are a very effective marketing tool. In some scenarios, it is necessary to check some customer's personal attributes at the redeeming phase (e.g. age, title, citizenship, etc.). But customers may be reluctant to use e-coupons if their privacy is in danger. Digital certificates and credentials could be suitable for validating customer attributes. However, a bad use of such electronic documents entails a loss of privacy, revealing more identity attributes than necessary. Here, we present the first secure protocol for e-coupons, achieving verification proofs of identity, with selective disclosure of customer's certified attributes. On the other hand, our proposal meets other necessary security requirements, such as forging protection and double-redeem protection.

Keywords: E-coupon · Security · Identity · Privacy

1 Introduction

A coupon is a usually small piece of printed paper that lets you get a service or product for free or at a lower price (Merriam-Webster definition). It is an effective marketing instrument [1], and quite used because merchants and customers are benefited. One one hand, merchants can increase loyalty of their customers or attract new customers. On the other hand, customers can achieve better prices or gifts. In this paper we deal with a type of e-coupon that is addressed to a group of customers that must meet certain identity requirements, for example, being in a certain age group, being resident in a specific country, etc. A real example of this type of promotions is found in a well-known chain of hamburgers [2], whose promotional bases indicate: customers must be over a certain age, resident in X, and have to purchase product Y.

Paper based coupons, redeemed face-to-face, allow the merchant to easily verify compliance with established requirements, using paper documents (such as an identity card). Such verification does not usually result in a high loss of privacy, because verification is instantaneous without the merchant registering private

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 165–176, 2018.
https://doi.org/10.1007/978-3-319-93524-9_11

information of the customers in their systems. But in the case of e-coupons, exchanges are made electronically, and therefore "verification documents" must also be in electronic form to be redeemed face-to-machine.

Digital certificates and credentials are suitable to provide authenticated information about customers. These documents are signed by a trusted third party, which assumes the responsibility of validating the data contained in them. But those documents contain more information than may be necessary in certain e-coupon scenarios, with the consequent risk of loss of privacy. As a result, many customers are reluctant to provide personal data, beyond the strictly necessary for the purpose they want to perform. Our goal is to maximize the privacy of customers, that is, only that information strictly necessary to validate compliance with the requirements of the e-coupon must be disclosed.

Contribution. We present the first e-coupon scheme with a selective and verified personal data disclosure mechanism, which provides a better degree of privacy, allowing the issuance of e-coupons for eligible customers. On the other hand, customer's compliance of requirements is verified during the redeem.

This paper is organized as follows. Section 2 reviews the related work. Section 3 defines the proposed scheme, the security requirements and the cryptographic background. Section 4 specifies in detail all the phases of the proposed protocol. Section 5 includes a brief security analysis, and finally, Sect. 6 lists the conclusions of this paper.

2 Related Work

In this section, we will review those most significant proposals that have shown concern about the authentication and privacy of customers.

As a first contribution of e-coupons, Kumar et al. [3] show that targeted e-coupons are intended for a group of customers who meet certain requirements, and they indicate that customers must be identified.

In contrast, Jakobsson et al. [4] state that an e-coupon system should not expose customer privacy more than other advertising system. They present a proposal where no attribute of the customer is verified.

Chang et al. [5] present a scheme with a registration phase, where the customer provides personal information to the issuer. Therefore, e-coupons are identified, but merchants do not receive this information from the customer.

Aigner et al. [6] explain two e-coupon schemes. One of them has an authentication process for the customer in front of the issuer and merchant.

Chang and Sun [7] explain a scheme where the customer and merchant must be registered at a trusted third party, indicating a mutual authentication.

Chang et al. [8] provide two e-coupon schemes, one for specific registered customers (with better discounts) and the other for non-specific customers.

Liu et al. [9] provide a proposal, in which customers remain anonymous if they are honest. Their scheme achieves traceability against dishonest customers.

As a conclusion, we can affirm that there is no previous solution that requires the authentication of some attributes of the customer, and that this authentication takes place without revealing other data related to that customer. That is,

the selective disclosure of attributes is a problem that has not been addressed so far in the area of e-coupons.

3 Scheme: Scenario and Security

In this section, we describe our proposal. First, we detail the scenario, the entities involved, their role and we outline the e-coupon structure. Then, the security requirements and the cryptographic background are defined.

3.1 Scenario

We define a custom environment, offering an online distribution marketing portal, using daily e-coupons promotions to be redeemed in manufacturer's branches for eligible and registered customers. The proposed scenario allows different grades of privacy using an *Idemix* [10] service, a selective method to disclose customer identity attributes. We consider the following entities: Trusted Third Party (TTP), Issuer, Merchant and Customer.

Trusted third party $\mathcal{T}$ is in charge of issuing the *Idemix* credentials based on a digital certificate, and carrying the system public parameters.

Issuer $\mathcal{I}$ is in charge of issuing and distributing e-coupons. In this scenario, $\mathcal{I}$ has to ensure that the credential has not been used previously for registering, and only one-time e-coupon is delivered per customer $\mathcal{C}$. A registered $\mathcal{C}$ can request the e-coupon policy from the portal, which defines the redemption requirements to compute the identity proof. If the proof is valid then $\mathcal{I}$ issues the e-coupon.

Merchant $\mathcal{M}$ is in charge of verifying if the e-coupon is valid and has not been used before at the same $\mathcal{M}$. As the e-coupon was issued for a specific $\mathcal{C}$, the merchant $\mathcal{M}$ has to ensure if $\mathcal{C}$ has the right to use it.

$\mathcal{C}$ is the actor who makes use of the platform to generate the identity proofs of its credential attributes, and requests and redeems the e-coupons.

Next, the promotional e-coupon is detailed as follows: SN represents the identifier of the selected e-coupon, I_1 includes the offer data of the promotion, I_2 contains all the public parameters to prove the ownership of the e-coupon during the redemption phase, and $T_1 - T_2$ define the time interval for redeeming.

3.2 Security Requirements

The protocol has to consider and guarantee the following security requirements:

- Owner authentication: Merchants should be able to verify the customer's e-coupon ownership during the redeem phase.
- Anonymity: The system has to provide full anonymity during the registration phase and redeem phase. The scheme allows a selective personal data disclosure during the issuance phase.
- Unforgeability: All the parts have to be able to verify if an e-coupon has been issued by an authorized issuer, and if the e-coupon has not been manipulated.

– Double-redeem protection: The system must provide protection about intents of reusing an e-coupon at the same merchant in an offline mode.
– Issuer non-repudiation: Once a valid e-coupon has been issued, issuer should not able to deny it.
– Non-transferability: Customers should not be able to share e-coupons.

3.3 Cryptographic Background

We briefly review the cryptography techniques that will be used in our proposal.

ZKP: Zero-Knowledge Proof
The aim of a zero-knowledge proof is to prove the validity of a statement given to a verifier. This kind of proof is suitable to prove the possession of a secret without revealing the content. The Schnorr protocol [11] is an identification scheme based on discrete logarithms, that can be used as a interactive zero knowledge proof. Figure 1 shows in detail the three steps (commitment, challenge and response) involved in the protocol.

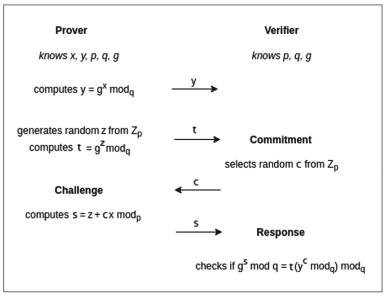

Fig. 1. Schnorr protocol

ABC: Attribute-Based Credential
The aim of an attribute-based credential is to provide an authentication mechanism based on a selective attributes method. In this way, every credential contains attributes that the user can either reveal or keep hidden. *Idemix* is an ABC protocol based on a Camenisch-Lysyanskaya (CL) signature scheme [12], that provides a solution for strong privacy-preserving authentication with a disclosure method of certified attributes. Table 1 defines the *Idemix* parameters involved in the protocol.

Table 1. *Idemix* protocol parameters

$\mathcal{S}$	CL-signature scheme, credential structure, context
x	Master secret key
m_{H_1}	Master secret key attribute
m_{H_i}	Credential/proof hidden attributes
m_{K_i}	Credential/proof known attributes
common	Public parameters
T	Commitment prove values (aggregation of t-values)
T'	Commitment verify values (aggregation of t-values)
c	Challenge prove
c'	Challenge verify
s_i	Response values (s-values)
$\mathcal{P}$	Proof = (c,s_i,common)
n_i	Random value

4 Proposal

This section is divided into two main subsections. The first subsection defines the prerequisites for the system set-up, and the second subsection explains in detail each step of the proposed protocol.

