Lorenzo Cavallaro
Dieter Gollmann (Eds.)

Information Security Theory and Practice

Security of Mobile and Cyber-Physical Systems

LNCS 7886

7th IFIP WG 11.2 International Workshop, WISTP 2013
Heraklion, Greece, May 2013
Proceedings

ifip

Lecture Notes in Computer Science 7886

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

Lorenzo Cavallaro Dieter Gollmann (Eds.)

Information Security Theory and Practice.

Security of Mobile and Cyber-Physical Systems

7th IFIP WG 11.2 International Workshop, WISTP 2013
Heraklion, Greece, May 28-30, 2013
Proceedings

Springer

Volume Editors

Lorenzo Cavallaro
Royal Holloway, University of London
Information Security Group
Egham Hill, Egham TW20 0EX, UK
E-mail: lorenzo.cavallaro@rhul.ac.uk

Dieter Gollmann
University of Technology
Institutes of the TU Hamburg-Harburg
Security in Distributed Applications
Harburger Schlossstrasse 20
21079 Hamburg, Germany
E-mail: diego@tuhh.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38529-2 e-ISBN 978-3-642-38530-8
DOI 10.1007/978-3-642-38530-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013938410

CR Subject Classification (1998): E.3, K.6.5, C.5.3, C.3, C.2.0, K.4.4

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Current developments in IT are characterized by an increasing use of personal mobile devices and a growing reliance on IT for supporting industrial applications in the physical world. A new perspective on socio-technical and cyber-physical systems is required that sees in IT more than just an infrastructure but focuses on the ever closer integration between social and technical processes.

Application markets, such as Google Play and Apple App Store, drive a mobile ecosystem, offering new business models with high turnovers and new opportunities, but at the same time attracting cybercriminals and raising new privacy concerns as well.

In the area of cyber-physical systems, research has to go beyond securing the IT infrastructure to consider attacks launched by combining manipulations in physical space and cyber space.

This seventh edition of WISTP featured a lower number of submissions than previous years, with nine out of 19 papers accepted for inclusion in the workshop and proceedings. Submissions were reviewed by at least three reviewers, in some cases by four. This long and rigorous process was only possible thanks to the hard work of the Program Committee members and additional reviewers, listed on the following pages. In addition, we are pleased that Corrado Leita (Symantec Research Labs) and Lejla Batina (Radboud University Nijmegen) accepted our invitation to speak on challenges and opportunities in securing critical infrastructures and near-field communication privacy threats, respectively.

We wish to thank all the people who invested time and energy to make WISTP 2013 a success: first and foremost come all the authors who submitted papers to WISTP and presented them at the workshop. The members of the Program Committee together with all the external reviewers worked hard in evaluating the submissions. The WISTP Steering Committee helped us graciously in all critical decisions. Thanks also go to the 2013 General Chairs Ioannis Askoxylakis and Louis Marinos, the local organizer Nikolaos Petroulakis and their respective teams for handling the local arrangements, to Symantec Research Labs for co-sponsoring WISTP 2013, to Damien Sauveron for maintaining the conference website, and to Claudio Agostino Ardagna and Mauro Conti for their efforts as Publicity Chairs.

May 2013

Lorenzo Cavallaro
Dieter Gollmann

Organization

WISTP 2013 is organized by FORTH-ICS in cooperation with ENISA.

General Chairs

Ioannis G. Askoxylakis	FORTH-ICS, Greece
Louis Marinos	ENISA, EU

Local Organizer

Nikolaos Petroulakis	FORTH-ICS, Greece

Workshop/Panel/Tutorial Chair

Damien Sauveron	XLIM, University of Limoges, France

Publicity Chairs

Claudio A. Ardagna	Università degli Studi di Milano, Italy
Mauro Conti	University of Padua, Italy

Steering Committee

Ioannis G. Askoxylakis	FORTH-ICS, Greece
Angelos Bilas	FORTH-ICS and University of Crete, Greece
Konstantinos Markantonakis	ISG-SCC, Royal Holloway University of London, UK
Joachim Posegga	Institute of IT-Security and Security Law, Germany
Jean-Jacques Quisquater	DICE, Catholic University of Louvain, Belgium
Damien Sauveron	XLIM, University of Limoges, France

Program Chairs

Lorenzo Cavallaro	Royal Holloway, University of London, UK
Dieter Gollmann	Hamburg University of Technology, Germany

Program Committee

Raja Naeem Akram	Edinburgh Napier University, UK
Claudio A. Ardagna	Università degli Studi di Milano, Italy
Ioannis G. Askoxylakis	FORTH-ICS, Greece
Lejla Batina	Radboud University Nijmegen, The Netherlands
Danilo Bruschi	Università degli Studi di Milano, Italy
Mauro Conti	University of Padua, Italy
Marco Cova	University of Birmingham, UK
Manuel Egele	Carnegie Mellon University, USA
Jaap-Henk Hoepman	Radboud University Nijmegen, The Netherlands
Andrea Lanzi	Insitut Eurecom, France
Corrado Leita	Symantec Research Labs, EU
Federico Maggi	Politecnico di Milano, Italy
Evangelos Markatos	FORTH-ICS, Greece
Lorenzo Martignoni	Google, Switzerland
Sjouke Mauw	University of Luxembourg, Luxembourg
Aikaterini Mitrokotsa	EPFL, Switzerland
Igor Muttik	McAfee Labs, UK
Flemming Nielson	Danish Technical University, Denmark
Wolter Pieters	TU Delft, The Netherlands
Christina Pöpper	ETH Zürich, Switzerland
Joachim Posegga	Institute of IT-Security and Security Law, Germany
Jean-Jacques Quisquater	DICE, Catholic University of Louvain, Belgium
William Robertson	Northeastern University, USA
Pierangela Samarati	Università degli Studi di Milano, Italy
Asia Slowinska	Vrije Universiteit Amsterdam, The Netherlands
Stefano Zanero	Politecnico di Milano, Italy
Jianying Zhou	Institute for Infocomm Research, Singapore

Additional Reviewers

Alessandro Barenghi	Istvan Haller	Maryna Krotofil
Bastian Braun	Jin Han	Mario Leone
Baris Ege	Wafa Ben Jaballah	Henrich C. Pöhls
Sara Foresti	Roman Kochanek	

Scientific Support

IFIP WG 11.2 Pervasive Systems Security

Main Sponsors

Since the early stages of inception of the workshop, organizers received positive feedback from a number of high-profile organizations. With the development of a strong Program and Organizing Committee, this was further capitalized into direct financial support. This enabled the workshop organizers to strengthen significantly their main objective for a high-standard academic workshop. The support helped significantly to keep the workshop registration costs as low as possible and at the same time offer a number of best paper awards.

We are wholeheartedly thankful to our Silver Sponsor Symantec Research Labs for supporting WISTP 2013.

Securing Critical Infrastructures: Challenges and Opportunities

Corrado Leita

Symantec Research Labs

Abstract. The threat landscape is continuously evolving. Large, wide spread worm infections are leaving more and more space to more stealthy attacks targeting highly valuable targets. Industrial control systems (ICS) are rapidly becoming a new major target of cyber-criminals: industrial control systems have converged with standard IT technologies and have brought powerful capabilities into the critical infrastructure environments, along with new and yet undiscovered threats. This was pointed out in multiple occasions over these last years and was confirmed by the discovery of highly sophisticated threats such as Stuxnet, that underlined a completely different threat model when compared to traditional malware witnessed in the wild in previous years. This talk will dive into the challenges and the opportunities associated to ICS security research, and on the tools at our disposal to improve our ability to protect such critical environments.

Near-Field Privacy

Lejla Batina

Radboud University Nijmegen

Abstract. With the expansion of versatile privacy-sensitive RFID applications a clear need for new identification schemes has been established. In particular, new attribute-based authentication schemes as reminiscences of Microsofts U-Prove technology were proposed. We survey related works and discuss a recent scheme for selective disclosure of attributes providing the designation of verification. A scenario of mobile payments is also considered and the use of NFC-enabled phones for proving credentials.

Table of Contents

Cryptography and Cryptanalysis

Mobile Security

Smart Cards and Embedded Devices

Multiplicative Homomorphic E-Auction with Formally Provable Security

Kun Peng and Matt Henricksen

Institute for Infocomm Research, Singapore
dr.kun.peng@gmail.com

Abstract. A new method, homomorphic e-auction based on multiplicative homomorphic encryption algorithm like ElGamal encryption is proposed in this paper. Its advantage is obvious and useful in practice: efficient distributed private key generation and thus efficient trust sharing. A long existing problem in homomorphic e-auction, inefficiency of bid validity check, is solved in the new multiplicative homomorphic e-auction scheme in this paper, which employs efficient bid re-formatting to enforce bid validity. Another contribution of the new multiplicative homomorphic e-auction scheme is that it is the first e-auction scheme to provide formal and comprehensive security analysis to achieve formally provable security (especially privacy).

1 Introduction

E-auction is a popular e-commerce application to distribute resources. In e-auction applications, the bids are often sealed for fairness and security. In many sealed-bid e-auction schemes, it is desired to protect privacy of the losing bids, which is called bid privacy. An obvious solution to protect bid privacy in e-auction is secure multiparty computation (called secure evaluation in [30]) as e-auction can be regarded as computation (evaluation) of some secret inputs (the bids) to obtain an output (the auction result). Secure-multiparty-computation-based solution to e-auction includes a few schemes [24,17,16,9,6,20]. As analysed in [30], these schemes are not efficient as they employ general multiparty computation techniques designed to evaluate any function. In comparison, special techniques designed to handle e-auction only are usually more efficient. A very popular such method is homomorphic bid opening. With this mechanism, each bidder employs a homomorphic encryption algorithm or a homomorphic secret sharing algorithm to seal their bids, while the auctioneers exploit homomorphism of the encryption algorithm or secret sharing algorithm to open the bids collectively instead of separately so that no losing bid is revealed. As the power of secret reconstruction or decryption is shared by multiple auctioneers such that the number of cooperating auctioneers must be over a threshold to gain the power, bid privacy is retained if the number of malicious auctioneers is not over the threshold.

Well known homomorphic e-auction schemes include [18,19,1,26,33,31,30,29]. They usually require that each bidder includes a bidding choice for every biddable price in his bid where every bidding choice must be one of two appointed

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 1–17, 2013.

integers, representing YES and NO respectively. To test whether there is a bid at a price, usually homomorphic bid opening schemes sum up all the bidders' bidding choices at that price. No separate bid choice is revealed and the sum is enough to show whether there is a bid at that price. Together with binary search for the wining price among the biddable prices, this summing-up bid opening mechanism is very efficient in finding the winning bid.

In the beginning, homomorphic e-auction schemes [18,19,33,30] employ Shamir's secret sharing [39] to seal the bids and exploit its homomorphism to implement homomorphic bid opening. As secret sharing and reconstruction have to be repeated multiple times in secret-sharing-based homomorphic e-auction, it is not efficient enough, especially when public verifiability is required and validity of secret sharing needs to be publicly verified. So homomorphic encryption algorithm becomes more popular and gradually replaces secret sharing in homomorphic e-auction [1,26,31,29] as a bid sealing tool.

The advantage of homomorphic encryption algorithm is obvious in homomorphic e-auction. Secret sharing only needs to be performed once to share the private key among the auctioneers. However, there is a difficulty in using homomorphic encryption in e-auction applications: the difficulty in threshold private key sharing for homomorphic algorithm. Most homomorphic e-auction schemes employ an additive homomorphic algorithm to seal the bids, where $D(c_1) + D(c_2) = D(c_1 c_2)$ for any ciphertexts c_1, c_2 and $D()$ denotes the decryption function. It is interesting to note that all the known additive homomorphic algorithms (e.g. [25,23,27]) employ factorization problem as a trapdoor. Although there exist some distributed key generation mechanisms for RSA [4,22,11], which is also factorization based, they (especially [4,22]) are inefficient. The distributed key generation technique in [11] improves efficiency to some extent by loosening the requirements on the parameters and using additional security assumptions, but is still inefficient compared to distributed key generation of DL based encryption algorithms [12,28,14] like ElGamal. So they cannot provide an efficient solution to distributed key generation for additive homomorphic encryption, not to mention their difficulty in public verification and that the relatively more efficient mechanism among them may not satisfy the parameter requirements with reasonable security assumptions when applied to e-auction. It is easy for a central trusted dealer to generate the private key [13,2,10] and then distribute it among the auctioneers. However, it requires too strong a trust and compromises the advantage of threshold trust, so is impractical in applications like e-auction. Although a modified ElGamal encryption in [21,35] is additive homomorphic and support efficient distributed key generation [12,28,14], it is not practical in most cases as it does not support efficient decryption.

Peng *et al* [30] design a special homomorphic e-auction scheme based on Goldwasser-Micali encryption, which is not additive homomorphic. As encryption and decryption operations are very efficient with Goldwasser-Micali encryption, their auction scheme needs few exponentiations with long exponents in computation. However, Goldwasser-Micali encryption depends on hardness of factorization problem as well, so suffers from lack of efficient distributed key generation as well.

Moreover, as the message space of Goldwasser-Micali encryption is only one bit long, bid opening must be repeated multiple times at any price in [30], which leads to two drawbacks in efficiency. Firstly, communicational cost is high. Secondly, a large number of multiplications are needed in computation.

A common efficiency bottleneck in homomorphic e-auction is bid validity check. Validity of homomorphic bid opening depends on two assumptions about validity of the bids. Firstly, each biding choice must represent either YES or NO. An attack called BBC attack is proposed in [31] to compromise correctness of auction when this condition is not satisfied. Secondly, each bidder chooses YES for all the biddable prices no higher than his offer and chooses NO for all the biddable prices higher than his offer. An attack called challenge attack is shown to work when this condition is not satisfied in Section 4. So validity of bids should be publicly proved and verified in secure homomorphic e-auction. Homomorphic e-auction schemes without bid validity check [18,19,33,30,31] are vulnerable to various attacks. For example, as explained in [30], colluding bidders in [18,19,33] can launch BBC attack. In [30,31], a malicious bidder can launch challenge attack as described in Section 4. Those attacks threatens fairness of the auction, so must be prevented. Some other homomorphic e-auction schemes [1,26] employ zero knowledge proof to prove validity of bids, but their proof is inefficient.

In [32,35], batch zero knowledge proof is employed to improve efficiency of bid validity check. Another batch proof technique is proposed in [8], which can be applied to bid validity check. However, their improvement in efficiency is not great enough to ease the efficiency concern in bid validity check. The most recent attempt to improve efficiency of bid validity check is [29], which allows the auctioneers to efficiently verify validity of the bids. However, it is not universally verifiable and one instance of proof can only convince one verifier. When there are other verifiers the bidders must provide a different proof to each of them. Moreover, it is required in [29] to extend the digital signature algorithm in [3] to distributed signature generation, where the power of signature generation is shared by the auctioneers. However, there is no known method to distribute signature generation for the digital signature algorithm in [3] as it is a special signature scheme to prevent malleability.

In this paper, a new homomorphic e-auction scheme is designed to overcome the two main drawbacks in the existing homomorphic e-auction schemes: lack of efficient distributed key generation and inefficiency of bid validity check. Firstly, a multiplicative homomorphic encryption algorithm, ElGamal encryption, is employed to seal the bids such that distributed key generation of the encryption algorithm can be efficiently implemented. Secondly, simpler operations are employed to replace bid validity proof and verification such that correctness of auction can be guaranteed even at the presence of malicious bidders. The new homomorphic e-auction scheme is designed in two steps. In the first step, efficient distributed key generation and multiplicative homomorphic bid opening are specified in Section 3 such that homomorphic bid opening at any price is always correct with an overwhelmingly large probability. In the second step,

the e-auction scheme is optimised in Section 4 to prevent the known attacks. A special contribution of this paper is that it presents the first formal and comprehensive security analysis for e-auction, while security (especially comprehensive privacy) of the existing e-auction schemes is intuitive only. Especially, privacy of the whole new e-auction scheme including all the published information is comprehensively and formally proved using a simulation-based model.

2 Symbols and Security Model

The most important security properties in e-auction are as follows.