4.1 System Set-Up

In this scheme, the entities $\mathcal{T}$ and $\mathcal{I}$ previously generated an asymmetric key pair. As a first step, $\mathcal{C}$ generates a master secret key x as m_{H_1} and enrolls a digital residence certificate with two fields: $PersonalID$ as m_{H_2} and $Zipcode$ as m_{H_3} in $\mathcal{T}$. Then, $\mathcal{T}$ generates a serial number attribute to identify the credential as m_{K_1}. Once the structure of the credential is defined, both parties agree to run the CL-signature scheme (Algorithm 1) to create the signature over the attributes specified in the credential structure. As a first step, $\mathcal{C}$ calls the function `CL.commit` to build the commitment for each of the hidden $\{m_{H_i}\}$ attributes using the system parameters retrieved from $\mathcal{T}((n, S, Z, \{R_i\}_{i \in M}))$. As a result, C obtains U as an aggregation of commitments. Then, $\mathcal{C}$ sends U to I, and I has to call the function `CL.sign` to prepare the pre-signature with the aggregation of U and the known $\{m_{K_i}\}$ attributes. Finally, I sends the pre-signature to C, and C calls the function `CL.build` to compute the signature. Figure 2 shows the credential $[(m_{H_1}, m_{H_2}, m_{H_3}, m_{K_1}), (\text{CL Signature})]$.

Algorithm 1. Camenisch-Lysyanskaya signature

function CL.COMMIT($\{m_i\}_{i \in M_H}, (n, S, Z, \{R_i\}_{i \in M})$)
  $v' \leftarrow$ RANDOM( )
  $U \leftarrow S^{v'} \mod n$
  for each $i \in M_H$ **do**
    $U \leftarrow U \cdot R_i^{m_i} \mod n$
  return (U, v')
function CL.SIGN($U, \{m_i\}_{i \in M \setminus M_H}, (n, S, Z, \{R_i\}_{i \in M}), (p', q')$)
  $v'' \leftarrow$ RANDOM( )
  $U \leftarrow U \cdot S^{v''} \mod n$
  for each $i \in M \setminus M_H$ **do**
    $U \leftarrow U \cdot R_i^{m_i} \mod n$
  $Q \leftarrow Z \cdot U^{-1} \mod n$
  $e \leftarrow$ RANDOMPRIME( )
  $d \leftarrow e^{-1} \mod (p' \cdot q')$
  $A \leftarrow Q^d \mod n$
  return (A, e, v'')
function CL.BUILD($v', (A, e, v'')$)
  $v \leftarrow v' + v''$
  return (A, e, v)

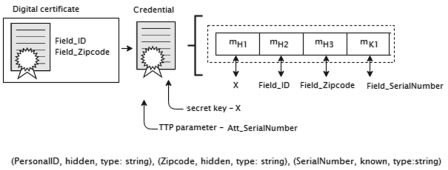

Fig. 2. *Idemix* credential structure

4.2 Phases

The phases of our system are: *Customer registration*, where the system checks the eligibility of the user and gets registered in the portal; *Issue*, that consists of selecting an eligible promotion and disclose the required identity attributes for issuing an e-coupon; *Redeem*, where the customer redeems an e-coupon at a merchant. Table 2 defines the notation used in the description of the phases.

Phase 1 *Customer registration*
As a first step, $\mathcal{C}$ contacts with $\mathcal{I}$ to prove her eligibility to be registered in the portal, following the protocol flow described in Fig. 3. Then, $\mathcal{I}$ sends a random value n_i to $\mathcal{C}$, required to build a proof (Algorithm 2). First, $\mathcal{C}$ has to define the proof specification, which contains the disclosed and undisclosed attributes. That is, m_{K_1} as a known attribute, while m_{H_1}, m_{H_2}, m_{H_3} remain hidden. During the registration phase there is not a disclosure of hidden attributes. C calls the function `ProveCL.randomise` to generate a randomised signature, and the function `ProveCL.tvalues` to compute the commitment T over the aggregation of the undisclosed attributes, and the randomised signature (t-values).

Table 2. Protocol parameters

x	Master secret key
y	Schnorr public key
z	Schnorr value
t	Schnorr commitment
c	Schnorr challenge
r	Schnorr response
$Policy$	e-coupon promotion policy
$Coupon$	$SN \mid I_1 \mid I_2 \mid T_1 \mid T_2$
$Sign(Coupon)$	$\mathcal{I}$'s signature on the e-coupon
$pk_{\mathcal{I}}$	Issuer public key
$pk_{\mathcal{T}}$	Trusted third party public key

Next, C calls the `ProveCL.challenge` function to compute the challenge c, and computes the responses $\{s_i\}$ for every t-value. As a result, the following certified and signed $\mathcal{P}$ proof is generated: $\mathcal{P} = (c, (\hat{e}, \hat{v}, \{\hat{m}_1\}, \{\hat{m}_2\}, \{\hat{m}_3\}), common)$. The computed proof $\mathcal{P}$ will be used to demonstrate the eligibility of $\mathcal{C}$ as a physical person who has certified attributes.

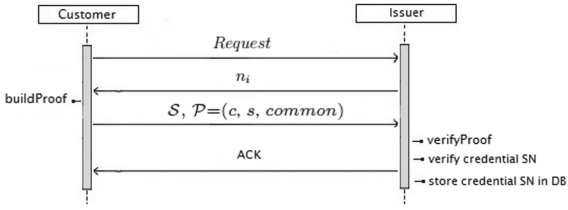

Fig. 3. Redeem protocol flow

$\mathcal{C}$ sends the generated proof $\mathcal{P}$ to $\mathcal{I}$, and $\mathcal{I}$ has to verify the proof $\mathcal{P}$ (Algorithm 3). Then, $\mathcal{I}$ calls the function `VerifyCL.tvalues` to validate the received $\mathcal{P}$, computing the aggregation of the randomised signature, the disclosed $\{m_{K_i}\}$ attributes and the undisclosed $\{m_{H_i}\}$ attributes. Then, I calls the function `VerifyCL.challenge` to compute the challenge c' to be compared against the received proof $\mathcal{P}$. If the proof $\mathcal{P}$ is accepted, $\mathcal{I}$ has to check the non-reusability of the credential, checking the credential serial number in an internal database. If the credential has not been used, $\mathcal{I}$ sends an ACK to $\mathcal{C}$. At this point, $\mathcal{C}$ is eligible to get registered in the portal. Moreover, $\mathcal{I}$ has to store the serial number of the credential in an internal database to prevent multiple registration using the same credential.

Algorithm 2. Build Proof

Protocol : $BuildProof$ [IN: $\{m_1$,$\mathcal{S}$,n_i,OUT: $\{\mathcal{P}\}$]
$ProveCL.randomise(\mathcal{S}) \rightarrow (e, v')$,common
$ProveCL.tvalues(\mathcal{S}$,common$) \rightarrow (\tilde{e}, \tilde{v}, \{\tilde{m}_i\}_{i \in m_H}), T$
$ProveCL.challenge$(context,$common$,T,$n_i) \rightarrow c$
$ProveCL.svalues(\mathcal{S}, (e, \tilde{e}, \tilde{v}, \{\tilde{m}_i\}_{i \in m_H}$,c$) \rightarrow s\text{-}values$
return $\mathcal{P} = (c$,$s\text{-}values$,$common)$

Algorithm 2.1. Prove CL

function ProveCL.randomise$([(A, e, v), (n, S, Z, \{R_i\}_{i \in M})])$
  $r \leftarrow$ Random( )
  $A' \leftarrow A \cdot S^r \mod n$
  $Common \leftarrow A'$
  $v' \leftarrow v - e \cdot r$
  return $(e, v'), A'$
function ProveCL.tvalues$([\{m_i\}_{i \in m_H}, (n, S, Z, \{R_i\}_{i \in M})], (A'))$
  $\tilde{e} \leftarrow$ Random( )
  $\tilde{v} \leftarrow$ Random( )
  $\tilde{Z} \leftarrow A'^{\tilde{e}} \cdot S^{\tilde{v}} \mod n$
  for each $i \in m_H$ **do**
    $\tilde{m}_i \leftarrow$ Random( )
    $\tilde{Z} \leftarrow \tilde{Z} \cdot R_i^{\tilde{m}_i} \mod n$
  return $(\tilde{e}, \tilde{v}, \{\tilde{m}_i\}_{i \in m_H}), \tilde{Z}$
function ProveCL.challenge$(context, common$,T, $n_i)$
  $c := H(context, common$,T, $n_i)$
  return $(\hat{e}, \hat{v}, \{\hat{m}_i\}_{i \in m_H})$
function ProveCL.svalues$(\{m_i\}_{i \in m_H}, (e, \tilde{e}, \tilde{v}, \{\tilde{m}_i\}_{i \in m_H}), c)$
  $\hat{e} \leftarrow \tilde{e} + c \cdot e$
  $\hat{v} \leftarrow \tilde{v} + c \cdot v'$
  for each $i \in m_H$ **do**
    $\hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
  return $(\hat{e}, \hat{v}, \{\hat{m}_i\}_{i \in m_H})$

Algorithm 3. Verify Proof

Protocol : $VerifyProof$ [IN: $\{\mathcal{S}$, (c,s-values,common), $n_i\}$, OUT: { accept or reject $\mathcal{P}\}$]
$VerifyCL.tvalues(\mathcal{S},\mathcal{P}) \rightarrow T$'
$VerifyCL.challenge$(context,$common$,T',$n_i) \rightarrow$ c'
return If $c \equiv c$' accept $\mathcal{P}$ or reject otherwise

Algorithm 3.1. Verify CL

function VerifyCL.tvalues$([\{m_i\}_{i \in m_K}), (n, S, Z, \{R_i\}_{i \in M}), [c, (\hat{e}, \hat{v}, \{\hat{m}_i\}_{i \in m_H}), A']$)
  $\hat{Z} \leftarrow Z^{-c} \cdot A'^{\hat{e}} \cdot S^{\hat{v}} \mod n$
  for each $i \in m_K$ **do**
    $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{c \cdot m_i} \mod n$
  for each $i \in m_H$ **do**
    $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{\hat{m}_i} \mod n$
  return $\hat{Z}$
function VerifyCL.challenge$(context, common$,T', $n_i)$
  $c' := H(context, common$,T, $n_i)$
  return c'

Phase 2 *Issue protocol*
When $\mathcal{C}$ wants to request a promotion from the portal, has to follow the protocol flow described in Fig. 4. As a first step, she has to request the specified e-coupon using its SN. Next, $\mathcal{I}$ replays with the policy of the selected e-coupon and a fresh random number n_i to $\mathcal{C}$, required to build a proof (Algorithm 2). First, $\mathcal{C}$ has to use the proof specification of the received promotion policy, which defines the disclosed and undisclosed attributes to be proven. In our schema all the promotions have to be redeem by customers who live in a specified area. So, the zip code attribute m_{H_3} has to be disclosed as m_{K_2} to check the residence requirement. That is, m_{K_1}, m_{K_2} are known attributes, while m_{H_1}, m_{H_2} remain hidden. $\mathcal{C}$ calls the function `ProveCL.randomise` to generate a randomised signature, and the function `ProveCL.tvalues` to compute the commitment T over the aggregation of the undisclosed attributes and the randomised signature (t-values). Next, C calls the `ProveCL.challenge` function to compute the challenge c, and computes the responses $\{s_i\}$ (s-values) for every t-value. As a result, the following certified and signed proof $\mathcal{P}$ is generated: $\mathcal{P} = (c, (\hat{e}, \hat{v}, \{\hat{m}_1\}, \{\hat{m}_2\}), common)$. The computed proof $\mathcal{P}$ will be used to demonstrate the residence area of $\mathcal{C}$, in particular the zip code.

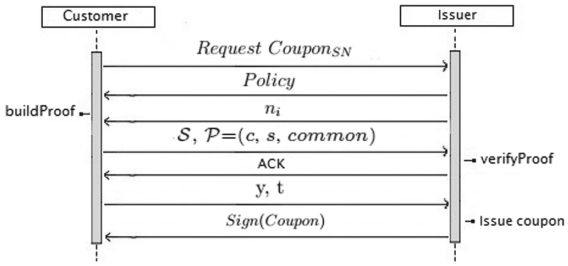

Fig. 4. Issue protocol flow

$\mathcal{C}$ sends the generated proof $\mathcal{P}$ to $\mathcal{I}$, and $\mathcal{I}$ has to verify the proof $\mathcal{P}$ (Algorithm 3). Then, $\mathcal{I}$ calls the function `VerifyCL.tvalues` to validate the received $\mathcal{P}$, computing the aggregation of the randomised signature, the disclosed $\{m_{K_i}\}$ attributes and the undisclosed $\{m_{H_i}\}$ attributes. Then, I calls the function `VerifyCL.challenge` to compute the challenge c' to be compared against the received proof $\mathcal{P}$. If the proof $\mathcal{P}$ is accepted, $\mathcal{I}$ sends an ACK to $\mathcal{C}$. Then, $\mathcal{C}$ contacts with $\mathcal{T}$ to obtain the Schnorr public parameters to compute the Schnorr public key y and the Schnorr commitment t. After that, both values are sent to $\mathcal{I}$. Next, $\mathcal{I}$ includes the Schnorr parameters obtained from C inside I_2, and prepares all the remaining data to be included inside the $Coupon$ as the offer information and the time interval to be redeem. As the last step, $\mathcal{I}$ signs the issued $Coupon$ and delivers $Sign(Coupon)$ to $\mathcal{C}$.

Phase 3 *Redeem protocol*
During the redeem protocol (see Fig. 5), $\mathcal{M}$ has to check that $Sign(Coupon)$ presented by $\mathcal{C}$, is not a fake copy. Thus, $\mathcal{M}$ has to verify the signature using the $\mathcal{I}$'s $pk_{\mathcal{I}}$. If the verification is successful, $\mathcal{M}$ has to check if $\mathcal{C}$ is the legitimate owner of $Coupon$. To do that, $\mathcal{M}$ starts the Schnorr identity protocol (see Sect. 3.3) to verify if $\mathcal{C}$ is able to answer a generated challenge using the information inside the I_2. $\mathcal{M}$ generates a time-variant random challenge c and sends it to $\mathcal{C}$. Then, $\mathcal{C}$ has to compute the response s and send it back to $\mathcal{M}$. As a result, $\mathcal{M}$ verifies the received response to know if $\mathcal{C}$ is eligible to redeem $Coupon$. If the verification is successful, $\mathcal{M}$ checks if the SN has not been used before, and if the time interval between T_1 and T_2 have not expired. If both verifications are successful, $\mathcal{M}$ redeem $Coupon$ and stores the SN in a local non-persistent database. In our scheme, portal offers e-coupons with a small limited lifetime period of use since the moment that $\mathcal{I}$ issues the e-coupons. As a last step, $\mathcal{M}$ sends an ACK to $\mathcal{C}$.

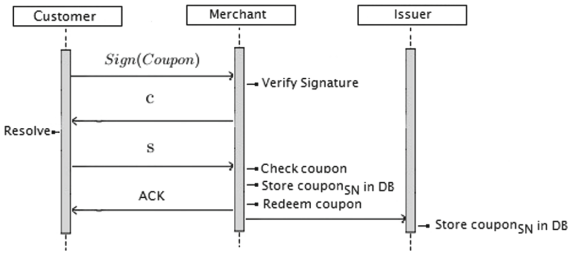

Fig. 5. Redeem protocol flow

5 Security Analysis

In this section we include a brief security analysis of the following requirements:

1. Owner authentication: During the issuance phase, $\mathcal{C}$ generates a public value y linked to the master secret key x. Only the legitime owner of x will be able to resolve the challenge-response step during the redemption phase.
2. Anonymity: During the registration phase, $\mathcal{C}$ sends the disclosed attributes to $\mathcal{I}$. In our scheme only the known attributes, as the serial number of the credential, was disclosed, while the other identity attributes remains hidden.
3. Unforgeability: $\mathcal{I}$ generates a digital signature $Sign(Coupon)$ during the issuing phase. Any forged e-coupon that is not generated by $\mathcal{I}$, or it is modified will be detected during the redemption verification as illegal.
4. Double-redeem protection: In our scheme we only consider the double-redeem issue if $\mathcal{C}$ tries to use the e-coupon more than one time in the same $\mathcal{M}$, which is resolved by using a memory database to check the redeemed e-coupons. Also, there is a lifetime period to mitigate the impact if the same $\mathcal{C}$ tries to use the e-coupon in different $\mathcal{M}$. In addition, it is possible to use a global database between all the merchants.

5. Issuer non-repudiation: A credential proof can be verified with the corresponding $pk_{\mathcal{T}}$, so the $\mathcal{T}$ can not deny it. An e-coupon can be verified with the corresponding $pk_{\mathcal{I}}$, so the $\mathcal{I}$ can not deny it.
6. Non-transferability: If $\mathcal{C}$ shares the e-coupon, it has to expose its master secret key x to other customers to pass the challenge-response step during the redeem phase. Thus, the protocol must to discourage customers to reveal a valuable secret which is linked with the credential.