- Correctness: the auction result is determined strictly according to the auction rule, while no bid is ignored or tampered with.
- Robustness: in abnormal situations (e.g. at presence of invalid bid), the auction can still run properly.
- Privacy: no secret information (e.g. the losing bid) except for the auction-result is revealed. More precisely, the auction transcript including all the published information in the auction can be simulated by a party without any secret knowledge but the auction result such that the simulating transcript is indistinguishable from the real auction transcript.
- Universal (public) verifiability: any one can publicly verify that no participant deviates from the auction protocol.

As the bids must be sealed to achieve bid privacy, privacy of e-auction at least depends on security of the employed sealing function (e.g. encryption algorithm) and no stronger privacy (e.g. unconditional privacy) is possible. We will formally illustrates that privacy of our new e-auction scheme only depends on semantic security of the employed encryption algorithm and a threshold trust in sharing the private key. The following symbols and parameters are used in this paper.

- There are m auctioneers $A_1, A_2, \ldots, A_m$ and n bidders $B_1, B_2, \ldots, B_n$.
- There are L biddable prices and they are denoted as $P_1, P_2, \ldots, P_L$ in descending order.
- p and q are large primes such that $p - 1 = 2q$. G is the cyclic subgroup in Z_p^* with order q and g is a generator of G.
- Integer l smaller than m is the trust threshold such that cooperation of at least l auctioneers is necessary to open any bid.
- L_1 and L_2 are security parameters smaller than q. They do not need to be very large on the condition that $2^{(L-1)L_2} < q$ and 2^{-L_1}, 2^{-L_2} are negligible.
- $H()$ is a one-way and collision-resistent hash function.

The formal proof to illustrate privacy of the new e-auction scheme in this paper needs to extend the traditional definition of semantic security [15] in Definition 1 to a new property called semantic security regarding multiple encryptions, which is defined in Definition 2 and illustrated to be a deduction of traditional semantic security in Theorem 1.

Definition 1. *(Traditional semantic security) An encryption algorithm with a random probabilistic encryption function $E()$ is semantically secure in the traditional definition if no polynomial adversary can win the following game with a probability non-negligibly larger than 0.5.*

1. *The adversary chooses two different messages m_0 and m_1 in the message space in any way he likes and sends them to a probabilistic encryption oracle.*
2. *The encryption oracle randomly chooses a bit I and returns $c = E(m_I)$.*
3. *Receiving c, the adversary wins the game if it finds out I.*

Definition 2. *A random probabilistic encryption function is semantically secure regarding multiple encryptions if no polynomial adversary can win the following game with a probability non-negligibly larger than 0.5.*

1. *Messages $m_1, m_2, \ldots, m_N$ with any possible distribution in the message space are given where $N \geq 1$.*
2. *Ciphertext $c_{0,1}, c_{0,2}, \ldots, c_{0,N}$ and $c_{1,1}, c_{1,2}, \ldots, c_{1,N}$ are given such that i is randomly chosen from $\{0, 1\}$ and $c_{i,j} = E(m_j)$ for $j = 1, 2, \ldots, N$ and every $c_{1-i,j}$ may be encryption of any message in the message space.*
3. *The adversary wins the game if it finds out i.*

Theorem 1. *If an encryption algorithm is semantically secure in the traditional definition, then it is semantically secure regarding multiple encryptions.*

Proof: If an encryption algorithm is not semantically secure regarding multiple encryptions, there is a polynomial algorithm A to win the game in Definition 2 with a probability non-negligibly larger than 0.5. A polynomial adversary can use A to break traditional semantic security of the encryption algorithm as follows.

1. Choosing m_0, m_1 and given c the adversary in Definition 1 needs to find I.
2. The adversary randomly chooses $m_2, m_3, \ldots, m_N$ from Z_q and generates two encryptions for each of them: $c_j = E(m_j)$ and $c'_j = E'(m_j)$ for $j = 2, 3, \ldots, N$ where $E'()$ denotes the same encryption algorithm using the same key but different probabilistic randomization from $E()$.
3. The adversary generates $c' = E(m_0)$.
4. The adversary submits messages $m_0, m_2, m_3, \ldots, m_N$ and ciphertexts $c', c'_2, c'_3, \ldots, c'_N$ and $c, c_2, c_3, \ldots, c_N$ to A.
5. A returns i and the adversary outputs $I = i$.

Note that

$$P[I = i] = P[I = i = 1] + P[I = i = 0]$$
$$= P[I = 1]P[i = 1|I = 1] + P[I = 0]P[i = 0|I = 0]$$

where $P[E]$ denotes the probability that event E happens and $P[E_1|E_2]$ denotes the probability that event E_1 happens on the condition that event E_2 happens. As A breaks semantic security regarding multiple encryptions, $P[i = 1|I = 1]$ is non-negligibly larger than 0.5 and denoted as ϵ. When $I = 0$, both

$c, c_2, c_3, \ldots, c_N$ and $c', c'_2, c'_3, \ldots, c'_N$ are encryptions of $m_0, m_2, m_3, \ldots, m_N$, so A outputs 0 or 1 with the same chance and thus $P[i = 0|I = 0] = 0.5$.

So

$$P[I = i] = 0.5\epsilon + 0.5 \times 0.5,$$

which is non-negligibly larger than 0.5. So the adversary can break traditional semantic security of the encryption algorithm in polynomial time by querying A. Since breaking semantic security regarding multiple encryptions implies breaking traditional semantic security, traditional semantic security implies semantic security regarding multiple encryptions. □

3 The Basic Protocol, Efficient Distributed Key Generation and Multiplicative Homomorphic Bid Opening

In this section, we start with a simple question: how to design homomorphic bid opening in first-bid sealed-bid auction, while the private key of the employed bid-sealing encryption algorithm must be generated in a distributed way to avoid too strong trust. Discussions of more complex questions will be given in Section 4 and Section 5. In our design, the auctioneers set up ElGamal encryption with distributed decryption for bid sealing as follows.

1. Each A_j randomly chooses x_j from Z_q and calculates $y_j = g^{x_j} \bmod p$ and publishes $H_j = H(y_j)$.
2. After $H_1, H_2, \ldots, H_m$ have been published, each A_j publishes y_j and any one can verify $H_j = H(y_j)$ for $j = 1, 2, \ldots, m$.
3. The public key of ElGamal encryption is $y = \prod_{j=1}^{m} y_j \bmod p$ and the private key $x = \sum_{j=1}^{m} x_j \bmod q$ needs to be secretly shared among the auctioneers with the trust threshold l.
4. Each A_j builds a polynomial $F_j(X) = \sum_{k=0}^{l-1} \mathbf{a}_{j,k} X^k$ where $\mathbf{a}_{j,0} = x_j$.
5. Each A_j calculates $s_{j,J} = F_j(J) \bmod q$ and secretly sends $s_{j,J}$ to every A_J for $j = 1, 2, \ldots, m$ and $J = 1, 2, , \ldots, m$.
6. Each A_J calculates his private key share $s_J = \sum_{j=1}^{m} s_{j,J} \bmod q$.

Public verification of the key distribution can be easily implemented by publishing $\alpha_{j,k} = g^{\mathbf{a}_{j,k}} \bmod p$ for $j = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, l-1$ and verifying each of the secret sharing operations using the existing publicly verifiable secret sharing techniques (e.g. [5,38]). Encryption and decryption are as follows.

- Encryption of a message M in G is $E(M) = (g^r \bmod p,\ My^r \bmod p)$ where r is randomly chosen from Z_q.
- A ciphertext $c = (a, b)$ can be decrypted by the cooperation of at least l auctioneers (denoted as $A_1, A_2, \ldots, A_l$ for simplicity of description) as follows.

1. Each auctioneers A_j calculates and publishes $\beta_j = a^{s_j} \bmod p$ for $j = 1, 2, \ldots, l$. If public verification is required, A_j publicly proves $\log_a \beta_j = \log_g \prod_{k=0}^{l-1} \theta_k^{j^k}$ using zero knowledge proof of equality of discrete logarithms [7] where $\theta_k = \prod_{j=1}^{m} \alpha_{j,k} \bmod p$.
2. The final result is $D(c) = b / \prod_{j=1}^{l} \beta_j^{u_j} \bmod p$ where $u_j = \prod_{1 \le v \le l, v \ne j} v/(v-j) \bmod q$.

Note that unlike the encryption algorithms employed in traditional homomorphic e-auction schemes ElGamal encryption is not additive homomorphic. Instead, it is multiplicative homomorphic. More precisely, in ElGamal encryption $D(c_1)D(c_2) = D(c_1c_2)$ for any ciphertexts c_1, c_2. So our homomorphic bid opening is different from the existing homomorphic bid opening mechanisms. It exploits multiplicative homomorphism instead of additive homomorphism of the employed encryption algorithm, so is called multiplicative homomorphic e-auction. One-choice-per-price strategy is employed in our design. The biddable prices are limited in a definite set and each bidder must make a choice (indicating willingness or unwillingness to pay) at every biddable price. If a bidder is willing to pay a price, he chooses an integer standing for "YES" as his choice at that price. If a bidder is unwilling to pay a price, he chooses an integer standing for "NO" as his choice at that price. The bidders seal their bidding vectors (including their choices at all the biddable prices) and publish the sealed bidding vectors. The detailed design of bid sealing is as follows.

1. Each bidder B_i chooses his evaluation P_{e_i} from $\{P_1, P_2, \ldots, P_L\}$.
2. Each B_i builds his bidding vector: $\mathbf{b}_i = (\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \ldots, \mathbf{b}_{i,L})$ where $\mathbf{b}_{i,t} = 1$ for $t < e_i$ and $\mathbf{b}_{i,t}$ is a random integer larger than 1 in G for $t \ge e_i$.
3. Each B_i seals his bidding vector in a ciphertext vector

$$c_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,L}) = (E(\mathbf{b}_{i,1}), E(\mathbf{b}_{i,2}), \ldots, E(\mathbf{b}_{i,L})).$$

Note that in our new design the bidders do not need to prove validity of any of his bidding choices as a bidding choice indicating "YES" can be any integer larger than 1. Namely, any integer in the messages space of the employed encryption algorithm is a valid bidding choice (1 for "NO" and otherwise for "YES"). It is illustrated in the following bid opening procedure that homomorphic bid opening can still work with such a tolerating bid format. The bid opening procedure employs binary search, where the biddable prices form a binary tree and the searching route starts at the tree root ($P_{L/2}$) and ends at a tree leaf. On each node of the route it is tested whether there is at least one "YES" choice without revealing the choices at that price. If there is one "YES" choice at that price, the search goes into the sub-tree with higher prices. If there is no "YES" choice at that price, the search goes into the sub-tree with lower prices. At every price P_ρ on the searching route the homomorphic bid opening is as follows.

1. The auctioneers cooperate to choose random integers $T_{\rho,1}, T_{\rho,2}, \ldots, T_{\rho,n}$ in $Z_{2^{L_1}}$ as follows.
   (a) Each A_j randomly chooses $T_{\rho,i,j}$ in $Z_{2^{L_1}}$ and publishes $T'_{\rho,i,j} = H(T_{\rho,i,j})$ for $i = 1, 2, \ldots, n$.
   (b) After all the $T'_{\rho,i,j}$s are published, each A_j publishes $T_{\rho,i,j}$ for $i = 1, 2, \ldots, n$.
   (c) $T_{\rho,i} = \sum_{j=1}^{m} T_{\rho,i,j} \bmod 2^{L_1}$ for $i = 1, 2, \ldots, n$.
2. The auctioneers calculate
$$C_\rho = (a_\rho, b_\rho) = \prod_{i=1}^{n} c_{i,\rho}^{T_{\rho,i}} = (\prod_{i=1}^{n} a_{i,\rho}^{T_{\rho,i}} \bmod p,\ \prod_{i=1}^{n} b_{i,\rho}^{T_{\rho,i}} \bmod p)$$
   where $c_{i,\rho}$ is denoted as $(a_{i,\rho}, b_{i,\rho})$.
3. An enough number of auctioneers cooperate to decrypt C_ρ into $d_\rho = D(C_\rho)$.

The binary search goes on until it stops at a leaf of the binary searching tree, which is declared as the winning price. Finally, the winner opens his bid (e.g. by publishing its encryption detail) to claim winning. It is illustrated in Theorem 2 that our homomorphic bid opening mechanism can work although any integer in G is a valid bidding choice.

Theorem 2. *Homomorphic bid opening at any price in the new multiplicative homomorphic e-auction is correct. More precisely, with an overwhelmingly large probability the decryption result at a price is larger than one iff there is at least one bidding choice larger than one at the price.*

Before Theorem 2 is proved, a lemma needs to be proved first.

Lemma 1. *Suppose $b_1, b_2, \ldots, b_N$ are integers in G. If $\prod_{i=1}^{N} b_i^{T_i} = 1 \bmod p$ with a probability larger than 2^{-L_1} for random L_1-bit integers $T_1, T_2, \ldots, T_N$, then $b_i = 1 \bmod p$ for $i = 1, 2, \ldots, N$.*

Proof: Given any integer k in $\{1, 2, \ldots, N\}$, there must exist integers $T_1, T_2, \ldots, T_{k-1}, T_{k+1}, \ldots, T_N$ in $\{0, 1, \ldots, 2^{L_1} - 1\}$ and two different integers T_k and $\hat{T}_k$ in $\{0, 1, \ldots, 2^{L_1} - 1\}$ such that the following two equations are correct.

$$\prod_{i=1}^{N} b_i^{T_i} = 1 \bmod p \tag{1}$$

$$(\prod_{i=1}^{k-1} b_i^{T_i}) b_k^{\hat{T}_k} \prod_{i=k+1}^{N} b_i^{T_i} = 1 \bmod p \tag{2}$$

Otherwise, for any L_1-bit integers $T_1, T_2, \ldots, T_{k-1}, T_{k+1}, \ldots, T_N$ there is at most one L_1-bit integer T_k to satisfy equation $\prod_{i=1}^{N} b_i^{T_i} = 1 \bmod p$, which implies that equation $\prod_{i=1}^{N} b_i^{T_i} = 1 \bmod p$ is satisfied with a probability no larger than 2^{-L_1} (with at most $2^{(N-1)L_1}$ combinations among the 2^{NL_1} possible combinations of $T_1, T_2, \ldots, T_N$) and is a contradiction.

(1) divided by (2) yields

$$b_k^{T_k - \hat{T}_k} = 1 \bmod p.$$

Note that T_k and $\hat{T}_k$ are L_1-bit integers, $2^{L_1} < q$ and q is prime, so $GCD(S_k - \hat{S}_k, q) = 1$. Therefore, $b_k = 1 \bmod p$. Also note that k can be any integer in $\{1, 2, \ldots, n\}$ and thus $b_i = 1 \bmod p$ for $i = 1, 2, \ldots, n$. □

Proof of Theorem 2: At a price P_ρ the result of multiplicative homomorphic bid opening is

$$d_\rho = D(C_\rho) = D(\prod_{i=1}^{n} c_{i,\rho}^{T_{\rho,i}}) = \prod_{i=1}^{n} D(c_{i,\rho})^{T_{\rho,i}} = \prod_{i=1}^{n} \mathbf{b}_{i,\rho}^{T_{\rho,i}} \bmod p.$$

- When all the bidding choices, $\mathbf{b}_{1,\rho}, \mathbf{b}_{2,\rho}, \ldots, \mathbf{b}_{n,\rho}$ are 1, d_ρ is always 1.
- According to Lemma 1, when any of $b_{1,\rho}, b_{2,\rho}, \ldots, b_{n,\rho}$ modulo p is larger than 1, d_ρ is larger than 1 with an overwhelmingly large probability. □

As traditional semantic security has been reduced to semantic security regarding multiple encryptions in Theorem 1, formal privacy of the new e-auction scheme can be proved in Theorem 3, which reduces distinguishability between the e-auction transcript and a simulating transcript to breaking semantic security regarding multiple encryptions as defined in Definition 2.

Theorem 3. *The new multiplicative homomorphic e-auction scheme is private and computationally reveals no bidding information other than the auction result. More precisely, the information revealed in the e-auction scheme can be simulated by a party without any knowledge of any bid but the auction result such that the simulating transcript is indistinguishable from the real transcript of the revealed information on the condition that ElGamal encryption algorithm is semantically secure regarding multiple encryptions.*

Proof: The revealed information in the new multiplicative homomorphic e-auction scheme includes:

- $c_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$;
- at each price p_ρ on the binary searching route C_ρ, d_ρ, $T_{\rho,i,j}$ and $T'_{\rho,i,j}$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$ and $T_{\rho,i}$ for $i = 1, 2, \ldots, n$;
- the integers published in the the multiple instances of underlying zero knowledge proof of equality of discrete logarithms, when complete public verifiability is required to include key generation and distributed decryption.

As ZK proof of equality of discrete logarithms in [7] has been formally proved to be zero knowledge, we only need to prove that $c_{i,t}$, C_ρ, d_ρ, $T_{\rho,i}$, $T_{\rho,i,j}$, $T'_{\rho,i,j}$ for p_ρ on the searching route, $1 \le i \le n$, $1 \le t \le L$ and $1 \le j \le m$ can be simulated by a party without any secret knowledge but the auction result. A party without any secret knowledge but the auction result can simulate them as follows.

1. He finds the binary searching route according to the auction result.
2. He randomly chooses $T_{\rho,i,j}$ for p_ρ on the searching route, $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$ from $Z_{2^{L_1}}$ and calculates $T'_{\rho,i,j} = H(T_{\rho,i,j})$.
3. He calculates $T_{\rho,i} = \sum_{j=1}^{m} T_{\rho,i,j} \bmod 2^{L_1}$ for p_ρ on the searching route and $i = 1, 2, \ldots, n$.
4. He randomly chooses $\mathbf{b}_{i,t}$ from G for $1 \le i \le n$ and $1 \le t \le L$.
5. He calculates $c_{i,t} = (a_{i,t}, b_{i,t}) = (g^{r'_{i,t}} \bmod p,\ \mathbf{b}_{i,t} y^{r'_{i,t}} \bmod p)$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$ where $r'_{i,t}$ is randomly chosen from Z_q.

6. He calculates $C_\rho = (\prod_{i=1}^{n} a_{i,\rho}^{T_{\rho,i}} \bmod p,\ \prod_{i=1}^{n} b_{i,\rho}^{T_{\rho,i}} \bmod p)$ for p_ρ on the searching route.
7. He calculates $d_\rho = \prod_{i=1}^{n} \mathbf{b}_{i,\rho}^{T_{\rho,i}} \bmod p$ for p_ρ on the searching route.

In this simulating transcript of the revealed information,

- each $T_{\rho,i,j}$ is uniformly distributed in $Z_{2^{L-1}}$;
- each $T_{\rho,i}$ is uniformly distributed in $Z_{2^{L-1}}$;
- each $c_{i,t}$ is distributed in G^2 in a way independent of the secret bids but depending on how $\mathbf{b}_{i,t}$ is chosen in the simulation;
- each d_ρ is uniformly distributed in G;
- each C_ρ is uniformly distributed in G^2;
- $T_{\rho,i} = \sum_{j=1}^{m} T_{\rho,i,j} \bmod 2^{L_1}$ for p_ρ on the searching route and $1 \le i \le n$;
- $T'_{\rho,i,j} = H(T_{\rho,i,j})$ for p_ρ on the searching route ,$1 \le j \le m$ and $1 \le i \le n$;
- $C_\rho = (\prod_{i=1}^{n} a_{i,\rho}^{T_{\rho,i}} \bmod p,\ \prod_{i=1}^{n} b_{i,\rho}^{T_{\rho,i}} \bmod p)$ for p_ρ on the searching route;
- $d_\rho = \prod_{i=1}^{n} \mathbf{b}_{i,\rho}^{T_{\rho,i}} \bmod p$ and $d_\rho = D(C_\rho)$ for p_ρ on the searching route.

In comparison, in the real transcript of the same information,

- each $T_{\rho,i,j}$ is uniformly distributed in $Z_{2^{L-1}}$;
- each $T_{\rho,i}$ is uniformly distributed in $Z_{2^{L-1}}$;
- each $c_{i,t}$ encrypts B_i's choice at p_t;
- each d_ρ is uniformly distributed in G;
- each C_ρ is uniformly distributed in G^2;
- $T_{\rho,i} = \sum_{j=1}^{m} T_{\rho,i,j} \bmod 2^{L_1}$ for p_ρ on the searching route and $1 \le i \le n$;
- $T'_{\rho,i,j} = H(T_{\rho,i,j})$ for p_ρ on the searching route ,$1 \le j \le m$ and $1 \le i \le n$;
- $C_\rho = (\prod_{i=1}^{n} a_{i,\rho}^{T_{\rho,i}} \bmod p,\ \prod_{i=1}^{n} b_{i,\rho}^{T_{\rho,i}} \bmod p)$ for p_ρ on the searching route;
- $d_\rho = \prod_{i=1}^{n} \mathbf{b}_{i,\rho}^{T_{\rho,i}} \bmod p$ and $d_\rho = D(C_\rho)$ for p_ρ on the searching route.

The only difference between the two transcripts lies in distribution of $c_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$. If a polynomial adversary can distinguish the two transcripts of $c_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$, it can be employed to break semantic security of ElGamal encryption regarding multiple encryptions as follows. Since the the adversary can distinguish the two transcripts of $c_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$ without any other information, given additional information $\mathbf{b}_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$, it can still distinguish the two transcripts as the additional information only makes the distinguishing work easier (or at least does not make it harder). So given messages $\mathbf{b}_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$ and two sets of ciphertexts $c_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$, one of which is encryption of $\mathbf{b}_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$, the adversary can distinguish which set of ciphertexts are in the real e-auction transcript and thus encryption of $\mathbf{b}_{i,t}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$. Namely, the adversary breaks semantic security of ElGamal encryption regarding multiple encryptions. □

According to Theorem 1 and Theorem 3, when the number of colluding malicious auctioneers is not over l, compromising privacy of the new e-auction scheme implies breaking semantic security of ElGamal encryption, which is widely known to be hard assuming hardness of the Decisional Diffie-Hellman problem. So the new e-auction scheme is private on the condition that the DDH problem is hard.

4 The Final Protocol, Optimisation to Achieve Robustness

Although the multiplicative homomorphic bid opening mechanism in Section 3 can work at any price, achieving correctness in the auction still needs an assumption: each bidder submits a 1 as his bidding choice at any price higher than his evaluation and an integer larger than 1 as his bidding choice at any price no higher than his evaluation. More precisely, although it is not needed to assume validity of every single bidding choice in multiplicative homomorphic e-auction (as any bidding choice in the message place of the employed encryption algorithm is valid), it is still necessary to assume that in each bid vector 1s are at the higher prices and larger integers are at the lower prices. If this assumption is not satisfied, the auction scheme is still vulnerable to some attacks. For example, a malicious bidder can submit YES at higher prices and submit NO at lower prices to launch a challenge attack (mentioned in Section 1), which enables him to dispute the auction result like attacking [30,31] as follows.

1. A malicious bidder includes in his bid vector a larger integer at price P_γ and 1s at lower prices.
2. In the binary search for winning bid, multiplicative homomorphic bid opening is performed at one price lower than P_γ and returns a decryption result 1. So the binary search goes on to lower prices and finally stops at a winning price lower than P_γ.
3. After the auction result is published, the malicious bidder can choose to challenge validity of the result by publishing his bidding choice at P_γ if he likes (e.g. if his colluding co-bidder does not win or he is not satisfied with the auction result for other reasons). His challenge is effective as he does submit YES at a price higher than the winning bid.

This attack obviously violates robustness of auction. Can the malicious bidder be denied of his winning or penalized or kicked out? It is a complex question. Note that in an open-cry auction, a bidder is usually allowed to keep silent at a lower price but bid at a higher price later. Then why is the bid invalid in sealed-bid auction while it is accepted in the bidding phase? If the challenge is acceptable, the malicious bidder compromises fairness of the auction and takes advantage of the other bidders. Even if the challenge can be clarified and rejected, the clarification needs to open the bidding choices separately and compromises privacy of the auction.

Our countermeasure to this attack is simple: before bid opening the auctioneers re-format the encrypted bids such that in each bid if there is a bidding choice larger than 1 all the other bidding choices at lower prices in the same bid vector are larger than 1 with an overwhelmingly large probability. It is described in details as follows.

1. The bidders still seal and submit their bids as in Section 3. Namely each bidder B_i builds his bidding vector: $\mathbf{b}_i = (\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \ldots, \mathbf{b}_{i,L})$ and seals it in a ciphertext vector $c_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,L})$

2. The auctioneers cooperate to choose random integers $S_{i,t}$ in $Z_{2^{L_2}}$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L-1$ just like they choose $T_{i,t}$ in Section 3.
3. The auctioneers calculate

$$c'_{i,t} = {c'}_{i,t-1}^{S_{i,t-1}} c_{i,t} = ({a'}_{i,t-1}^{S_{i,t-1}} a_{i,t} \bmod p,\ {b'}_{i,t-1}^{S_{i,t-1}} b_{i,t} \bmod p)$$
$$\text{for } i = 1, 2, \ldots, n \text{ and } t = 2, 3, \ldots, L$$

where $c'_{i,1} = c_{i,1}$, $c_{i,t}$ is denoted as $(a_{i,t}, b_{i,t})$ and $c'_{i,t}$ is denoted as $(a'_{i,t}, b'_{i,t})$.

After the re-formating, the encrypted bidding choices become

$$c'_{i,t} = ((\prod_{K=1}^{t-1} a_{i,K}^{\prod_{J=K}^{t-1} S_{i,J}}) a_{i,t} \bmod p, (\prod_{K=1}^{t-1} b_{i,K}^{\prod_{J=K}^{t-1} S_{i,J}}) b_{i,t} \bmod p) \quad (3)$$
$$\text{for } i = 1, 2, \ldots, n \text{ and } t = 2, 3, \ldots, L.$$

In this new bid format, every $c'_{i,t}$ encrypts an integer larger than 1 with an overwhelmingly large probability as illustrated in Theorem 4 if any of the bidding choices at higher prices in the same bid vector is larger than 1.

Theorem 4. *If any of $D(c_{i,1}), D(c_{i,2}), \ldots, D(c_{i,t-1})$ is larger than 1 for any t in $\{2, 3, \ldots, n\}$, then $D(c'_{i,t}) > 1$ with a probability no smaller than $1 - 2^{-L_2}$.*

Before Theorem 4 is proved, a lemma is proved first.

Lemma 2. *Suppose $b_1, b_2, \ldots, b_N$ are integers in G. If $b_N \prod_{i=1}^{N-1} b_i^{S_i} = 1 \bmod p$ with a probability larger than 2^{-L_2} where $S_1, S_2, \ldots, S_{N-1}$ are random integers at least L_2 bits long and smaller than q, then $b_i = 1 \bmod p$ for $i = 1, 2, \ldots, N-1$.*

Proof: Given any integer k in $\{1, 2, \ldots, N-1\}$, there must exist one instance of integers $S_1, S_2, \ldots, S_{k-1}, S_{k+1}, \ldots, S_{N-1}$ among all their possible choices and two different instances for the choice of S_k, denoted as S_k and $\hat{S}_k$, such that the following two equations are correct.

$$b_N \prod_{i=1}^{N-1} b_i^{S_i} = 1 \bmod p \quad (4)$$
$$b_N (\prod_{i=1}^{k-1} b_i^{S_i}) b_k^{\hat{S}_k} \prod_{i=k+1}^{N-1} b_i^{S_i} = 1 \bmod p \quad (5)$$

Otherwise, for any possible choice of integers $S_1, S_2, \ldots, S_{k-1}, S_{k+1}, \ldots, S_N$ there is at most one choice for integer S_k among all its possible choices to satisfy equation $b_N \prod_{i=1}^{N-1} b_i^{S_i} = 1 \bmod p$, which implies equation $b_N \prod_{i=1}^{N-1} b_i^{S_i} = 1 \bmod p$ is satisfied with a probability no larger than 2^{-L_2} (as the number of possible choices for S_k is at least 2^{L_2}) and is a contradiction.

(4) divided by (5) yields

$$b_k^{S_k - \hat{S}_k} = 1 \bmod p.$$

Note that S_k and $\hat{S}_k$ are smaller than q and q is prime, so $GCD(S_k - \hat{S}_k, q) = 1$. Therefore, $b_k = 1 \bmod p$. Also note that k can be any integer in $\{1, 2, \ldots, N-1\}$ and thus $b_i = 1 \bmod p$ for $i = 1, 2, \ldots, N-1$. □

Proof of Theorem 4: (3) and multiplicative homomorphism of ElGamal encryption imply

$$D(c'_{i,t}) = (\prod_{K=1}^{t-1} D(c_{i,K})^{\prod_{J=K}^{t-1} S_{i,J}}) D(c_{i,t}) \bmod p$$
$$\text{for } i = 1, 2, \ldots, n \text{ and } t = 2, 3, \ldots, L.$$

Note that any $\prod_{J=K}^{t-1} S_{i,J}$ is smaller than q as $2^{(L-1)L_2} < q$. If $D(c'_{i,t}) > 1$ is not guaranteed with a probability no smaller than $1 - 2^{-L_2}$ for any t, then $D(c'_{i,t}) = 1$ with a probability larger than 2^{-L_2}. So according to Lemma 2, all of $D(c_{i,1}), D(c_{i,2}), \ldots, D(c_{i,t-1})$ are 1s, which is contradictory to the fact that at least one of $D(c_{i,1}), D(c_{i,2}), \ldots, D(c_{i,t-1})$ is larger than 1. Therefore, $D(c'_{i,t}) > 1$ must be guaranteed with a probability no smaller than $1 - 2^{-L_2}$. □

As the re-formatted encrypted bids are enforced to contain consistent bidding choices with an overwhelmingly large probability, multiplicative homomorphic bid opening and binary search can be performed on them and challenge attack can be prevented. Moreover, as the bidding choices except those at the top price in all the bids are already randomized, multiplicative homomorphic bid opening can actually be simplified and the randomization through raising the encrypted bidding choices to the power of $T_{i,t}$ in Section 3 can be removed unless multiplicative homomorphic bid opening is performed at the top price. The simplified bid opening operation is as follows.

1. The auctioneers reformat the encrypted bids as detailed earlier in this section and obtain the re-formatted encrypted bids $c'_{i,t} = (a'_{i,t}, b'_{i,t})$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, L$.
2. The auctioneers perform binary search for the winning bid. If the binary search goes to the top price P_1, the multiplicative homomorphic bid opening at P_1 is the same as described in Section 3. Otherwise, the multiplicative homomorphic bid opening at any price P_ρ is as follows.
   (a) The auctioneers calculate
   $$C_\rho = \prod_{i=1}^{n} c'_{i,\rho} = (\prod_{i=1}^{n} a'_{i,\rho} \bmod p, \prod_{i=1}^{n} b'_{i,\rho} \bmod p).$$
   (b) An enough number of them cooperate to decrypt C_ρ into $d_\rho = D(C_\rho)$ as described in Section 3.
   (c) If $d_\rho > 1$, the search goes into the sub-tree with prices higher than P_ρ. If $d_\rho = 1$, the search goes into the sub-tree with prices lower than P_ρ.
3. The binary search goes on until it stops at a leaf of the binary searching tree, which is declared as the winning price. Finally, the winner opens his bid to claim winning.

In comparison with the original e-auction scheme in Section 3, the optimisation in this section changes the way the bidding choices are randomized. The new randomization operation not only enables multiplicative homomorphic bid opening but also enforces validity of the bids. The optimised e-auction scheme is correct and private as illustrated in Theorem 2 and Theorem 3 (since the change in randomization operation does not affect their applicability to the e-auction scheme) and its robustness is guaranteed by Theorem 4.