6 Conclusions and Further Work

Our proposal offers a portal to retrieve e-coupons, depending on the issuer promotion requirements. Customers can generate proofs from a digital certificate to prove identity requirements with a selective privacy mechanism. We offer a trusted registration mechanism providing anonymity. The design of the system has been performed taking into account security and privacy requirements described for e-coupons. As a future work, we want to extend this solution to other scenarios and build a prototype. Moreover, the security and the performance analysis will be presented in a formal way.

Acknowledgments. This work is partially financed by the European Social Fund and the Spanish Government under the projects TIN2014-54945-R and TIN2015-70054-REDC.

References

1. Coupon Savings Report (2016). https://www.nchmarketing.com/nchpressreleases.aspx. Accessed 17 Aug 2017
2. Terms and Conditions - Christmas promotion 31 days. https://app.mcdonalds.es/landing/legal/legal31DiasLocosNavidad.html. Accessed 17 Aug 2017
3. Kumar, M., Anand, R., Jhingran, A., Mohan, R.: Sales promotions on the Internet. In: Proceedings of the 3rd USENIX Workshop on Electronic Commerce, pp. 167–176, Boston (1998)
4. Jakobsson, M., Mackenzie, P.D., Stern, J.P.: Secure and lightweight advertising on the web. J. Comput. Telecommun. Netw. **31**(11), 1101–1109 (1999)
5. Chang, C.C., Wu, C.C., Lin, I.C.: A secure e-coupon system for mobile users. Int. J. Comput. Sci. Netw. Secur. **6**(1), 273–280 (2006)
6. Aigner, M., Dominikus, S., Feldhofer, M.: A system of secure virtual coupons using NFC technology. In: Proceedings of IEEE International Conference on Pervasive Computing and Communication Workshops, pp. 362–366, New York (2007)
7. Chang, C.C., Sun, C.Y.: A secure and efficient authentication scheme for e-coupon systems. Wirel. Pers. Commun. **77**(4), 2981–2996 (2014)
8. Chang, C.C., Lin, I.C., Chi, Y.L.: Secure electronic coupons. In: Proceedings - 10th Asia Joint Conference on Information Security, AsiaJCIS 2015, pp. 104–109 (2015)
9. Liu, W., Mu, Y., Yang, G., Yu, Y.: Efficient E-coupon systems with strong user privacy. Telecommun. Syst. **64**, 695–708 (2017)
10. Camenisch, J., Mödersheim, S., Sommer, D.: A formal model of identity mixer. In: Kowalewski, S., Roveri, M. (eds.) FMICS 2010. LNCS, vol. 6371, pp. 198–214. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15898-8_13

11. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
12. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_20

Revisiting Two-Hop Distance-Bounding Protocols: Are You Really Close Enough?

Nektaria Kaloudi[1] and Aikaterini Mitrokotsa[2(✉)]

[1] University of the Aegean, Samos, Greece
n.kaloudi033@gmail.com
[2] Chalmers University of Technology, Gothenburg, Sweden
aikmitr@chalmers.se

Abstract. The emergence of ubiquitous computing has led to multiple heterogeneous devices with increased connectivity. In this communication paradigm everything is inter-connected and proximity-based authentication is an indispensable requirement in multiple applications including contactless payments and access control to restricted services/places. Distance-bounding (DB) protocols is the main approach employed to achieve accurate proximity-based authentication. Traditional distance-bounding requires that the prover and the verifier are in each other's communication range. Recently, Pagnin *et al.* have proposed a two-hop DB protocol that allows proximity-based authentication, when the prover and the verifier need to rely on an intermediate untrusted party (linker). In this paper, we investigate further the topic of two-hop distance-bounding. We analyse the security of the Pagnin *et al.* protocol for internal adversaries and we investigate the impact of the position of the linker in the distance-bounding process. We propose a new two-hop DB protocol that is more lightweight and avoids the identified problems. Finally, we extend the protocol to the multi-hop setting and we provide a detailed security analysis for internal adversaries.

Keywords: Distance-bounding · Authentication · Relay attacks

1 Introduction

Ubiquitous computing technologies have affected radically our communications. Multiple heterogeneous devices are inter-connected and proximity-based authentication has been adopted in a wide range of applications *e.g.,* remote unlocking, contactless payments, proximity cards for access control in services/places. Distance-bounding (DB) protocols [1,2] is a valuable tool in this ubiquitous computing paradigm, since they determine an upper bound on the physical distance between two communicating parties by measuring the round-trip-time between the exchanged challenges and responses. Distance-bounding protocols have received a lot of attention in the literature and multiple works have been published focusing on the selection of optimal parameters for DB protocols [3–5],

G. P. Hancke and E. Damiani (Eds.): WISTP 2017, LNCS 10741, pp. 177–188, 2018.
https://doi.org/10.1007/978-3-319-93524-9_12

on related privacy issues [6,7], as well as multiple attacks [8–11] against existing DB protocols and proposed solutions [12–14] that combat identified weaknesses. Furthermore, the concept of grouping proof distance-bounding protocols [15] has been recently introduced in order to provide not only a proof of the presence of multiple provers at the same time but also assurance regarding the physical proximity of the provers. However, in many cases (*e.g.,* communication in Vehicular ad-hoc Networks (VANETs) and Wireless Sensor Networks (WSNs)), we need to have a proof of proximity even when the two main parties prover and verifier are not in the direct communication range of each other, but have to rely on an untrusted intermediate node (linker). Recently, Pagnin *et al.* [16] proposed an extension of traditional DB protocols that can verify the proximity of both next-hop and two-hop neighbours.

In this article, we investigate the security of the Pagnin *et al.* [16] protocol for internal adversaries and we identify three weaknesses. We discuss how the location of intermediate untrusted linkers affects the DB process and we propose a new two-hop DB protocol that overcomes the identified problems, requires no computation from the linker, and provides lower attack success probabilities in comparison to the Pagnin *et al.* protocol. Finally, we investigate how the proposed two-hop DB protocol can be extended to the multi-hop setting [17] *i.e.,* when multiple intermediate linkers are used to reach the prover and we discuss how the attack success probabilities are affected.

2 Two-Hop Distance Bounding

Traditional one-hop DB protocols consider two "legitimate" parties: a single verifier and a single prover within one-hop communication range. Pagnin *et al.* [16] introduced the concept of two-hop DB which involves three entities: a trusted verifier ($\mathcal{V}$), an untrusted prover ($\mathcal{P}$) and an untrusted in-between linker ($\mathcal{L}$). The goal is to bound the distance between $\mathcal{V}$ and $\mathcal{P}$, while relying on the intermediate untrusted linker $\mathcal{L}$. More precisely, $\mathcal{P}$ wants to be authenticated by $\mathcal{V}$, while $\mathcal{V}$ wants to bound the distance of $\mathcal{P}$. The prover $\mathcal{P}$ is not in the communication range of the verifier $\mathcal{V}$ (two-hop neighbours), while $\mathcal{L}$ is within the communication range (one-hop) of both $\mathcal{V}$ and $\mathcal{P}$. All nodes ($\mathcal{P}$, $\mathcal{V}$ and $\mathcal{L}$) use the same communication channel.

The two-hop DB protocol proposed by Pagnin *et al.* is depicted in Fig. 1 and employs two secret keys: $x_{\mathcal{VL}}$ shared between $\mathcal{V}$ and $\mathcal{L}$, and $x_{\mathcal{VP}}$ shared between $\mathcal{V}$ and $\mathcal{P}$. $N_{\mathcal{V}}$, $N_{\mathcal{P}}$ and $N_{\mathcal{L}}$ denote random nonces, $f_y()$ a pseudorandom function, $\mathsf{Enc}_y()$ an encryption function and $\mathsf{MAC}_y()$ a function that computes a message authentication code. All three functions use an appropriate key here denoted as y. More precisely, it is composed of the following phases:

– **Initialization phase:** $\mathcal{V}$, $\mathcal{L}$, $\mathcal{P}$ generate randomly their corresponding nonces. Both $\mathcal{V}$ and $\mathcal{L}$ use the pseudorandom function h and the secret key $x_{\mathcal{VL}}$ in order to calculate two n-bit sequences a_0 and a_1, while $\mathcal{P}$ uses the pseudorandom function h with the key $x_{\mathcal{VP}}$ to generate the d_0 and d_1 registers.