Table 1. Security comparison of homomorphic e-auction schemes

Auction schemes	Bid opening power sharing	Bid validity check	Universal verifiability	Vulnerability or problem	Correctness & privacy
[18]	nL instances of secret sharing	no	yes	BBC attack and challenge attack	intuitive
[19]	nL instances of secret sharing	no	yes	BBC attack and challenge attack	intuitive
[33]	nL instances of secret sharing	no	yes	BBC attack and challenge attack	intuitive
[1]	sharing 1 key only but no efficient distributed key generation	yes	yes	no	intuitive
[26]	sharing 1 key only but no efficient distributed key generation	yes	yes	no	intuitive
[30]	sharing 1 key only but no efficient distributed key generation	no	yes	challenge attack	intuitive
[31]	nL instances of secret sharing	no	yes	challenge attack	intuitive
[29]	sharing 1 key only but no efficient distributed key generation	yes	no	unknown how to generate Boneh signature [3] in a distributed way	intuitive
New	sharing 1 key only efficient distributed key generation	enforcing validity by bid re-formatting	yes	no	formally proved

Table 2. Efficiency comparison of secure homomorphic e-auction schemes

Auction	Bidder		Auctioneer	
schemes	multiplication	example	multiplication	example
[1,26]	$\geq 12291.5L$	50346140	$\geq 12292nL + (10752 + 2n)\log_2 L$	50348957633
[29]	$1536(2L + 8)$ per verifier	12595200 per verifier	$1536(0.2nL + 20\log_2 L + 16n)$	1283297280
New	$3072L$	12582912	$3L_2n(L-1) + 4608\log_2 L$	368605296

5 Comparison and Conclusion

Security and efficiency of the new e-auction scheme is compared with the existing e-auction schemes with bid privacy in this section. As explained in Section 1, secure-multiparty-computation-based e-auction schemes [24,17,16,9,6,20] and e-auction schemes employing downward search [37,40,41,36,34] are less efficient and so we focus our comparison on homomorphic e-auction. Firstly, comparison of security properties is given in Table 1. Then, efficiency comparison of secure homomorphic e-auction schemes (with bid validity check and invulnerable to known attacks) is given in Table 2, where the number of multiplications is counted and for simplicity an exponentiation with a **L**-bit exponent is counted as 1.5**L** multiplications. A full-length exponent in cryptographic operations in G or Z_p is supposed to be 1024 bits long. Note that like the analysis in the existing e-auction schemes, cost of preparation work (e.g. distributed key generation) is not included in Table 2. If it is taken into account, advantage of our new scheme will be greater as it is the only homomorphic e-auction scheme with efficient distributed key generation. In the example in Table 2, $n = 1000$ and $L = 4096$, while $L_2 = 30$ such that 2^{-L_2} is smaller than one out of one billion.

The comparisons clearly demonstrate that the new e-auction scheme is secure and efficient. It satisfies all the security properties and is the only e-auction scheme to achieve formally provable security. It is much more efficient than any secure homomorphic e-auction scheme, which already employs a relatively more efficient solution to secure e-auction. Extending our technique to more complex auction rules is an open question.

References

1. Abe, M., Suzuki, K.: M+1-st price auction using homomorphic encryption. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 115–124. Springer, Heidelberg (2002)
2. Baudron, O., Fouque, P., Pointcheval, D., Poupard, G., Stern, J.: Practical multi-candidate election system. In: ACM PODC 2001, pp. 274–283 (2001)
3. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
4. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg (1997)
5. Boudot, F., Traoré, J.: Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 87–102. Springer, Heidelberg (1999)
6. Cachin, C.: Efficient private bidding and auctions with an oblivious third party. In: ACM CCS 1999, pp. 120–127 (1999)
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
8. Chida, K., Yamamoto, G.: Batch processing for proofs of partial knowledge and its applications. IEICE Trans. Fundamentals, 150–159 (January 2008)
9. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)
10. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001)
11. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (2001)
12. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: FOCS 1987, pp. 427–437 (1987)
13. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 90–104. Springer, Heidelberg (2001)
14. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (1999)
15. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer Security 28(2), 270–299 (1984)

16. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 162–177. Springer, Heidelberg (2000)
17. Juels, A., Szydlo, M.: A two-server, sealed-bid auction protocol. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 72–86. Springer, Heidelberg (2003)
18. Kikuchi, H., Harkavy, M., Tygar, J.: Multi-round anonymous auction. In: IEEE WDRES 1998, pp. 62–69 (1998)
19. Kikuchi, H., Hotta, S., Abe, K., Nakanishi, S.: Distributed auction servers resolving winner and winning bid without revealing privacy of bids. In: IEEE Workshop on Next Generation Internet 2000, pp. 307–312 (2000)
20. Kurosawa, K., Ogata, W.: Bit-slice auction circuit. In: Gollmann, D., Karjoth, G., Waidner, M. (eds.) ESORICS 2002. LNCS, vol. 2502, pp. 24–38. Springer, Heidelberg (2002)
21. Lee, B., Kim, K.: Receipt-free electronic voting through collaboration of voter and honest verifier. In: JW-ISC 2000, pp. 101–108 (2000)
22. MacKenzie, P., Frankel, Y., Yung, M.: Robust efficient distributed RSA-key generation. In: STOC 1998, p. 320 (1998)
23. Naccache, D., Stern, J.: A new public key cryptosystem based on higher residues. In: ACM Computer Science Conference 1998, pp. 160–174 (1998)
24. Naor, M., Pinkas, B., Sumner, R.: Privacy perserving auctions and mechanism design. In: ACM Conference on Electronic Commerce 1999, pp. 129–139 (1999)
25. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 308–318. Springer, Heidelberg (1998)
26. Omote, K., Miyaji, A.: A second-price sealed-bid auction with verifiable discriminant of p_0-th root. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 57–71. Springer, Heidelberg (2003)
27. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
28. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991)
29. Peng, K., Bao, F.: Efficiency improvement of homomorphic E-auction. In: Katsikas, S., Lopez, J., Soriano, M. (eds.) TrustBus 2010. LNCS, vol. 6264, pp. 238–249. Springer, Heidelberg (2010)
30. Peng, K., Boyd, C., Dawson, E.: A multiplicative homomorphic sealed-bid auction based on Goldwasser-Micali encryption. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 374–388. Springer, Heidelberg (2005)
31. Peng, K., Boyd, C., Dawson, E.: Optimization of electronic first-bid sealed-bid auction based on homomorphic secret sharing. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 84–98. Springer, Heidelberg (2005)
32. Peng, K., Boyd, C., Dawson, E.: Batch verification of validity of bids in homomorphic e-auction. Computer Communications 29, 2798–2805 (2006)
33. Peng, K., Boyd, C., Dawson, E., Viswanathan, K.: Robust, privacy protecting and publicly verifiable sealed-bid auction. In: Deng, R.H., Qing, S., Bao, F., Zhou, J. (eds.) ICICS 2002. LNCS, vol. 2513, pp. 147–159. Springer, Heidelberg (2002)
34. Peng, K., Boyd, C., Dawson, E., Viswanathan, K.: Non-interactive auction scheme with strong privacy. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 407–420. Springer, Heidelberg (2003)

35. Peng, K., Dawson, E.: Efficient Bid Validity Check in ElGamal-Based Sealed-Bid E-Auction. In: Dawson, E., Wong, D.S. (eds.) ISPEC 2007. LNCS, vol. 4464, pp. 209–224. Springer, Heidelberg (2007)
36. Sako, K.: An auction protocol which hides bids of losers. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 422–432. Springer, Heidelberg (2000)
37. Sakurai, K., Miyazaki, S.: A bulletin-board based digital auction scheme with bidding down strategy. In: IWCTE 1999, pp. 180–187 (1999)
38. Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 148–164. Springer, Heidelberg (1999)
39. Shamir, A.: How to share a secret. Communication of the ACM 22(11), 612–613 (1979)
40. Suzuki, K., Kobayashi, K., Morita, H.: Efficient sealed-bid auction using hash chain. In: Won, D. (ed.) ICISC 2000. LNCS, vol. 2015, pp. 183–191. Springer, Heidelberg (2001)
41. Watanabe, Y., Imai, H.: Reducing the round complexity of a sealed-bid auction protocol with an off-line ttp. In: STOC 2000, pp. 80–86 (2000)

Malleable Signatures for Resource Constrained Platforms

Henrich C. Pöhls[1,3,⋆], Stefan Peters[3], Kai Samelin[2,3,⋆⋆], Joachim Posegga[1,3], and Hermann de Meer[2,3]

[1] Chair of IT-Security
[2] Chair of Computer Networks and Computer Communication
[3] Institute of IT-Security and Security Law (ISL), University of Passau, Germany
{hp,ks,jp}@sec.uni-passau.de, peters_stefan@gmx.net, demeer@fim.uni-passau.de

Abstract. Malleable signatures allow the signer to control alterations to a signed document. The signer limits alterations to certain parties and to certain parts defined during signature generation. Admissible alterations do not invalidate the signature and do not involve the signer. These properties make them a versatile tool for several application domains, like e-business and health care. We implemented one secure redactable and three secure sanitizable signature schemes on secure, but computationally bounded, smart card. This allows for a secure and practically usable key management and meets legal standards of EU legislation. To gain speed we securely divided the computing tasks between the powerful host and the card; and we devise a new accumulator to yield a useable redactable scheme. The performance analysis of the four schemes shows only a small performance hit by the use of an off-the-shelf card.

1 Introduction

Digital signatures are technical measures to protect the integrity and authenticity of data. Classical digital schemes that can be used as electronic signatures must detect any change that occurred after the signature's generation. Digital signatures schemes that fulfill this are *unforgeable*, such as RSA-PSS. In some cases, controlled changes of signed data are required, e.g., if medical health records need to be sanitized before being made available to scientists. These *allowed* and *signer-controlled* modifications must not result in an invalid signature and must not involve the signer. This rules out re-signing changed data or changes applied to the original data by the signer. *Miyazaki* et al. called this constellation the "digital document sanitization problem" [20]. Cryptographic solutions to this problem are

⋆ Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project.
⋆⋆ The research leading to these results was supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (http://www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence (grant agreement no. [288021]).

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 18–33, 2013.

sanitizable signatures (SSS) [2] or redactable signatures (RSS) [15]. These have been shown to solve a wide range of situations from secure routing or anonymization of medical data [2] to e-business settings [22,23,28]. For a secure and practically usable key management, we implemented four malleable signature schemes on an off-the-shelf smart card. Hence, all the algorithms that involve a parties secret key run on the smart card of that party. Smart cards are assumed secure storage and computation devices which allow to perform these actions while the secret never leaves the card's protected computing environment. However, they are computationally bounded.

1.1 Contribution

To the best of our knowledge, no work on how to implement these schemes on resource constraint platforms like smart cards exists. Additional challenges are sufficient speed and low costs. Foremost, the smart card implementation must be reasonably fast and manage all the secrets involved on a resource constraint device. Secondly, the implementation should run on off-the-shelf smart cards; cheaper cards only offer fast modular arithmetics (e.g., needed for RSA signatures). The paper's three core contribution are the:

(1) analysis and selection of suitable and secure schemes;
(2) implementation of three SSSs and one RSS scheme to measure runtimes;
(3) construction of a provably secure RSS based on our newly devised accumulator with a semi-trusted third party.

Previously only accumulators with fully-trusted setups where usably fast. This paper shows how to relax this requirement to a semi-trusted setup. Malleable signatures on smart cards allow fulfilling the legal requirement of keeping keys in a "secure signature creation device" [12].

1.2 Overview and State of the Art of Malleable Signatures

With a classical signature scheme, *Alice* generates a signature σ using her private key sk_{sig} and the SSign algorithm. *Bob*, as a verifier, uses *Alice*'s public key pk_{sig} to verify the signature on the given message m. Hence, the authenticity and integrity of m is verified. Assume *Alice*'s message m is composed of a uniquely reversible concatenation of ℓ blocks, i.e., $m = (m[1], m[2], \ldots, m[\ell])$. When *Alice* uses a RSS, it allows that every third party can **redact** a block $m[i] \in \{0,1\}^*$. To *redact* $m[i]$ from m means creating a m' without $m[i]$, i.e., $m' = (\ldots, m[i-1], m[i+1], \ldots)$. Redacting further requires that the third-party is also able to compute a new valid signature σ' for m' that verifies under *Alice*'s public key pk_{sig}. Contrary, in an SSS, *Alice* decides for each block $m[i]$ whether **sanitization** by a designated third party, denoted `Sanitizer`, is admissible or not. *Sanitization* means that $\texttt{Sanitizer}^i$ can replace each admissible block $m[i]$ with an arbitrary string $m[i]' \in \{0,1\}^*$ and hereby creates a modified message $m' = (\ldots, m[i-1], m[i]', m[i+1], \ldots)$. In comparison to RSSs, sanitization requires a secret, denoted as sk_{san}, to derive a new signature σ', such that (m', σ') verifies under the given public keys.

A secure RSS or SSS must at least be *unforgeable* and *private*. *Unforgeability* is comparable to classic digital signature schemes allowing only controlled modifications. Hence, a positive verification of m' by *Bob* means that all parts of m' are authentic, i.e., they have not been altered in a malicious way. *Privacy* inhibits a third party from learning anything about the original message, e.g., from a signed redacted medical record, one cannot retrieve any additional information besides what is present in the given redacted record.

The concept behind RSSs has been introduced by *Steinfeld* et al. [27] and by *Johnson* et al. [15]. The term SSS has been coined by *Ateniese* et al. [2].

Brzuska et al. formalized the standard security properties of SSSs [5]. RSSs were formalized for lists by *Samelin* et al. [25]. We follow the nomenclatures of *Brzuska* et al. [5]. If possible, we combine explanations of RSSs and SSSs to indicate relations. In line with existing work we assume the signed message m to be split in blocks $m[i]$, indexed by their position. W.l.o.g., we limit the algorithmic descriptions in this paper to simple structures to increase readability. Algorithms can be adapted to work on other data-structures. We keep our notation of `Sanitizer` general, and also cater for multiple sanitizers, denoted as $\texttt{Sanitizer}^i$ [10]. Currently, there are no *implementations* of malleable signatures considering multi-sanitizer environments.

A related concept are proxy signatures [18]. However, they only allow generating signatures, not controlled modifications. We therefore do not discuss them anymore. For implementation details on resource constrained devices, refer to [21].

1.3 Applications of Malleable Signatures

One reason to use malleable signatures is the unchanged root of trust: the verifier only needs to trust the signer's public key. Authorized modifications are specifically endorsed by the signer in the signature and subsequent signature verification establishes if none or only authorized changes have occurred. In the e-business setting, SSS allows to control the change and to establish trust for intermediary entities, as explained by *Tan* and *Deng* in [28]. They consider three parties (*manufacturer*, *distributor* and *dispatcher*) that carry out the production and the delivery to a forth party, the *retailer*. The *distributor* produces a malleable signature on the document and the *manufacturer* and *dispatcher* become sanitizers.Due to the SSS, the *manufacturer* can add the product's serial number and the *dispatcher* adds shipment costs. The additions can be done without involvement of the *distributor*. Later, the *retailer* is able to verify all the signed information as authentic needing only to trust the *distributor*. Legally binding digital signatures must detect "any subsequent change" [12], a scheme by *Brzuska* et al. was devised to especially offer this *public accountability* [8].

Another reason to use a malleable signature scheme is their ability to sign a large data set once, and then to only partly release this information while retaining verifiability. This privacy notion allows their application in healthcare environments as explained by *Ateniese* et al. [2]. For protecting trade secrets and for data protection it is of paramount important to use a *private* scheme.

Applications that require to hide the fact that a sanitization or redaction has taken place must use schemes that offer *transparency*, which is stronger than privacy [5]. However, the scheme described by *Tan* and *Deng* is not private according to the state-of-the-art cryptographic strict definition [5].

1.4 Motivation for Smart Cards

To facilitate RSSs and SSSs in practical applications, they need to achieve the same level of integrity and authenticity assurance as current standard digital signatures. This requires them to be *unforgeable* while being linkable to the legal entity that created the signature on the document. To become fully recognized by law, i.e., to be legally equivalent to hand-written signatures, the signature needs to be created by a "secure signature creation device" (SSCD) [12]. Smart cards serve as such an SSCD [19]. They allow for using a secret key, while providing a high assurance that the secret key does not leave the confined environment of the smart card. Hence, smart cards help to close the gap and make malleable signatures applicable for deployment in real applications. State of the art secure RSSs and SSSs detect all modifications not endorsed by the signer as forgeries. Moreover, *Brzuska* et al. present a construction in [8] and show that their construction fulfills EU's legal requirements [22].

2 Sanitizable and Redactable Signature Schemes

We assume the verifier trusts and possesses the `Signer`'s public key pk_{sig} and can reconstruct all other necessary information from the message-signature pair (m, σ) alone. Existing schemes have the following polynomial time algorithms:

$$\mathsf{SSS} := (\mathsf{KGen}_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}_{\mathsf{SSS}}, \mathsf{Sanit}_{\mathsf{SSS}}, \mathsf{Verify}_{\mathsf{SSS}}, \mathsf{Proof}_{\mathsf{SSS}}, \mathsf{Judge}_{\mathsf{SSS}})$$
$$\mathsf{RSS} := (\mathsf{KGen}_{\text{sig}}, \mathsf{Sign}_{\mathsf{RSS}}, \mathsf{Verify}_{\mathsf{RSS}}, \mathsf{Redact}_{\mathsf{RSS}})$$

Key Generation (SSS, RSS). Generates key pairs. Only SSSs need $\mathsf{KGen}_{\text{san}}$.

$$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda), \qquad (pk^i_{\text{san}}, sk^i_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$$

Signing (SSS, RSS). Requires the `Signer`'s secret key sk_{sig}. For $\mathsf{Sign}_{\mathsf{SSS}}$, it additionally requires all sanitizers' public keys $\{pk^1_{\text{san}}, \ldots, pk^n_{\text{san}}\}$. ADM describes the sanitizable or redactable blocks, i.e., ADM contains their indices.

$$(m, \sigma) \leftarrow \mathsf{Sign}_{\mathsf{SSS}}(m, sk_{\text{sig}}, \{pk^1_{\text{san}}, \ldots, pk^n_{\text{san}}\}, \text{ADM}), (m, \sigma) \leftarrow \mathsf{Sign}_{\mathsf{RSS}}(m, sk_{\text{sig}})$$

Sanitization (SSS) and Redaction (RSS). The algorithms modify m according to the instruction in MOD, i.e., $m' \leftarrow \text{MOD}(m)$. For RSSs, MOD contains the indices to be redacted, while for SSSs, MOD contains index/message pairs $\{i, m[i]'\}$ for those blocks i to be sanitized. They output a new signature σ' for m'. SSSs require a sanitizer's private key, while RSSs allow for public alterations.

$$(m', \sigma') \leftarrow \mathsf{Sanit}_{\mathsf{SSS}}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk^i_{\text{san}}), (m', \sigma') \leftarrow \mathsf{Redact}_{\mathsf{RSS}}(m, \text{MOD}, \sigma, pk_{\text{sig}})$$

Verification (SSS, RSS). The output bit $d \in \{\texttt{true}, \texttt{false}\}$ indicates the correctness of the signature with respect to the supplied public keys.

$$d \leftarrow \mathsf{Verify}_{\mathsf{SSS}}(m, \sigma, pk_{\text{sig}}, \{pk^1_{\text{san}}, \ldots, pk^n_{\text{san}}\}), \quad d \leftarrow \mathsf{Verify}_{\mathsf{RSS}}(m, \sigma, pk_{\text{sig}})$$

Proof (SSS). Uses the signer's secret key sk_{sig}, message/signature pairs and the sanitizers' public keys to output a string $\pi \in \{0,1\}^*$ for the $\mathsf{Judge}_{\mathsf{SSS}}$ algorithm.

$$\pi \leftarrow \mathsf{Proof}_{\mathsf{SSS}}(sk_{\text{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}, \{pk^1_{\text{san}}, \ldots, pk^n_{\text{san}}\})$$

Judge (SSS). Using proof π and public keys it decides $d \in \{\texttt{Sig}, \texttt{San}^i\}$ indicating who created the message/signature pair (`Signer` or $\texttt{Sanitizer}^i$).

$$d \leftarrow \mathsf{Judge}_{\mathsf{SSS}}(m, \sigma, pk_{\text{sig}}, \{pk^1_{\text{san}}, \ldots, pk^n_{\text{san}}\}, \pi)$$

2.1 Security Properties of RSSs and SSSs

We consider the following security properties as formalized in [5,8] :

Unforgeability (SSS, RSS) assures that third parties cannot produce a signature for a "fresh" message. "Fresh" means it has been issued neither by the signer, nor by the sanitizer. This is similar to the unforgeability requirements of standard signature schemes.

Immutability (SSS, RSS) immutability prevents the sanitizer from modifying non-admissible blocks. Most RSSs do treat all blocks as redactable, but if they differentiate, immutability exists equally, named "disclosure secure" [25].

Privacy (SSS, RSS) inhibits a third party from reversing alterations without knowing the original message/signature pair.

Accountability (SSS) allows to settle disputes over the signature's origin.

Trade secret protection is initially achieved by the above privacy property. Cryptographically stronger privacy notions have also been introduced:

Unlinkability (SSS, RSS) prohibits a third party from linking two messages.

All current notions of unlinkability require the use of group signatures [7]. Schemes for statistical notions of unlinkability only achieve the less common notion of selective unforgeability [1]. We do not consider unlinkability, if needed it can be achieved using a group signature instead of a normal signature [9].

Transparency (SSS, RSS) says that it should be impossible for third parties to decide which party is accountable for a given signature-message pair.

However, stronger privacy has to be balanced against legal requirements. In particular, transparent schemes do not fulfill the EU's legal requirements for digital signatures [22]. To tackle this, *Brzuska* et al. devised a non-transparent, yet private, SSS with non-interactive public accountability [8]. Their scheme does not impact on privacy and fulfills all legal requirements [8,22].

Non-interactive public accountability (SSS, RSS) offers a public judge, i.e., without additional information from the signer and/or sanitizer any third party can identify who created the message/signature pair (`Sig` or $\texttt{San}^i$).

3 Implementation on Smart Cards

First, the selected RSSs and SSSs must be secure following the state-of-the-art definition of security, i.e, immutable, unforgeable, private and either transparent *or* public-accountable. Transparent schemes can be used for applications with high privacy protection, e.g., patient records. Public accountability is required for a higher legal value [8]. Second, the schemes underlying cryptographic foundation must perform well on many off-the-shelf smart cards. Hence, we chose primitives based on RSA operations computing efficiently due to hardware acceleration.

The following schemes fulfill the selection criterions and have been implemented:

BFF+09: Transparent, private, single-sanitizer SSS by *Brzuska* et al. [5]: uses RSA signatures and RSA-based chameleon hash[1]

BFLS09: Public accountable, private, multi-sanitizer with delegation SSS by *Brzuska* et al. [6]: works with several RSA signatures

BPS12: Public accountable, private, multi-sanitizer SSSs by *Brzuska* et al. [8]: work with several RSA signatures

PSPdM12: Transparent, private RSS by *Pöhls* et al. [24]: uses RSA signature and accumulator based on modular exponentiations

Each participating party has its own smart card, protecting each entities' secret key. The algorithms that require knowledge of the private keys sk_{sig} or sk^i_{san} are performed on card. Hence, at least Sign and Sanit involve the smart card. When needed, the host obtains the public keys out of band, e.g., via a PKI.

3.1 SSS Scheme BFF+09 [5]

The scheme's core idea is to generate a digest for each admissible block using a tag-based chameleon hash [5]. Finally, all digests are signed with a standard signature scheme. At first, let $\mathsf{S} := (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVerify})$ be a regular UNF-CMA secure signature scheme. Moreover, let $\mathcal{CH} := (\mathsf{CHKeyGen}, \mathsf{CHash}, \mathsf{CHAdapt})$ be a tag-based chameleon hashing scheme secure under random-tagging attacks. Finally, let $\mathcal{PRF}$ be a pseudo random function and $\mathcal{PRG}$ a pseudo random generator. We modified the algorithms presented in [5] to eliminate the vulnerability identified by *Gong* et al. [14]. See [5] for the algorithms and the security model.

Key Generation: $\mathsf{KGen}_{\text{sig}}$ on input of 1^λ generates a key pair $(sk, pk) \leftarrow \mathsf{SKGen}(1^\lambda)$, chooses a secret $\kappa \leftarrow \{0,1\}^\lambda$ and returns $(sk_{\text{sig}}, pk_{\text{sig}}) \leftarrow ((sk, \kappa), pk)$. $\mathsf{KGen}_{\text{san}}$ generates a key pair $(sk^{\text{ch}}_{\text{san}}, pk^{\text{ch}}_{\text{san}}) \leftarrow \mathsf{CHKeyGen}(1^\lambda)$.

Signing: Sign on input of $m, sk_{\text{sig}}, pk^{\text{ch}}_{\text{san}}, \text{ADM}$ it generates $\text{NONCE} \leftarrow \{0,1\}^\lambda$, computes $x \leftarrow \mathcal{PRF}(\kappa, \text{NONCE})$, followed by $\text{TAG} \leftarrow \mathcal{PRG}(x)$, and chooses $r[i] \xleftarrow{\$} \{0,1\}^\lambda$ for each $i \in \text{ADM}$ at random. For each block $m[i] \in m$ let

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(pk^{\text{ch}}_{\text{san}}, \text{TAG}, (m, m[i]), r[i]) & \text{if } i \in \text{ADM} \\ m[i] & \text{otherwise} \end{cases}$$

[1] Modified to eliminate the vulnerability identified by *Gong* et al. [14].

and computes $\sigma_0 \leftarrow \mathsf{SSign}(sk_{\text{sig}}, (h, pk^{\text{ch}}_{\text{san}}, \text{ADM}))$, where $h = (h[0], \ldots, h[l])$. It returns $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, r[0], \ldots, r[k])$, where $k = |\text{ADM}|$.

Sanitizing: Sanit on input of a message m, information MOD, a signature $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[0], \ldots, r[k])$, pk_{sig} and $sk^{\text{ch}}_{\text{san}}$ checks that MOD is admissible and that σ_0 is a valid signature for $(h, pk_{\text{san}}, \text{ADM})$. On error, return $\perp$. It sets $m' \leftarrow \text{MOD}(m)$, chooses values $\text{NONCE}' \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\text{TAG}' \xleftarrow{\$} \{0,1\}^{2\lambda}$ and replaces each $r[j]$ in the signature by $r'[j] \leftarrow \mathsf{CHAdapt}(sk^{\text{ch}}_{\text{san}}, \text{TAG}, (m, m[j]), r[j], \text{TAG}', (m', m'[j]))$. It assembles $\sigma' = (\sigma_0, \text{TAG}', \text{NONCE}', \text{ADM}, r'[0], \ldots, r'[k])$, where $k = |\text{ADM}|$, and returns (m', σ').

Verification: Verify on input of a message m, a signature $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[0], \ldots, r[k])$, pk_{sig} and $pk^{\text{ch}}_{\text{san}}$ lets, for each block $m[i] \in m$,

$$h[i] \leftarrow \begin{cases} \mathsf{CHash}(pk^{\text{ch}}_{\text{san}}, \text{TAG}, (m, m[i]), r[i]) & \text{if } i \in \text{ADM} \\ m[i] & \text{otherwise} \end{cases}$$

and returns $\mathsf{SVerify}(pk_{\text{san}}, (h, pk^{\text{ch}}_{\text{san}}, \text{ADM}), \sigma_0)$, where $h = (h[0], \ldots, h[l])$.

Proof: Proof on input of $sk_{\text{sig}}, m, \sigma, pk^{\text{ch}}_{\text{san}}$ and a set of tuples $\{(m_i, \sigma_i)\}_{i \in \mathbb{N}}$ from all previously signer generated signatures it tries to lookup a tuple $(pk^{\text{ch}}_{\text{san}}, \text{TAG}, m[j], r[j])$ such that $\mathsf{CHash}(pk^{\text{ch}}_{\text{san}}, \text{TAG}, (m, m[j]), r[j]) = \mathsf{CHash}(pk^{\text{ch}}_{\text{san}}, \text{TAG}_i, (m_i, m_i[j]), r_i[j])$. Set $\text{TAG}_i \leftarrow \mathcal{PRG}(x_i)$, where $x_i \leftarrow \mathcal{PRF}(\kappa, \text{NONCE}_i)$. Return $\pi \leftarrow (\text{TAG}_i, m_i, m_i[j], j, pk_{\text{sig}}, pk^{\text{ch}}_{\text{san}}, r[j]_i, x_i)$. If at any step an error occurs, $\perp$ is returned.

Judge: Judge on input of m, a valid $\sigma, pk_{\text{sig}}, pk^{\text{ch}}_{\text{san}}$ and π obtained from Proof checks that $pk_{\text{sig}} = pk_{\text{sig}_\pi}$ and that π describes a non-trivial collision under $\mathsf{CHash}(pk_{\text{san}}, \cdot, \cdot, \cdot)$ for the tuple $(\text{TAG}, (j, m[j], pk_{\text{sig}}), r[j])$ in σ. It verifies that $\text{TAG}_\pi = \mathcal{PRG}(x_\pi)$ and on success outputs `San`, else `Sig`.

3.2 SSS Scheme BFF+09 [5] on Smart Card

In this scheme, the algorithms Sign, Proof and CHAdapt from Sanit require secret information. The smart card's involvement is illustrated in Fig. 1. First, the generation of the tag in the Sign algorithm uses the secret information κ. During $\mathsf{KGen}_{\text{sig}}$ we generate κ as a 1024 Bit random number using the smart card's pseudo random generator and store it on card. To obtain x, illustrated as invocation of $\mathcal{PRF}(\cdot, \cdot)$, the host passes a NONCE to the card, which together with κ forms the input for the $\mathcal{PRF}$ implementation on card. The card returns x to the host. On the host system, we let $\text{TAG} \leftarrow \mathcal{PRG}(x)$. Second, CHAdapt used in Sanit requires a modular exponentiation using d as exponent. d is part of the 2048 Bit private RSA key obtained by CHKeyGen. The host computes only the intermediate result $i = ((\mathcal{H}(\text{TAG}, m, m[i]) \cdot r^e) \cdot (\mathcal{H}(\text{TAG}', m', m'[i])^{-1}) \mod N$ from the hash calculation described in [5] and sends i to the smart card. The final modular exponentiation is performed by the smart card using the RSA decrypt

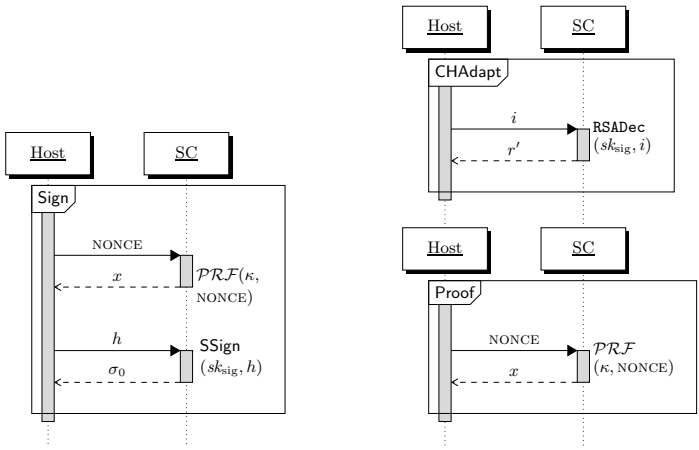


Fig. 1. BFF$^+$09: Data flow for algorithms Sign, CHAdapt and Proof

operation, provided by the Java Card API[2], to calculate $r' = i^d \mod N$ and returns r'. Finally, to execute the Proof algorithm on the `Signer`'s host requires the seed x as it serves as the proof that TAG has been generated by the signer. To obtain x, the host proceeds exactly as in the Sign algorithm, calling the $\mathcal{PRF}$ implementation on the card with the NONCE as parameter.

3.3 SSS Schemes BFLS09 [6] and BPS12 [8]

The core idea is to create and verify two signatures: first, fixed blocks and the `Sanitizer`'s pk_{san} must bear a valid signature under `Signer`'s pk_{sig}. Second, admissible blocks must carry a valid signature under either pk_{sig} or pk_{san}. The scheme by *Brzuska* et al. [8] is a modification of the scheme proposed by *Brzuska* et al. [6], that is shown to achieve message level public accountability [8] using an additional algorithm called Detect. Both, BFF$^+$09 and BPS12, solely build upon standard digital signatures. We implemented both; due to space restrictions and similarities, we only describe the BPS12 scheme, which achieves blockwise public accountability. Refer to [6] and [8] for the security model. In this section, the uniquely reversible concatenation of all non-admissible blocks within m is denoted FIX_m, that of all admissible blocks is denoted as ADM_m.