– **Distance-bounding phase:** In this phase, a fast bit exchange phase takes place, where $\mathcal{V}$ sends out one bit c_i to $\mathcal{L}$ and starts two clocks $t_{\mathcal{L}}$ and $t_{\mathcal{P}}$. $\mathcal{L}$ responds to both $\mathcal{V}$ and $\mathcal{P}$ with one calculated bit $\ell_i = (a_{c_i})_i$ and $\mathcal{V}$ stops the first timer $t_{\mathcal{L}}$. The delay time of the responses enables $\mathcal{V}$ to compute an upper-bound on the next-hop distance with $\mathcal{L}$ and after observing the second timer $t_{\mathcal{P}}$, it computes the other bound on the two-hop distance with $\mathcal{P}$. This phase is time-critical and is separated by n rounds of timed challenge/response exchanges.
– **Verification phase:** After $\mathcal{V}$ receives $\mathcal{P}$'s nonce, ℓ_i from $\mathcal{L}$ and the responses r_i, it computes the d_0, d_1 registers in order to verify r_i and ℓ_i. According to the result, $\mathcal{V}$ measures the bound on the physical distance to $\mathcal{V}$ using the clocks $t_{\mathcal{L}}$, $t_{\mathcal{P}}$.

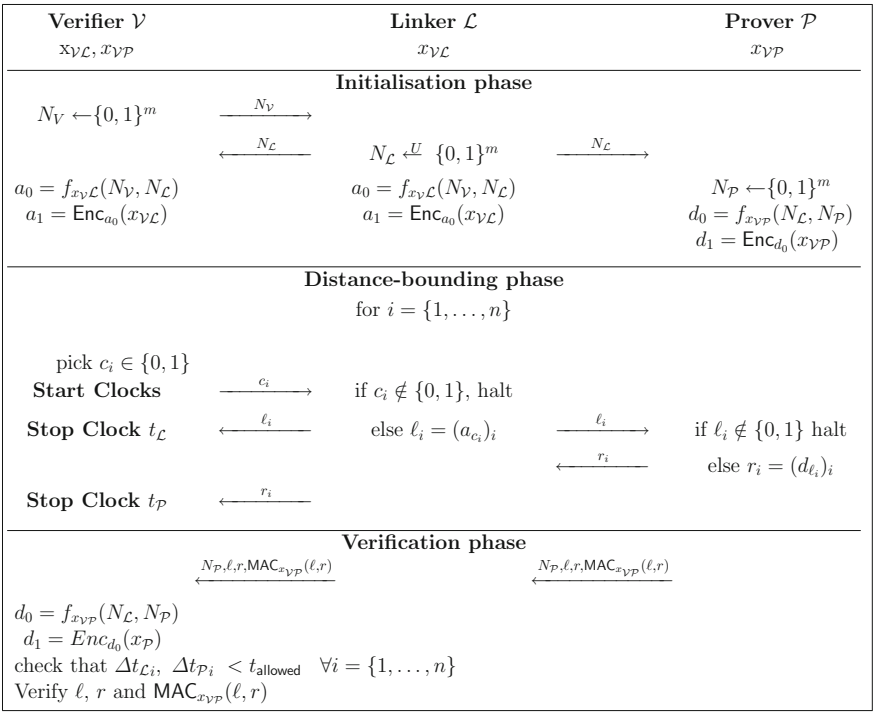

Fig. 1. The Pagnin *et al.* [16] two-hop distance-bounding protocol.

2.1 Security Analysis

We describe what are the effects of a malicious linker $\mathcal{L}$ in the different phases of the Pagnin *et al.* protocol and analyse malicious behaviour that has not been considered before. We need to stress that the first two described security issues

are related to weaknesses of the Pagnin *et al.* protocol to identify possible modifications of the transmitted messages that may subsequently lead to denial of service (DoS) attacks.

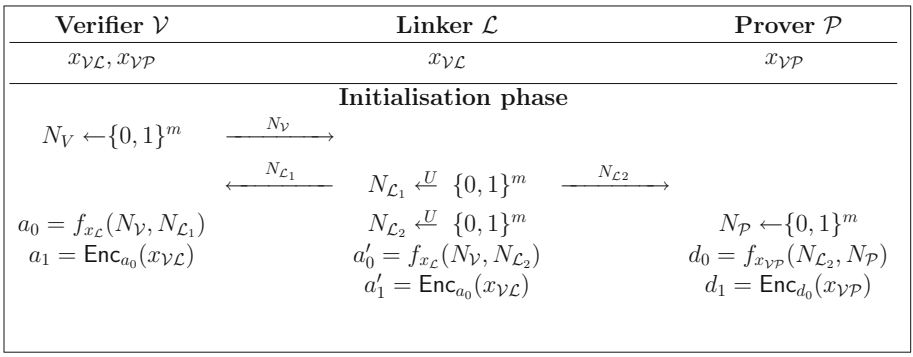

Fig. 2. Attack using different nonces.

- *Malicious $\mathcal{L}$ in the initialisation phase:* A malicious $\mathcal{L}$ that receives $N_{\mathcal{V}}$ from $\mathcal{V}$, could send different nonces $N_{\mathcal{L}_1}$ and $N_{\mathcal{L}_2}$ to $\mathcal{V}$ and $\mathcal{P}$ correspondingly (Fig. 2), thus, disrupting the whole protocol and leading $\mathcal{P}$ and $\mathcal{V}$ to compute different registers $d_0 = f_{x_{\mathcal{VP}}}(N_{\mathcal{L}_2}, N_{\mathcal{P}})$ *i.e.,* computed by $\mathcal{P}$ in the initialisation, and $d'_0 = f_{x_{\mathcal{VP}}}(N_{\mathcal{L}_1}, N_{\mathcal{P}})$ *i.e.,* computed by $\mathcal{V}$ in the verification phase. Consequently, the computation will be completely different and as a result the protocol will fail.
- *Malicious $\mathcal{L}$ in the verification phase:* Similarly to the previous attack, a malicious $\mathcal{L}$ could modify the transmitted $N_{\mathcal{P}}$, thus disrupting the computation of d_0 by $\mathcal{V}$ and consequently leading to the failure of the protocol. This can be easily detected if a MAC of the nonce $N_{\mathcal{P}}$ is also sent. Although this would not stop the DoS attack, it would definitely be useful to detect on time the misbehaviour of $\mathcal{L}$.
- *Malicious $\mathcal{L}$ in the distance-bounding phase:* A malicious $\mathcal{L}$ can act as a man-in-the-middle and perform the attack that was initially described by Kim *et al.* [18] against one-hop DB protocols [19]– against the Pagnin *et al.* [16] protocol in order to recover bits of the key $x_{\mathcal{VP}}$, when as Enc function is employed the one time pad *i.e.,* $d_1 = d_0 \oplus x_{\mathcal{VP}}$. More precisely, $\mathcal{L}$ can during the DB phase, toggle the value of one bit ℓ_i *i.e.,* $\ell'_i \neq \ell_i$ transmit the same ℓ'_i to $\mathcal{V}$ and $\mathcal{P}$ and leave all other messages untouched. Then, $\mathcal{L}$ can observe the verifier's reaction. If $\mathcal{V}$ accepts $\mathcal{P}$, it means that $\mathcal{P}$'s answer r_i was correct. Thus, the bit of the key $x_{\mathcal{VP}i}$ will be 0, because $d_{0i} = d_{1i}$. If $\mathcal{V}$ does not accept then $d_{0i} \neq d_{1i}$, thus, $x_{\mathcal{VP}i} = 1$. We should note that in the Pagnin *et al.* protocol is stated: "$\mathcal{V}$ computes d_0 and d_1 and verifies that all received ℓ_i and r_i, $\forall i \in \{1, \ldots, n\}$ are correct." However, this leaves rather unclear if $\mathcal{V}$ actually computes ℓ_i or simply verifies that the ℓ_is and r_is received during the distance-bounding phase match the ones received during the verification

phase. To avoid this attack, the necessary condition is that $\mathcal{V}$ should recompute ℓ_i using $\ell_i = (a_{c_i})_i$ in the verification phase and verify the received ℓ_i's. If this recomputation is performed, then the attack will not be successful and the protocol will be aborted because $\mathcal{V}$ will see the differences in ℓ_i. However, if $\mathcal{V}$ simply verifies that the values of ℓ and r received in the DB phase match the ones received in the verification phase and the corresponding $\mathsf{MAC}_{x_{\mathcal{V}\mathcal{P}}}(\ell, r)$ the attack can be performed successfully.

2.2 Effects of Possible Positions of the Linker

It is easy to see that the estimated distance between $\mathcal{V}$ and $\mathcal{P}$ in two-hop DB mainly depends on the position of $\mathcal{L}$. More precisely, $\mathcal{L}$ can be located either on the same line with the other two entities or anywhere else between them. In the second case, a triangle is formed. Let us denote by t_1 the estimated time required to transmit a message from $\mathcal{V}$ to $\mathcal{L}$ and t_2 the corresponding time required to transmit a message from $\mathcal{L}$ to $\mathcal{P}$. t_1 and t_2 can be easily estimated using the $\Delta t_{\mathcal{P}_i}$ and $\Delta t_{\mathcal{L}_i}$ in a two-hop DB protocol [16]. Let us denote by $d(A, B)$ the actual distance between two entities A and B.