Key Generation: On input of 1^λ $\mathsf{KGen}_{\text{sig}}$ generates a key pair $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{SKGen}(1^\lambda)$. $\mathsf{KGen}_{\text{san}}$ generates a key pair $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{SKGen}(1^\lambda)$.

Signing: Sign on input of $m, pk_{\text{sig}}, sk_{\text{sig}}, pk_{\text{san}}$ and ADM_m, randomly chooses a TAG and computes $\sigma_{\text{FIX}} = \mathsf{SSign}(sk_{\text{sig}}, (0, \text{FIX}_m, \text{ADM}_m, pk_{\text{san}}, \text{TAG}))$. For

[2] RSA implementation must not apply any padding operations to its input. Otherwise, i is not intact anymore. We use Java Card's `ALG_RSA_NOPAD` to achieve this.

each $i \in \text{ADM}$. Compute $\sigma[i] \leftarrow \mathsf{SSign}(sk_{\text{sig}}, (1, i, m[i], pk_{\text{san}}, pk_{\text{sig}}, \text{TAG}, \perp))$ to form $\sigma_{\text{FULL}} \leftarrow (\sigma[0], \ldots, \sigma[l])$. Return $\sigma \leftarrow (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}_m, \text{TAG}, \perp)$.

Sanitizing: Sanit on input of message m, MOD, a signature σ generated by Sign, $pk_{\text{sig}}, sk_{\text{san}}$ and pk_{san} checks that MOD is admissible and that σ_{FIX} is valid under pk_{sig}. On error it returns $\perp$. Otherwise it generates the modified message $m' = \text{MOD}(m)$, draws a random TAG' and computes $\sigma'[i] \leftarrow \mathsf{SSign}(sk_{\text{san}}, (1, i, m'[i], pk_{\text{san}}, pk_{\text{sig}}, \text{TAG}, \text{TAG}'))$ for each block $i \in \text{MOD}$. For each $i \in \text{MOD}$ it replaces $\sigma[i] \in \sigma_{\text{FULL}}$ with $\sigma'[i]$ to obtain σ'_{FULL}. It returns (m', σ'), where $\sigma' \leftarrow (\sigma_{\text{FIX}}, \sigma'_{\text{FULL}}, \text{ADM}_m, \text{TAG}, \text{TAG}')$.

Verification: Verify on input of message $m, pk_{\text{sig}}, pk_{\text{san}}, \text{ADM}_m$ and a signature $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}_m, \text{TAG}, \text{TAG}')$ first verifies that σ_{FIX} is valid under pk_{sig}. If it is not valid it returns `false`, else it tries to verify that σ_{FULL} is valid under either pk_{sig} or pk_{san}. If σ_{FULL} is not valid under any of the public keys, `false` is returned and `true` otherwise.

Proof: Proof always returns $\perp$, as it is not required by Judge.

Judge: Judge on input of $(m, \sigma, pk_{\text{sig}}, pk_{\text{san}})$ first verifies that the signature σ is valid using Verify. If not, $\perp$ is returned. For each block $m[i] \in m$ it computes $d[i] \leftarrow \mathsf{Detect}(m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, i, \text{TAG}, \text{TAG}')$. If at any point $d[i] = \texttt{San}$, `San` is returned, `Sig` otherwise.

Detection: Detect on input of $(m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, i, \text{TAG}, \text{TAG}')$ returns `Sig` if $\mathsf{SVerify}(pk_{\text{sig}}, (1, m[i], pk_{\text{san}}, pk_{\text{sig}}, \text{TAG}, \text{TAG}')) = \texttt{true}$ and `San` if $\mathsf{SVerify}(pk_{\text{san}}, (1, m[i], pk_{\text{san}}, pk_{\text{sig}}, \text{TAG}, \text{TAG}')) = \texttt{true}$. If both SVerify evaluate to `false`, $\perp$ is returned.

3.4 SSS Schemes BFLS09 [6] and BPS12 [8] on Smart Card

We implemented Sign and Sanit with involvement of the smart card. Fig. 2 illustrates the interactions. The algorithms are executed on the host system as described in the scheme's description. For the Sign algorithm, cryptographic hash values over the values for σ_{FIX} and all the $\sigma[i]$ are signed with a RSA signature scheme using a 2048 Bit RSA key sk_{sig} and the signature functions provided by the card's API. The resulting signature values are returned to the host. The host assembles all $\sigma[i]$ to build the complete signature σ. In the Sanit algorithm the sanitizer's host first checks if $\text{MOD}(m)$ is admissible in ADM_m and, if admissible, modifies the message to obtain m'. For each block $m[i] \in \text{MOD}$, $\sigma'[i]$ is computed on the card, sending a cryptographic hash value over $m[i], pk_{\text{sig}}, \text{TAG}$ and TAG'. The sanitizer's host produces σ', combining all the $\sigma'[i]$ generated on card.

3.5 RSS Scheme PSPdM12 [24]

The scheme's core idea is to hash each block and accumulate all digests with a cryptographic accumulator. This accumulator value is signed with a standard signature scheme. Each time a block is accumulated, a witness that it is part of

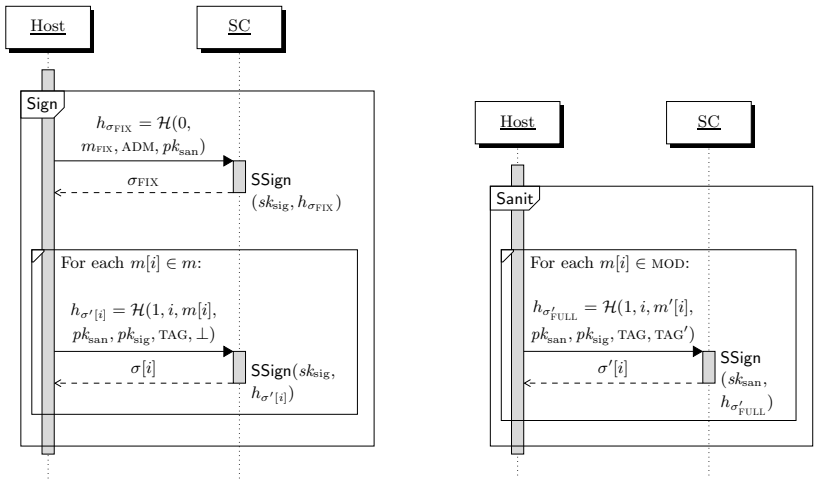


Fig. 2. BFLS09: Data flow between smart card and host for Sign and Sanit

the accumulated value is generated. Hence, the signed accumulator value is used to provide assurance that a block was signed given the verifier knows the block and the witness. A redaction removes the block and its witness. They further extended the RSS's algorithms with $\mathsf{Link}_{\mathsf{RSS}}$, $\mathsf{Merge}_{\mathsf{RSS}}$. We omit them, as they need no involvement of the smart card because they require no secrets. Refer to [24] for details on the security model.

Building Block: Accumulator. For more details than the algorithmic description, refer to [3,4,17,26]. We require the correctness properties to hold [3].

ACC consists of five PPT algorithms $\mathsf{ACC} := (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Dig}, \mathsf{Proof}, \mathsf{Verf})$:

Setup. Setup on input of the security parameter λ returns the parameters parm, i.e., $\mathsf{parm} \leftarrow \mathsf{Setup}(1^\lambda)$

Gen. Gen, on input of the security parameter λ and parm outputs pk i.e., $pk \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{parm})$.

Dig. Dig, on input of the set S, the public parameter pk outputs an accumulator value a and some auxiliary information aux, i.e, $(a, \mathsf{aux}) \leftarrow \mathsf{Dig}(pk, S)$

Proof. Proof, on input of the public parameter pk, a value $y \in \mathcal{Y}_{pk}$ and aux returns a witness p from a witness space $\mathcal{P}_{pk}$, and $\perp$ otherwise, i.e., $p \leftarrow \mathsf{Proof}(pk, \mathsf{aux}, y, S)$

Verf. On input of the public parameters parm, public key pk, an accumulator $a \in \mathcal{X}_{pk}$, a witness p, and a value $y \in \mathcal{Y}_{pk}$ Verf outputs a bit $d \in \{0, 1\}$ indicating whether p is a valid proof that y has been accumulated into a, i.e., $d \leftarrow \mathsf{Verf}(pk, a, y, p)$. Note, $\mathcal{X}_{pk}$ denotes the output and $\mathcal{Y}_{pk}$ the input domain based on pk; and parm is always correctly recoverable from pk.

Our Trade-Off between Trust and Performance. *Pöhls* et al. [24] require ACC to be collision-resistant without trusted setup. Foremost, they require the ACC's setup to hide certain values used for the parameter generation from untrusted parties, as knowledge allows efficient computation of collisions and thus forgeries of signatures. All known collision-resistant accumulators based on number theoretic assumptions either require a trusted third party (TTP), named the accumulator manager [4,16], or they are very inefficient. As said, the TTP used for setup of the ACC must be trusted not to generate collisions to forge signatures. However, existing schemes without TTP are not efficiently implementable, e.g., the scheme introduced by *Sander* requires a modulus size of $\gg 40,000$ Bit [26].

Our trade-off still requires a TTP for the setup, but inhibits the TTP from forging signatures generated by signers. In brief, we assume that the TTP which signs a participant's public key also runs the ACC setup. The TTP already has as a secret the standard RSA modulus $n = pq$, $p, q \in \mathbb{P}$. If we re-use n as the RSA-accumulator's modulus [4], the TTP could add new elements without detection. However, if we add "blinding primes" during signing, neither the TTP nor the signer can find collisions, *as long as the TTP and the signer do not collude.* We call this semi-trusted setup. Note, as we avoid algorithms for jointly computing a modulus of unknown factorization, we do not require any protocol runs. Thus, keys can be generated off-line. The security proof is in the appendix.

On this basis we build a practically usable *undeniable* RSS, as introduced in [24]. It is based on a standard signature scheme $\mathsf{S} := (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVerify})$ and our accumulator with semi-trusted setup $\mathsf{ACC} := (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Dig}, \mathsf{Proof}, \mathsf{Verf})$.

Key Generation: The algorithm KeyGen generates $(sk_{\mathsf{S}}, pk_{\mathsf{S}}) \leftarrow \mathsf{SKGen}(1^{\lambda})$. It lets $parm \leftarrow \mathsf{Setup}(1^{\lambda})$ and $pk_{\mathsf{ACC}} \leftarrow \mathsf{Gen}(1^{\lambda}, \mathsf{parm})$. The algorithm returns $((pk_{\mathsf{S}}, \mathsf{parm}, pk_{\mathsf{ACC}}), (sk_{\mathsf{S}}))$.

Signing: Sign on input of sk_{S}, pk_{ACC} and a set S, it computes $(a, \mathsf{aux}) \leftarrow \mathsf{Dig}(pk_{\mathsf{ACC}}, (S))$. It generates $P = \{(y_i, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk_{\mathsf{ACC}}, \mathsf{aux}, y_i, S) \mid y_i \in S\}$, and the signature $\sigma_a \leftarrow \mathsf{SSign}(sk_{\mathsf{S}}, a)$. The tuple (S, σ_s) is returned, where $\sigma_s = (pk_{\mathsf{S}}, \sigma_a, \{(y_i, p_i) \mid y_i \in S\})$.

Verification: Verify on input of a signature $\sigma = (pk_{\mathsf{S}}, \sigma_a, \{(y_i, p_i) \mid y_i \in S\})$, parm and a set S first verifies that σ_a verifies under pk_{S} using SVerify. For each element $y_i \in S$ it tries to verify that $\mathsf{Verf}(pk_{\mathsf{ACC}}, a, y_i, p_i) = \texttt{true}$. In case Verf returns `false` at least once, Verify returns `false` and `true` otherwise.

Redaction: Redact on input of a set S, a subset $R \subseteq S$, an accumulated value a, pk_{S} and a signature σ_s generated with Sign first checks that σ_s is valid using Verify. If not $\perp$ is returned. Else it returns a tuple (S', σ'_s), where $\sigma'_s = (pk_{\mathsf{S}}, \sigma_a, \{(y_i, p_i) \mid y_i \in S'\})$ and $S' = S \setminus R$.

3.6 RSS Scheme PSPdM12 [24] on Smart Card

This scheme involves the smart card for the algorithms Setup and Sign, illustrated in Fig. 3. We use the smart card to obtain the blinding primes of the modulus described in Sect. 3.5, needed by Setup. To compute these primes on card, we

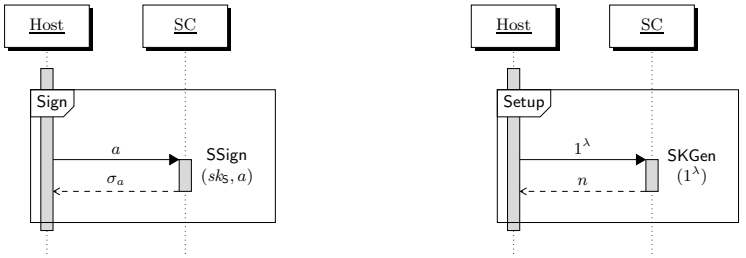


Fig. 3. PSPdM12: Data flow between smart card and host for Sign and Setup

generate standard RSA parameters (N, e, d) with N being of 2048 Bit length, but store only N on card and discard the exponents. On the host system this modulus is multiplied with that obtained from the TTP to form the modulus used by ACC. Additionally, the smart card performs SSign to generate σ_a.

4 Performance and Lessons Learned

We implemented in *Java Card* [11] 2.2.1 on the "SmartC@fé® Expert 4.x" from *Giesecke and Devrient* [13]. The host system was an *Intel* i3-2350 Dual Core 2.30 GHz with 4 GiB of RAM. For the measurements in Tab. 1, we used messages with 10, 25 and 50 blocks of equal length, fixed to 1 Byte. The block size has little impact as inputs are hashed. However, the number of blocks impacts performance in some schemes. $\lfloor \frac{1}{2}\ell \rfloor$ blocks were marked as sanitizable. The Sanit and Redact operations modify all sanitizable blocks. The BFLS12 scheme allows multiple sanitizers and was measured with 10 sanitizers. Verify and Judge always get sanitized or redacted messages. The results for the BFLS12 scheme include the verification against all possible public keys (worst-case). We measured the complete execution of the algorithms, including those steps performed on the host system. We omit the time KeyGen takes for 2048 bit long key pairs, as keys are usually generated in advance.

We carefully limited the involvement of the smart card, hence we expect the performance impact to be comparable to the use of cards in regular signature schemes. For the RSS we have devised and proven a new collision-resistant accumulator. If one wants to compare, BPS12 states around 0.506s for signing 10

Table 1. Performance of SSS prototypes; median runtime in seconds

	Sign			Sanit/ Redact			Verify			Judge			Detect/ Proof		
ℓ	10	25	50	10	25	50	10	25	50	10	25	50	10	25	50
[5]	1.22[5]	1.25[5]	1.25[5]	4.25[5]	9.40[5]	17.96[5]	1.09	1.11	1.12	1.78	1.77	1.76	1.53[5]	1.54[5]	1.57[5]
[6]	1.09[5]	1.09[5]	1.08[5]	0.58[5]	0.57[5]	0.57[5]	0.017	0.017	0.017	0.017	0.017	0.017	-[4]	-[4]	-[4]
[8]	3.12[5]	7.16[5]	13.24[5]	2.60[5]	6.65[5]	12.74[5]	0.016	0.039	0.084	0.043	0.051	0.060	0.001	0.001	0.002
[24]	11.16[5]	59.97[5]	221.97[5]	1.42	3.17	6.32	1.32	3.12	6.12	-[4]	-[4]	-[4]	-[4]	-[4]	-[4]

[4]Algorithm not defined by scheme [5]Involves smart card operations

blocks with 4096 bit keys [8]. We only make use of the functions exposed by the API. Hence, our implementations are portable to other smart cards, given they provide a cryptographic co-processor that supports RSA algorithms. We would have liked direct access to the cryptographic co-processor, as raised in [29], instead of using the exposed `ALG_RSA_NOPAD` as a workaround.