If we construct a segment $d(\mathcal{V}, \mathcal{L})$ and a circle with centre $\mathcal{L}$ (Fig. 3, where $\mathcal{P}$ and $\mathcal{P}'$ denote possible locations of the prover), then $\mathcal{P}$ can be any point inside or on this circle. We would like to estimate the third side of the formed triangle $d(\mathcal{V}, \mathcal{P})$. If we knew the included angle between $d(\mathcal{V}, \mathcal{L})$ and $d(\mathcal{L}, \mathcal{P})$, then we could determine the length of the third side. For instance, when the angle is equal to 180° *i.e.,* $\mathcal{V}$, $\mathcal{P}$ and $\mathcal{L}$ are all in the diameter of the circle with centre $\mathcal{L}$ then $d(\mathcal{V}, \mathcal{P}) = d(\mathcal{V}, \mathcal{L}) + d(\mathcal{L}, \mathcal{P}) \approx c(t_1 + t_2)$ where c denotes the speed of light. Thus, in that case we have minimal error in estimating $d(\mathcal{V}, \mathcal{P})$ using a two-hop DB.

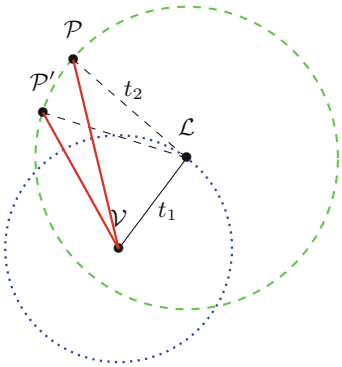

Fig. 3. Depiction of possible locations of a linker, a verifier, and a prover.

We may distinguish two cases in determining the estimation error ϵ on the physical distance between $\mathcal{V}$ and $\mathcal{P}$:

$\mathcal{L}$'s Position is Unknown. Using the triangular inequality, we have an upper and lower bound for the distance: $c \mid t_1 - t_2 \mid -\epsilon < d(\mathcal{V}, \mathcal{P}) < c(t_1 + t_2) - \epsilon$ where ϵ denotes the error in the distance-bounding process. If we have multiple linkers $\mathcal{L}_j$, $j \in \{1, \ldots, m\}$ in the communication range of $\mathcal{V}$ and $\mathcal{P}$, we can run multiple times a two-hop DB protocol. We denote by $t_{j,1}$ the estimated time required to transmit a message from $\mathcal{V}$ to $\mathcal{L}_j$ and by $t_{j,2}$ the estimated time required to transmit a message from $\mathcal{P}$ to $\mathcal{L}_j$. By observing the different sums $(t_{j,1} + t_{j,2})$, we can deduce which linker is closer (*i.e.,* produces the smallest sum) and thus, which $\mathcal{L}_j$ has the smallest error in the distance estimation, but we cannot find its exact position.

$\mathcal{L}$'s Position is Known. In this case, if we know the exact position of $\mathcal{V}$ and $\mathcal{L}$, we can find the position of $\mathcal{P}$ and consequently $d(\mathcal{V}, \mathcal{P})$. However, to achieve this with high accuracy, we need to have at least three linkers in the communication range of $\mathcal{V}$ and $\mathcal{P}$. We may run a DB protocol three times each for a different linker $\mathcal{L}_j$, where $j \in \{1, 2, 3\}$ to get a good estimate of the three distances $d(\mathcal{L}_j, \mathcal{P})$. If we consider that $d_{\mathcal{L}_j\mathcal{P}}$ denotes the estimated distance via the DB protocol we can consider $d_{\mathcal{L}_j\mathcal{P}} \approx d(\mathcal{L}_j, \mathcal{P})$. Then, using trilateration [20] we can determine the exact location of $\mathcal{P}$. We can compute $d(\mathcal{V}, \mathcal{P}) = \sqrt{(x_\mathcal{V} - x)^2 + (y_\mathcal{V} - y)^2}$, where $x_\mathcal{V}$, $y_\mathcal{V}$ are the coordinates of $\mathcal{V}$ and x, y the coordinates of $\mathcal{P}$. If we know the angle between $d(\mathcal{V}, \mathcal{L}_j)$ and $d(\mathcal{L}_j, \mathcal{P})$ this computation is simplified, *i.e.,* if the triangle is orthogonal we only need to know the locations of two linkers. Alternatively, using the *Received Signal Strength Indicator* (RSSI) method [21] by employing multiple reference points and using the strength of the transmitted signals, we are able to estimate the distance between $\mathcal{V}$ and $\mathcal{P}$. After computing $d(\mathcal{V}, \mathcal{P})$, it is easy to see that the estimation error of the distance $d_{\mathcal{V}\mathcal{P}}$ computed via a two-hop DB protocol is $\epsilon = d_{\mathcal{V}\mathcal{P}} - d(\mathcal{V}, \mathcal{P})$.

Selection of the Best Common Neighbour: In any case, the linker $\mathcal{L}$, should be a common neighbour of both $\mathcal{V}$ and $\mathcal{P}$. Several linkers may be located in the range of $\mathcal{V}$. For every linker $\mathcal{L}_j$ in the communication range δ of the verifier $\mathcal{V}$ it holds $d(\mathcal{V}, \mathcal{L}_j) \leqslant \delta$. Obviously, if in order to reach $\mathcal{P}$, the verifier $\mathcal{V}$ has to rely on multiple linkers (*e.g.,* $\mathcal{L}_1$ and $\mathcal{L}_2$), the error estimation will increase. Optimally, we want to choose the linker $\mathcal{L}_j$ that satisfies the condition $\max\{d(V, \mathcal{L}_j)\} \leq \delta$, while at the same time gives the lowest estimation error ϵ among the possible linkers.

3 The Proposed Two-Hop DB Protocol

In this section, we describe a novel two-hop DB protocol (depicted in Fig. 4), that overcomes the problems identified in Sect. 2.1, while requires no computation from the intermediate linker *i.e.,* $\mathcal{L}$ simply relays messages between $\mathcal{V}$ and $\mathcal{P}$. Furthermore, the proposed protocol as we will show in the security analysis presents higher resistance to the attacks considered by Pagnin *et al.* [16] for internal adversaries.

We consider the same setting of a verifier $\mathcal{V}$, an untrusted linker $\mathcal{L}$, and an untrusted prover $\mathcal{P}$ that wants to be authenticated and prove its proximity. We also consider that there is only one secret key $x_{\mathcal{VP}}$ that is shared between $\mathcal{V}$ and $\mathcal{P}$. In the *initialization phase*, we would like to be sure that $\mathcal{L}$ transfers the correct nonces. A simple solution is that each node uses the shared key to compute a MAC on the nonce it sends. As a result, $\mathcal{P}$ and $\mathcal{V}$ can understand if the received nonce from $\mathcal{L}$ has been altered or not. Subsequently, $\mathcal{V}$ and $\mathcal{P}$ compute the values a_0 and a_1 using a pseudorandom function and an encryption function correspondingly. In the *distance-bounding phase*, $\mathcal{V}$ chooses a random challenge bit, which is forwarded by $\mathcal{L}$ to $\mathcal{P}$, while $\mathcal{P}$ computes and sends back the responses $r_i = a(c_i)_i \ \forall i \in \{1 \ldots, n\}$. Finally, in the *verification phase*, $\mathcal{P}$ sends c, r and the corresponding $\mathsf{MAC}_{x_{\mathcal{V},\mathcal{P}}}(c, r)$ that is forwarded by $\mathcal{L}$ and finally verified by $\mathcal{V}$.

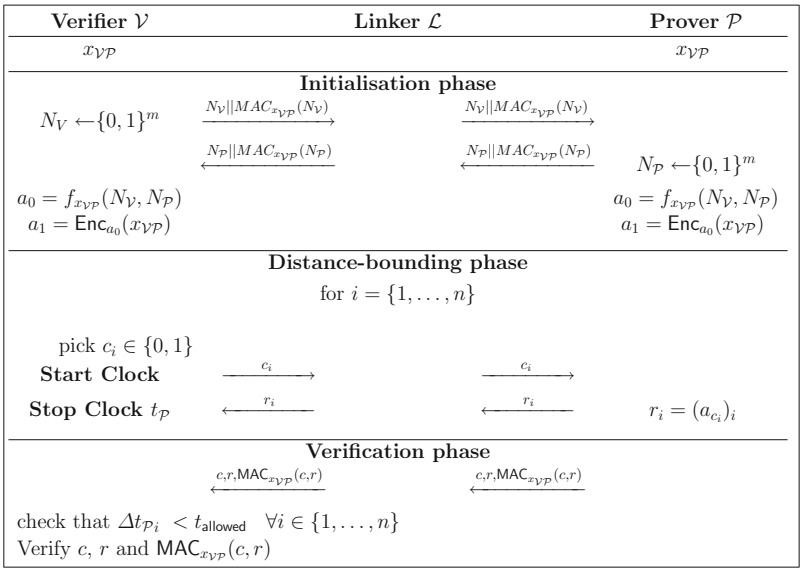

Fig. 4. Efficient two-hop distance-bounding protocol.

3.1 Security Analysis

In this section, we analyse the security of the proposed two-hop DB protocol, considering internal adversaries as in [16].

Case A - Dishonest Prover $\tilde{\mathcal{P}}$, honest linker $\mathcal{L}$: A dishonest prover $\tilde{\mathcal{P}}$ might be located far away from $\mathcal{L}$ and may want to appear closer. So, in the distance-bounding phase $\tilde{\mathcal{P}}$ has to send the wrong response $\tilde{r}_i$ before it receives the challenge c_i from $\mathcal{L}$. Since r_i is determined by the two response registers a_0

and a_1, $\tilde{\mathcal{P}}$ knows r_i if $a_{0i} = a_{1i}$. If $a_{0i} \neq a_{1i}$ then $\tilde{\mathcal{P}}$ has to guess the response r_i. Thus, the success probability is $\left(\frac{3}{4}\right)^n$.

Case B - Honest Prover $\mathcal{P}$, dishonest linker $\tilde{\mathcal{L}}$: $\tilde{\mathcal{L}}$ may want to shorten the distance between $\mathcal{P}$ and $\mathcal{V}$. We may consider two main strategies: In the *first* strategy, $\tilde{\mathcal{L}}$ waits for c_i from $\mathcal{V}$ and sends it to $\mathcal{P}$. Then, $\tilde{\mathcal{L}}$ sends a random response before receiving r_i from $\mathcal{P}$. Since r_i is determined by a_0, a_1, and c_i, the success probability is equal to $\left(\frac{1}{2}\right)$ per round. In the *second* strategy, $\tilde{\mathcal{L}}$ sends a random challenge before receiving c_i from $\mathcal{V}$. Then, $\tilde{\mathcal{L}}$ waits for the response from $\mathcal{P}$ and forwards it to $\mathcal{V}$ when it sends c_i. The success probability is again equal to $\left(\frac{1}{2}\right)$ per round. Thus, the *overall* success probability is $\left(\frac{1}{2}\right)^n$.

We need to note here that the problems identified in Sect. 2.1, when $\mathcal{L}$ is malicious do not apply in the new protocol. By computing the MAC of the nonces in the initialisation phase, $\mathcal{L}$ cannot modify the nonces without being detected, while there is no need to transfer $N_{\mathcal{P}}$ in the verification phase. Also the Kim *et al.* [18] attack cannot be applied since a MAC of all transmitted c_i's and r_i's is verified at the end.

Case C - Dishonest Prover $\tilde{\mathcal{P}}$, dishonest linker $\tilde{\mathcal{L}}$: We may discriminate into two sub-cases.

- *$\tilde{\mathcal{P}}$ and $\tilde{\mathcal{L}}$ do not collaborate:* In this sub-case, the success probability depends on whether Case A or Case B succeeds. More precisely, the success probability depends on whether $\tilde{\mathcal{P}}$ can guess r_i correctly with probability $\left(\frac{3}{4}\right)^n$ (Case A). For a dishonest $\tilde{\mathcal{L}}$ (Case B), the success depends on guessing c_i and r_i correctly (*i.e.,* $(\frac{1}{2})^n$). Thus, the overall success probability is $\left(\frac{5}{8}\right)^n$, (*i.e.,* $\left(\frac{3}{4}\right)$ for Case A or $\left(\frac{1}{2}\right)$ for Case B).
- *$\tilde{\mathcal{P}}$ and $\tilde{\mathcal{L}}$ collaborate:* In this sub-case, $\tilde{\mathcal{P}}$ collaborates with $\tilde{\mathcal{L}}$ in order to appear within the allowed distance bound. $\tilde{\mathcal{P}}$ has two options: to reveal some information about his secret key $x_{\mathcal{VP}}$ (something he does not want to do) or to send one register a_0 or a_1 to $\tilde{\mathcal{L}}$. Thus, in the latter case $\tilde{\mathcal{L}}$ can compute the half of responses r_i correctly and send them in time to $\mathcal{V}$. The other half, have to be guessed by $\tilde{\mathcal{L}}$. So, the overall success probability of the attack in this sub-case is $\left(\frac{3}{4}\right)^n$.

We should point out that in the above security analysis the focus is mainly on distance-shortening attacks since DB protocols are mainly employed in proximity-based authentication settings. In case that a malicious linker on purpose delays to relay information between $\mathcal{P}$ and $\mathcal{V}$ this would lead to failure of the protocol when the condition $\Delta t_{\mathcal{P}_i} < t_{\mathsf{allowed}}$ does not hold.

3.2 Extension to the Multi-hop DB Setting

Up to now, we have considered two-hop DB protocols where $\mathcal{P}$ and $\mathcal{V}$ need to rely on a single intermediate linker. In this section, we investigate how the proposed two-hop DB protocol can be extended to a multi-hop setting *i.e.,* when $\mathcal{V}$ and $\mathcal{P}$ have to rely on multiple intermediate linkers $\mathcal{L}_j$ where $j \in \{i, \ldots, m\}$. Multi-hop

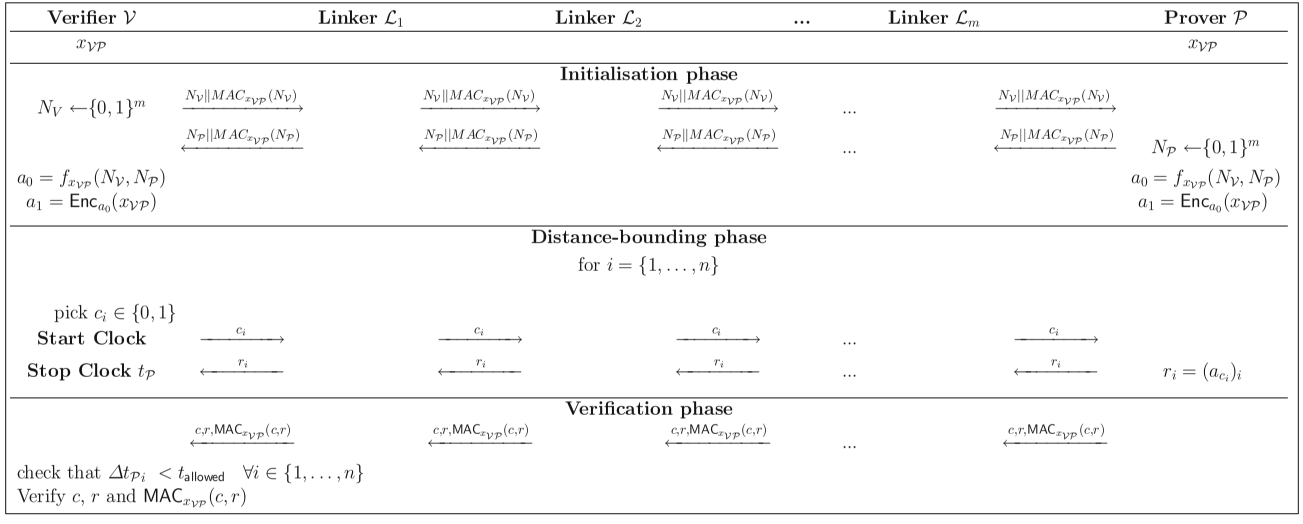

Fig. 5. Extended multi-hop distance-bounding protocol

distance estimation as a generalisation of distance-bounding was proposed for the first time by Mitrokotsa *et al.* [17]. In our proposed multi-hop DB protocol, similarly as before, the goal of $\mathcal{V}$ is to determine an upper-bound of the distance to $\mathcal{P}$ and the role of intermediates $\mathcal{L}_j$ is to forward the messages to $\mathcal{P}$. Our proposed multi-hop DB protocol (depicted in Fig. 5) is a natural extension of the proposed two-hop DB protocol; we discuss briefly its security in this section. It is easy to see that the protocol is similar to the proposed two-hop DB protocol.

The security analysis of the multi-hop DB protocol is similar to the one for the two-hop DB protocol.