References

1. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 1–20. Springer, Heidelberg (2012)
2. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable Signatures. In: De Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005)
3. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
4. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
5. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of Sanitizable Signatures Revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009)
6. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Sanitizable signatures: How to partially delegate control for authenticated data. In: Proc. of BIOSIG. LNI, vol. 155, pp. 117–128. GI (2009)
7. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 444–461. Springer, Heidelberg (2010)
8. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-interactive public accountability for sanitizable signatures. In: Proc. of EuroPKI 2012. LNCS. Springer (2012)
9. Canard, S., Girault, M.: Implementing group signature schemes with smart cards. In: Proc. of CARDIS (2002)
10. Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 35–52. Springer, Heidelberg (2012)
11. Chen, Z.: Java Card Technology for Smart Cards: Architecture and Programmer's Guide. Addison-Wesley (2000)
12. EC: Directive 1999/93/EC from 13 December 1999 on a Community framework for electronic signatures. Official Journal of the EC L 12, 12–20 (2000)
13. Giesecke & Devrient GmbH. SmartC@fé® Expert 4.0 V.05.2008 (2008)
14. Gong, J., Qian, H., Zhou, Y.: Fully-secure and practical sanitizable signatures. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 300–317. Springer, Heidelberg (2011)
15. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)

16. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer, Heidelberg (2007)
17. Lipmaa, H.: Secure accumulators from euclidean rings without trusted setup. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 224–240. Springer, Heidelberg (2012)
18. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: Proc. of ACM CCS, CCS 1996, pp. 48–57. ACM (1996)
19. Meister, G., Vogel, M.: Protection profiles and generic security targets for smart cards as secure signature creation devices - existing solutions for the payment sector. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 179–187. Springer, Heidelberg (2001)
20. Miyazaki, K., Susaki, S., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H.: Digital documents sanitizing problem. Technical Report ISEC2003-20, IEICE (2003)
21. Okamoto, T., Tada, M., Okamoto, E.: Extended proxy signatures for smart cards. In: Zheng, Y., Mambo, M. (eds.) ISW 1999. LNCS, vol. 1729, pp. 247–258. Springer, Heidelberg (1999)
22. Pöhls, H.C., Höhne, F.: The role of data integrity in EU digital signature legislation — achieving statutory trust for sanitizable signature schemes. In: Meadows, C., Fernandez-Gago, C. (eds.) STM 2011. LNCS, vol. 7170, pp. 175–192. Springer, Heidelberg (2012)
23. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable Signatures in XML Signature — Performance, Mixing Properties, and Revisiting the Property of Transparency. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 166–182. Springer, Heidelberg (2011)
24. Pöhls, H.C., Samelin, K., Posegga, J., de Meer, H.: Transparent mergeable redactable signatures with signer commitment and applications. Technical Report MIP-1206, University of Passau (August 2012)
25. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable signatures for independent removal of structure and content. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 17–33. Springer, Heidelberg (2012)
26. Sander, T.: Efficient accumulators without trapdoor extended abstract. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 252–262. Springer, Heidelberg (1999)
27. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)
28. Tan, K.W., Deng, R.H.: Applying sanitizable signature to web-service-enabled business processes: Going beyond integrity protection. In: Proc. of ICWS 2009, pp. 67–74 (2009)
29. Tews, H., Jacobs, B.: Performance issues of selective disclosure and blinded issuing protocols on java card. In: Markowitch, O., Bilas, A., Hoepman, J.-H., Mitchell, C.J., Quisquater, J.-J. (eds.) WISTP 2009. LNCS, vol. 5746, pp. 95–111. Springer, Heidelberg (2009)

Experiment $\mathsf{Semi-Trusted-Collision-ResistancePK}^{\mathsf{ACC}}_{\mathcal{A}}(\lambda)$
$\mathsf{parm} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
$(pk^*, p^*, m^*, a^*) \leftarrow \mathcal{A}^{\mathsf{ODig}(\cdot,\cdot,\cdot)}(1^\lambda, \mathsf{parm})$
where oracle ODig, on input of S_i, pk_i returns:
$(a_i, \mathsf{aux}_i) \leftarrow \mathsf{Dig}(pk_i, S_i)$ (answers/queries indexed by i, $1 \le i \le k$)
$P_i = \{(s_j, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk_i, \mathsf{aux}_i, s_j, S_i), s_j \in S_i\}$
return (a_i, P_i)
return 1, if:
$\mathsf{Verf}(pk^*, a^*, m^*, p^*) = 1$ and
$\exists i, 1 \le i \le k : a_i = a^*$ and $m^* \notin S_i$

Fig. 4. Collision-Resistance with Semi-Trusted Setup Part I

Experiment $\mathsf{Semi-Trusted-Collision-ResistancePARM}^{\mathsf{ACC}}_{\mathcal{A}}(\lambda)$
$(\mathsf{parm}^*, s^*) \leftarrow \mathcal{A}(1^\lambda)$
$(pk^*, p^*, m^*, a^*) \leftarrow \mathcal{A}^{\mathsf{ODig}(\cdot,\cdot,\cdot),\mathsf{GetPk}()}(1^\lambda, s^*)$
where oracle ODig, on input of pk_i, S_i:
$(a_i, \mathsf{aux}_i) \leftarrow \mathsf{Dig}(pk_i, S_i)$ (answers/queries indexed by i, $1 \le i \le k$)
$P_i = \{(s_j, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk_i, \mathsf{aux}_i, s_j, S_i), s_j \in S_i\}$
return (a_i, P_i)
where oracle GetPk returns:
$pk_j \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{parm}^*)$ (answers/queries indexed by j, $1 \le j \le k'$)
return 1, if:
$\mathsf{Verf}(pk^*, a^*, m^*, p^*) = 1$ and
$\exists i, 1 \le i \le k : a_i = a^*$, $m^* \notin S_i$ and $\exists j, 1 \le j \le k' : pk^* = pk_j$

Fig. 5. Collision-Resistance with Semi-Trusted Setup Part II

A Collision-Resistant Acc. with Semi-Trusted Setup

Definition 1 (Collision-Resistance with Semi-Trusted Setup (Part I)). *We say that an accumulator ACC with semi-trusted setup is collision-resistant for the public key generator, iff for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 4 returns* 1*, is negligible (as a function of λ).*

The basic idea is to let the adversary generate public key *pk*. The other part is generated by the challenger. Afterwards, the adversary has to find a collision.

Definition 2 (Collision-Resistance with Semi-Trusted Setup (Part II)). *We say that an accumulator ACC with semi-trusted setup is collision-resistant for the parameter generator, iff for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. A returns* 1*, is negligible (as a function of λ).*

The basic idea is to either let the adversary generate the public parameters parm, but not any public keys; they are required to be generated honestly. Afterwards, the adversary has to find a collision.

Setup. The algorithm Setup generates two safe primes p_1 and q_1 with bit length λ. It returns $n_1 = p_1 q_1$.

Gen. On input of the parameters parm, containing a modulus $n_1 = p_1 q_1$ of unknown factorization and a security parameter λ, the algorithm outputs a

multi-prime RSA-modulus $N = n_1 n_2$, where $n_2 = p_2 q_2$, where $p_2, q_2 \in \mathbb{P}$ are random safe primes with bit length λ.

Verf. On input of the parameters $\mathsf{parm} = n_1$, containing a modulus $N = p_1 q_1 p_2 q_2 = n_1 n_2$ of unknown factorization, a security parameter λ, an element y_i, an accumulator a, and a corresponding proof p_i, it checks, whether $p_i^{y_i} \pmod N = a$ and if $n_1 \mid N$ and $n_2 = \frac{N}{n_1} \notin \mathbb{P}$. If either checks fails, it returns 0, and 1 otherwise

Other Algorithms: The other algorithms work exactly like the standard collision-free RSA-accumulator, i.e., [3].

Theorem 1 (The Accumulator is Collisions-Resistant with Semi -Trusted Setup.). *If either the parameters* parm *or the public key* pk *has been generated honestly, the sketched construction is collision-resistant with semi-trusted setup.*

Proof. Based on the proofs given in [3], we have to show that an adversary able to find collisions is able to find the e^{th} root of a modulus of unknown factorization. Following the definition given in Fig. 4 and Fig. A, we have three cases:

I) **Malicious Semi-Trusted Third Party.** As parm is public knowledge, every party can compute $n_2 = \frac{N}{n_1}$. For this proof, we assume that the strong RSA-assumption [3] holds in $(\mathbb{Z}/n_1\mathbb{Z})$ and $(\mathbb{Z}/n_2\mathbb{Z})$. Moreover, we require that $\gcd(n_1, n_2) = 1$ holds. As $(\mathbb{Z}/N\mathbb{Z}) \cong (\mathbb{Z}/n_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z})$ we have a group isomorphism φ_1. Furthermore, as the third party knows the factorization of n_1, we have another group isomorphism φ_2. It follows: $(\mathbb{Z}/N\mathbb{Z}) \cong (\mathbb{Z}/p_1\mathbb{Z}) \times (\mathbb{Z}/q_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z})$. Assuming that $\mathcal{A}$ can calculate the e^{th} root in $(\mathbb{Z}/N\mathbb{Z})$, it implies that it can calculate the e^{th} root in $(\mathbb{Z}/n_2\mathbb{Z})$, as calculating the e^{th} root in $(\mathbb{Z}/p\mathbb{Z})$, with $p \in \mathbb{P}$ is trivial. It follows that $\mathcal{A}$ breaks the strong RSA-assumption in $(\mathbb{Z}/n_2\mathbb{Z})$. Building a simulation and an extractor is straight forward.

II) **Malicious Signer.** Similar to I).

III) **Outsider.** Outsiders have less knowledge, hence a combination of I) and II).

Obviously, if the factorization of n_1 **and** n_2 is known, one can simply compute the e-th root in $(\mathbb{Z}/N\mathbb{Z})$. However, we assumed that signer and TTP do not collude. All other parties can collude, as the factorization of n_2 remains secret with overwhelming probability.

Cryptographic Key Exchange in IPv6-Based Low Power, Lossy Networks

Panagiotis Ilia, George Oikonomou, and Theo Tryfonas

Cryptography Group, Faculty of Engineering, University of Bristol,
Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK
panagiotis.ilia.2011@my.bristol.ac.uk,
{g.oikonomou,theo.tryfonas}@bristol.ac.uk

Abstract. The IEEE 802.15.4 standard for low-power radio communications defines techniques for the encryption of layer 2 network frames but does not discuss methods for the establishment of encryption keys. The constrained nature of wireless sensor devices poses many challenges to the process of key establishment. In this paper, we investigate whether any of the existing key exchange techniques developed for traditional, application-centric wireless sensor networks (WSN) are applicable and viable for IPv6 over Low power Wireless Personal Area Networks (6LoWPANs). We use Elliptic Curve Cryptography (ECC) to implement and apply the Elliptic Curve Diffie Hellman (ECDH) key exchange algorithm and we build a mechanism for generating, storing and managing secret keys. The mechanism has been implemented for the Contiki open source embedded operating system. We use the Cooja simulator to investigate a simple network consisting of two sensor nodes in order to identify the characteristics of the ECDH technique. We also simulate a larger network to examine the solution's performance and scalability. Based on those results, we draw our conclusions, highlight open issues and suggest further work.

Keywords: 6LoWPAN, Key Exchange, ECC, ECDH.

1 Introduction

Wireless Sensor Networks (WSNs) consist of a large number of autonomous devices that cooperate to collect important data and send them through wireless communication channels to a base station or a data centre. Every node mainly consists of a microcontroller, a memory unit, a transceiver, a power source and one or more sensing elements. Due to their nature, wireless sensors are very constrained in terms of available RAM, speed of computation, network bandwidth and battery lifetime.

In 2003, the IEEE published the first version of IEEE 802.15.4, a specification for the physical and link layer operation for low-power radio communication. Initial research efforts suggested that TCP/IP was not viable for WSNs and that bespoke, application-centric network stacks were more suitable [1, 2]. However,

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 34–49, 2013.

the release of uIP demonstrated that standards-compliant TCP/IP stacks for embedded devices are viable [3]. Subsequently, it was shown that a TCP/IP-based WSN could outperform traditional, application-centric network designs [4]. As a result, a series of Internet specifications have been suggested for the transmission and routing of datagrams with IPv6 over Low power Wireless Personal Area Networks (6LoWPANs) [5].

With 6LoWPAN, WSN nodes with IEEE 802.15.4 radio transceivers are directly accessible from the Internet and are exposed to a host of security threats. Modern sensor devices are often equipped with an encryption/decryption coprocessor and link layer frames can be transmitted encrypted with the 128-bit Advanced Encryption Standard (AES) algorithm. However, there are challenges associated with key management and the process of key exchange in 6LoWPANs is not trivial. On many occasions, keys are individually pre-loaded to the nodes, which can be characterized as an important security vulnerability.

In this paper, we investigate whether any of the existing techniques for dynamic generation and exchange of cryptographic keys are applicable and can be adopted for 6LoWPANs. We argue that techniques based on Elliptic Curve Cryptography (ECC) [6] are very promising and we implement an Elliptic Curve Diffie Hellman (ECDH) shared key generation and establishment algorithm. This mechanism is also responsible for the management of existing encryption keys, searching and returning them to the link layer when requested, or starting the key exchange process if two neighbouring nodes do not have a shared secret key. It additionally handles key storage, expiration and replacement.

We implemented the key management mechanism and ECDH algorithm for the Contiki embedded operating system[1] and we evaluated it in terms of applicability, viability and scalability with network size and density. We evaluate memory footprint for a single node as well as how the number of stored keys affects network scalability. In addition, we use the Cooja simulator to conduct several simulations and assess node energy consumption, network lifetime and performance.

2 Background

Key management schemes proposed for WSNs are divided mainly into two different categories: i) symmetric key schemes where the keys are either pre-installed or assigned by a trusted party and ii) schemes based on Public Key Cryptography (PKC).

2.1 Symmetric Key Schemes

Compared to PKC, symmetric key schemes have the advantage that they are less computationally intensive, requiring fewer micro-controller instruction cycles to perform the calculations required for encryption and decryption.

[1] `http://www.contiki-os.org`

Keys are pre-installed during the network's deployment and initialization phase and neighbouring nodes discover and establish a shared key during the network formation phase [7].

Among existing efforts is a key exchange approach which requires the existence of a trusted party in the WSN, acting as a Key Distribution Centre (KDC), a role which can be assumed by a Base Station (BS) [8]. In this scheme the KDC must establish a secure, single-hop communication channel with every node of the network, in order to deliver the secret keys. The trusted entity-based key exchange scheme cannot be applied in 6LoWPANs because the assumption that all nodes are within a single hop of the base station does not always hold true: The aim of 6LoWPAN and related specifications is to facilitate the formation of multi-hop networks [9].

A different approach relies on each node having a pre-installed set of keys chosen from a key pool. These schemes can be either deterministic or probabilistic. Under deterministic schemes, every node is capable of establishing a pair-wise key with all its neighbors. One method that stands out is the one proposed in [10], whereby every two nodes in the network share exactly one common key. According to Bechit et al., deterministic schemes do not scale well with network size [11] and are thus unsuitable for 6LoWPANs where scalability is a desirable feature.

Under probabilistic schemes, a common key is present between two neighbors with some probability. For instance, under the scheme documented in [12], a small subset of k keys is chosen randomly out of a large key pool S. Every network node exchanges the identifiers of its keys with its neighbors and, if a common key exists, it is used as their session secret key. A more contemporary probabilistic scheme is the one proposed in [11]. Because of the probabilistic nature of such schemes, many pairs of nodes do not share a common secret key and thus, they try to find a secure routing path through their neighbors in order establish it. If the connectivity of the network graph is not high, it is possible that the network becomes partitioned into sub-networks and thus it may be impossible to discover paths for key establishment between the two parts. Since full network connectivity is not guaranteed, such schemes are unsuitable for 6LoWPANs [8]. Additionally, if a single node is compromised a large subset of the global key pool may be revealed to the attacker [11]. However, because keys are pre-installed, revocation and replacement is not trivial.