Case A - Dishonest Prover $\tilde{\mathcal{P}}$, honest linkers $\mathcal{L}$'s: The overall success probability remains $\left(\frac{3}{4}\right)^n$ since only $\mathcal{P}$ is dishonest.

Case B - Honest Prover $\mathcal{P}$, dishonest linkers $\tilde{\mathcal{L}}$'s: When we have one dishonest linker the success probability is $(\frac{1}{2})^n$ as we have shown. It is easy to see that even if we have k dishonest linkers the overall success probability is again $(\frac{1}{2})^n$ since there is no dependency on the forwarded messages.

Case C - Dishonest Prover $\tilde{\mathcal{P}}$, dishonest linkers $\tilde{\mathcal{L}}$'s: We have two sub-cases:

- *$\tilde{\mathcal{P}}$ and $\tilde{\mathcal{L}}$ do not collaborate:* This sub-case depends on the Cases A and B. Either $\tilde{\mathcal{P}}$ (Case A) will succeed with probability $\left(\frac{3}{4}\right)$ or k $\tilde{\mathcal{L}}$'s (Case B) will succeed with probability $\left(\frac{1}{2}\right)$. Thus, the overall success probability is $\left(\frac{5}{8}\right)^n$.
- *$\tilde{\mathcal{P}}$ and $\tilde{\mathcal{L}}$'s collaborate:* If $\tilde{\mathcal{P}}$ collaborates with k $\tilde{\mathcal{L}}$'s *i.e.,* reveals either a_0 or a_1 (thus, half of the responses) then, the overall success probability will be $\left(\frac{3}{4}\right)^n$. Either $\tilde{\mathcal{P}}$ (Case A) will succeed with probability $\left(\frac{3}{4}\right)$ or k $\tilde{\mathcal{L}}$'s (Case B) will succeed with probability $\left(\frac{1}{2}\right)$. Thus, the overall success probability is $\left(\frac{5}{8}\right)^n$.

Table 1 summarises the best case attack success probabilities for the three protocols that we studied in this paper. It covers all the cases of malicious internal participants.

Table 1. Best case attack success probabilities

Protocol	$\tilde{\mathcal{P}}$, $\mathcal{L}$	$\mathcal{P}$, $\tilde{\mathcal{L}}$	$\tilde{\mathcal{P}}$, $\tilde{\mathcal{L}}$ collaboration	$\tilde{\mathcal{P}}$, $\tilde{\mathcal{L}}$ no collaboration
Two-hop DB	$\left(\frac{3}{4}\right)^n$	$\left(\frac{3}{4}\right)^n$	$\left(\frac{3}{4}\right)^n$	$\left(\frac{3}{4}\right)^n$
New two-hop DB	$\left(\frac{3}{4}\right)^n$	$\left(\frac{1}{2}\right)^n$	$\left(\frac{3}{4}\right)^n$	$\left(\frac{5}{8}\right)^n$
Multi-hop DB	$\left(\frac{3}{4}\right)^n$	$\left(\frac{1}{2}\right)^n$	$\left(\frac{3}{4}\right)^n$	$\left(\frac{5}{8}\right)^n$

4 Conclusion

In this paper, we investigated the problem of two-hop DB protocols. More precisely, we examined the security of the first two-hop DB protocol [16] and we

identified some weaknesses. We also discussed how the location of the linker may affect the distance-bounding process. Furthermore, we proposed a novel two-hop DB protocol, that does not require any computation from the intermediate linker, does not suffer from the identified weaknesses and reduces the attack success probabilities for internal adversaries. Finally, we investigated how the proposed two-hop DB protocol can be extended to the multi-hop DB setting and how the attack success probabilities are affected.

Acknowledgements. This work was partially supported by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 608743, the VR grant "PRECIS: Privacy and Security in Wearable Computing Devices" no 621-2014-4845, the STINT grant "Secure, Private & Efficient Healthcare with wearable computing no IB2015-6001 and the ERASMUS+HE2015 project.

References

1. Dimitrakakis, C., Mitrokotsa, A.: Distance-bounding protocols: are you close enough? IEEE Secur. Priv. **13**(4), 47–51 (2015)
2. Mitrokotsa, A.: Authentication in constrained settings. In: Ors, B., Preneel, B. (eds.) BalkanCryptSec 2014. LNCS, vol. 9024, pp. 3–12. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21356-9_1
3. Dimitrakakis, C., Mitrokotsa, A., Vaudenay, S.: Expected loss bounds for authentication in constrained channels. In: Proceedings of INFOCOM 2012, Orlando, Florida, March 2012
4. Dimitrakakis, C., Mitrokotsa, A., Vaudenay, S.: Expected loss analysis for authentication in constrained channels. J. Comput. Secur. **23**(3), 309–329 (2015)
5. Mitrokotsa, A., Peris-Lopez, P., Dimitrakakis, C., Vaudenay, S.: On selecting the nonce length in distance-bounding protocols. Comput. J. **56**, 1216–1227 (2013)
6. Mitrokotsa, A., Onete, C., Vaudenay, S.: Location leakage in distance bounding: why location privacy does not work. Comput. Secur. **45**, 199–209 (2014)
7. Aumasson, J.-P., Mitrokotsa, A., Peris-Lopez, P.: A note on a privacy-preserving distance-bounding protocol. In: Qing, S., Susilo, W., Wang, G., Liu, D. (eds.) ICICS 2011. LNCS, vol. 7043, pp. 78–92. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25243-3_7
8. Pagnin, E., Yang, A., Hu, Q., Hancke, G., Mitrokotsa, A.: HB+ DB: distance bounding meets human based authentication. Future Gener. Comput. Syst. **80**, 627–639 (2016)
9. Mitrokotsa, A., Dimitrakakis, C., Peris-Lopez, P., Castro, J.C.H.: Reid et al'.s distance bounding protocol and mafia fraud attacks over noisy channels. IEEE Commun. Lett. **14**(2), 121–123 (2010)
10. Bay, A., Boureanu, I., Mitrokotsa, A., Spulber, I., Vaudenay, S.: The Bussard-Bagga and other distance-bounding protocols under attacks. In: Kutyłowski, M., Yung, M. (eds.) Inscrypt 2012. LNCS, vol. 7763, pp. 371–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38519-3_23
11. Mitrokotsa, A., Onete, C., Vaudenay, S.: Mafia fraud attack against the RC distance-bounding protocol. In: Proceedings of the 2012 IEEE RFID Technology and Applications (IEEE RFID T-A), pp. 74–79. IEEE Press, Nice, November 2012

12. Pagnin, E., Yang, A., Hancke, G.P., Mitrokotsa, A.: HB+ DB, mitigating man-in-the-middle attacks against HB+ with distance bounding. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, New York, NY, USA, 22–26 June 2015, pp. 3:1–3:6 (2015)
13. Boureanu, I., Mitrokotsa, A., Vaudenay, S.: Practical and provably secure distance-bounding. J. Comput. Secur. **23**(2), 229–257 (2015)
14. Boureanu, I., Mitrokotsa, A., Vaudenay, S.: Practical and provably secure distance-bounding. In: Proceedings of the 16th Information Security Conference (ISC), Dallas, Texas, USA, November 2013
15. Karlsson, C., Mitrokotsa, A.: Grouping-proof-distance-bounding protocols: keep all your friends close. IEEE Commun. Lett. **20**(7), 1365–1368 (2016)
16. Pagnin, E., Hancke, G., Mitrokotsa, A.: Using distance-bounding protocols to securely verify the proximity of two-hop neighbours. IEEE Commun. Lett. **19**(7), 1173–1176 (2015)
17. Mitrokotsa, A., Onete, C., Pagnin, E., Perera, M.: Multi-hop distance estimation: how far are you? Cryptology ePrint Archive, Report 2017/705 (2017). http://eprint.iacr.org/2017/705
18. Kim, C.H., Avoine, G., Koeune, F., Standaert, F.-X., Pereira, O.: The swiss-knife RFID distance bounding protocol. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 98–115. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00730-9_7
19. Tu, Y.J., Piramuthu, S.: RFID distance bounding protocols. In: Proceeidngs of 1st International EURASIP Workshop on RFID Technology (2007)
20. Shih, C.Y., Marrón, P.J.: Cola: complexity-reduced trilateration approach for 3D localization in wireless sensor networks. In: 2010 Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM), pp. 24–32, July 2010
21. Papamanthou, C., Preparata, F.P., Tamassia, R.: Algorithms for location estimation based on RSSI sampling. In: Fekete, S.P. (ed.) ALGOSENSORS 2008. LNCS, vol. 5389, pp. 72–86. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92862-1_7

Author Index