2.2 Public Key Cryptography

Public key schemes are more demanding in terms of computation and energy consumption than symmetric key schemes. However, Zhang and Varadharajan argue that public key schemes provide a higher level of security, they scale better with network size and they have lower storage requirements [7].

The classic Diffie Hellman algorithm uses keys of very large size and does not provide any authentication mechanism, unless used alongside other protocols. It is reported that the RSA encryption algorithm is viable in WSNs despite using a large key [13–15]. The advantage of RSA over Diffie Hellman is that it can

provide both key exchange and authentication with a single pair of keys (public-private). However, the processes of key generation and encryption (for secret key establishment) are still very slow and energy consuming.

Elliptic Curve Cryptography (ECC) is a very attractive solution for 6LoWPANs, since key length is considerably smaller than that used by other traditional PKC schemes. Consequently, according to NIST recommendations, the strength of a 160-bit ECC key is equivalent to a 1024-bit RSA key [16].

Reportedly RSA and ECC cryptosystems with 1024-bits and 160-bits key size respectively have been implemented on MICA motes and PKC is a feasible security solution for sensor networks [13]. Additionally, software implementations of RSA and ECC public-key algorithms exist for Atmel AVR Atmega128 microcontrollers [14], which is a hardware platform commonly encountered in 6LoWPAN deployments. It is observed that the 160-bits ECC is not only much faster than the equivalent 1024-bits RSA, but it also uses less memory for data and code hosting. The authors of that work extend their research by implementing RSA key exchange with mutual authentication as well as ECDH with ECDSA, between two non-trusted parties [15]. They discuss a simplified and lightweight Secure Sockets Layer (SSL) protocol in order to allow sensor nodes to perform a handshake and subsequently to negotiate and establish a secret key. Energy consumption for a full handshake with ECC is four times lower than with RSA.

Bianchi et al. introduce the asymmetric scheme of Identity Based Cryptography (IBC), which is based on bilinear pairings on elliptic curves, as a promising key exchange scheme for WSNs [17]. The IPC scheme was subsequently adopted for IP-based WSNs [18]. The fundamental idea of IBC is that every string, like the identity of each node (ID), can be used as a valid public key and thus the use of large certificates for authentication is avoided. By using this approach, nodes are able to establish a common secret key without any communication. However, the main drawback stems from the fact that private keys are computed only by the trusted authority (TA) by using the ID of the each node and its secret key. If a particular key is leaked, the TA must pick a new secret key and start a re-keying phase.

3 Implementation of the Key Exchange Technique

We implemented the ECDH key exchange technique for the Contiki OS, which is a portable and lightweight operating system, specifically designed for use by devices with limited resources. The Contiki OS supports a full TCP/IP network stack, including support for a host of standard internet protocols, such as IPv6, UDP, TCP, ICMP and HTTP. It also implements the 6LoWPAN adaptation layer as defined in IETF's Request For Comments (RFC) 4944 [5] and the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [9].

3.1 Link Layer Frame and Framer

Our proposed key exchange solution is implemented as a daemon process and underpins the network stack's secure link layer frame transmission by establishing

secret keys for symmetric encryption between single-hop neighbor nodes. Contiki's implementation of IEEE 802.15.4 Medium Access Control (MAC) frame generation and parsing does not currently support the security header specified by the IEEE 802.15.4 standard. We have modified the respective code module to support the generation and parsing of the security header in a standards-compliant fashion. If the application determines that the MAC frame needs to be secure (by defining the security bit in the frame control field), the security header is appended to the address field of the MAC header, prior to the data payload.

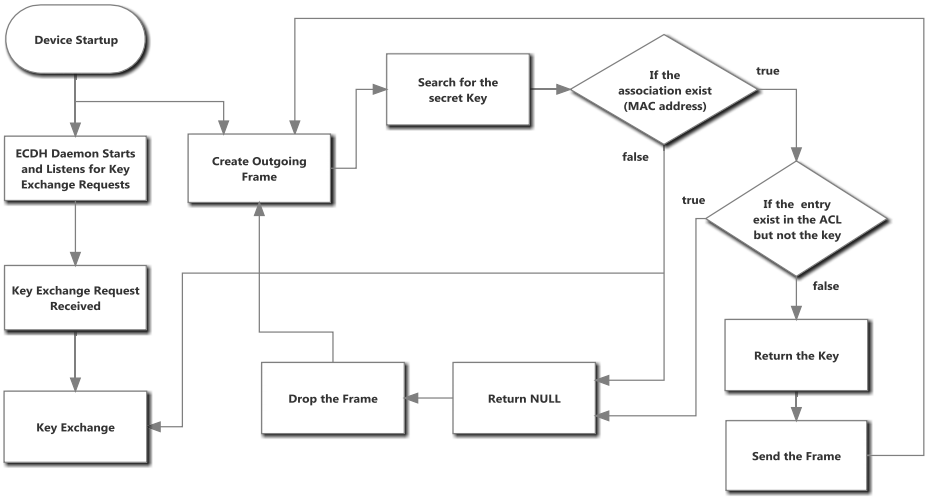


Fig. 1. Flowchart for Secret Key Search, Frame Creation and Transmission

The link layer frame generation module accepts outgoing frames from the layer 2 driver, adds the MAC header at the beginning of each frame and delivers them to the radio transceiver's driver. It also receives incoming frames, parses and removes the MAC header and delivers the payload to the upper layers. Apart from the frame structure, we also modified the link layer framer to fill the values of the Auxiliary Security Header.

The link layer on the transmitting node queries our driver for the existence of a shared key with the intended recipient, identified by the frame's destination MAC address. We search our Access Control List (ACL) for an entry matching the destination MAC address and, if an entry exists, we return the established key as shown in Fig. 1. If the association is not in the ACL the frame is dropped due to the lack of security guarantees and an internal transparent process for key exchange starts.

Fig. 2 illustrates a state transition diagram for our key exchange daemon process. A device will spend most of its time in the *Listening* state. When the network stack attempts to transmit a frame, the node will transition to the

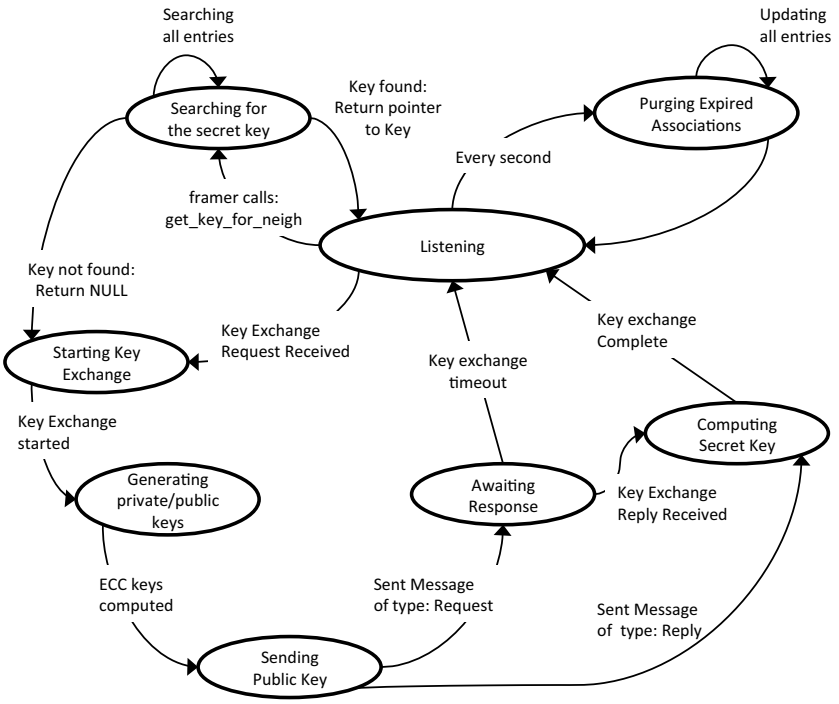


Fig. 2. State Transition Diagram for the Process of Key Exchange

Searching for the secret key state. Depending on the outcome of the key search, the daemon will either return a pointer to the shared key and revert to the *Listening* state or it will move to the *Starting Key Exchange* state.

3.2 ECC Implementation

In the current work, we used source code from the ContikiECC project [19] in order to implement the basic elliptic curve operations. The ContikiECC project is a Contiki port of the TinyECC library [20]. It implements functions to handle very large numbers as multiple 8-bit or 16-bit words and provides the basic numerical operations for 8-bit and 16-bit microprocessors. Moreover, it provides a number of elliptic curves of sizes 128, 160 and 192 bits by specifying each curve's parameters and base point. As discussed above, modern sensor hardware platforms provide hardware acceleration for 128-bit AES encryption, hence our decision to use a curve of 128 bits for the construction of secret keys.

From the ContikiECC library, we use the basic large number operations to implement the ECC operations of point addition, point doubling and multiplication by a scalar multiplier. ECC operations are based on the sliding window method which provides more optimised characteristics in comparison to other methods.

In order to implement the Diffie Hellman algorithm over elliptic curves, each node creates an ephemeral private key as a random 128-bit number. By multiplying the private key with the elliptic curve's base point, we compute the node's public key. For the establishment of a shared secret key between two parties, each party multiplies its own private key with the public key received by the node it is negotiating with. Public keys are transmitted as clear text. A new private-public key pair is used for each negotiation.

For the generation of random numbers, we use Contiki's library which provides a platform-independent Application Programming Interface (API). Each platform supported by Contiki provides a hardware-specific Random Number Generator (RNG) implementation, which underpins this API. For instance, the cc2430 and cc2530 System-on-Chip (SoC) platforms provide hardware-based RNG implementations, while other platforms rely on software. In most cases, random bits from the Radio Transceiver's receive path are used to seed the RNG implementation.

3.3 Key Storage and Management

We construct a custom data structure (key association) that links the destination node's MAC address, the established secret key, key lifetime and the state of the key exchange procedure. The information remains stored in the structure until the shared key expires. After the shared secret has been established, the private-public key pair used to generate it is erased.

Every node in the network has a statically pre-allocated ACL table for storing key association data structures. Each key association has a lifetime (in seconds), which is set when the key establishment process is over. Every second, the daemon periodically enters the *Purging Expired Associations* state (Fig. 2) and decrements key lifetimes by one. When an entry's lifetime reaches zero the shared secret is erased, the MAC address is set to all zeros and the entry's state is reset. This releases the association, which can then be allocated for a new key establishment in the future.

At the beginning of the key exchange process the sensor node allocates a free ACL entry to store the new association in relation to the destination's MAC address. A special situation is the case where an entry is not completed but is already allocated, which means that another key exchange process is in progress (with different neighbor). By handling this situation we avoid the re-allocation of an already allocated association and we can support multiple concurrent key establishment negotiations.

3.4 The Key Exchange Process

The Diffie Hellman key exchange daemon is a background process, which remains idle as long as a key exchange is not requested. The process is transparent and application-independent.

The ECDH daemon is triggered if a secret key is requested by the network stack but does not exist in the ACL table. In order to start the key exchange

process, the daemon first allocates a free ACL entry. It then queries the node's Neighbor Discovery (ND) cache to determine the IPv6 address of the destination node. If both steps are successful, the daemon computes the ECC private-public key pair and sends a key exchange request to the destination node over UDP.

We build a simple protocol for the key exchange messages, defining the Protocol Version, Message Type and Payload, which actually is the sender's public key. Reception of a key exchange request also triggers the ECDH daemon. Upon reception of an ECDH message, the receiving node first validates the protocol version and message type, as illustrated in Fig. 3.

If the message type is Request, it means that the sender node is asking for key exchange and sends its public key. Thus, the receiver allocates an ACL entry, creates its own ephemeral ECC keys, sends a Reply message, computes the secret shared key by performing elliptic curve point multiplication and sets the key lifetime. In this case, the daemon enters the states of *Generating Private/Public Keys*, *Sending Public Key*, *Awaiting Response* and *Computing Secret Key* in that order, as shown in the state transition diagram in Fig. 2. On the other hand, if the received message is a key exchange reply, the node searches to find the related ACL association and computes the secret shared key by using the data of the specific association entry and the received public key.

To overcome the situation of an ACL association being permanently allocated because of indefinitely waiting for a key exchange reply message, we set a key-exchange timeout by setting the association's lifetime to a low value during the

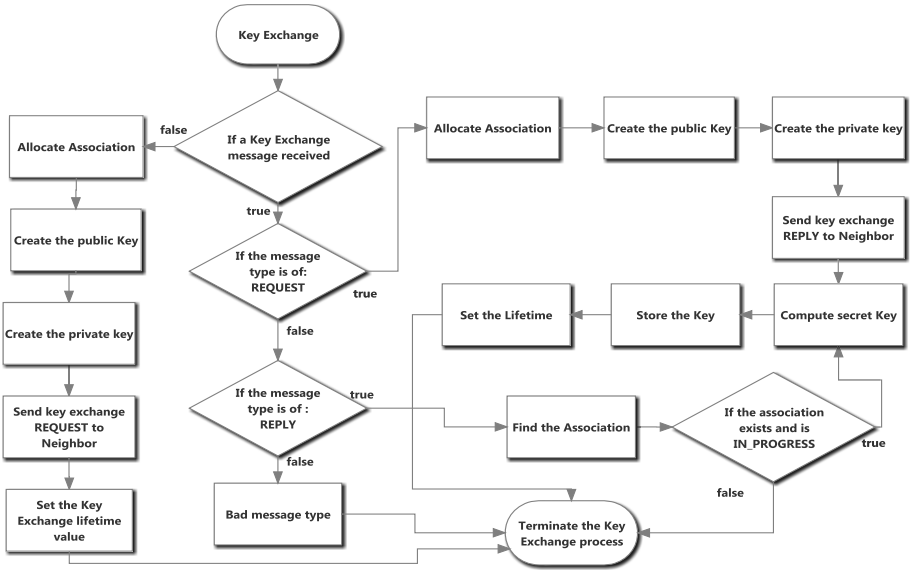


Fig. 3. Flowchart of the Key Exchange Process

negotiation. After the temporary lifetime is initialized, it decreases periodically similarly to the secret key lifetime. If the key-exchange attempt expires, the allocated ACL entry is released and the daemon transitions from the *Awaiting Response* state to *Listening*.

4 Experimental Setup, Results and Analysis

We evaluate the key exchange technique with the Cooja simulator[2], which is distributed with the Contiki OS. Default parameters used for our experiments are provided in Table 1. Some experiments use different configuration parameters, in which case the modifications are clearly discussed in the text. Cooja can emulate motes at the hardware level, allowing precise inspection of system behavior. The sky platform is very commonly used and very well supported by Cooja and for that reason it has been chosen for our experiments.

Table 1. Simulation Configuration

Parameter	Value
Motes	Tmote Sky
TX Range	50m
MAC Layer	IEEE 802.15.4
Radio Access	CSMA
Duty Cycling	ContikiMAC
Max Neighbors	4
ACL Size	4
ACL Entry Lifetime	1000, 1500, ..., 2500 secs
Key Exchange Lifetime	50 seconds

4.1 Memory Requirements

The memory requirements of the key exchange technique are presented in Table 4.1. For implementing the ECDH key exchange method we use the 16-bit mode of ContikiECC's libraries (line *nn* in the table). Moreover, we use the Standards for Efficient Cryptography Group (SECG) standardized elliptic curve *SECP128R1*, by defining curve parameters a and b and its base point. The *ecdh* line relates to the ECDH daemon, which implements the ACL table, provides the key management mechanism and handles the process of key exchange.

Results show that we spend about 7.7 Kb of the device's ROM memory for source code hosting and about 1.1 Kb of RAM for storing curve parameters and ACL associations.

[2] `http://www.contiki-os.org/start.html#start-cooja`

Table 2. Memory and Code Footprints in Bytes

Code Module	ROM	RAM		Overall
		data	bss	
nn	3372	0	0	3372
ecc	2494	0	676	3170
ecdh	1477	10	392	1879
secp128r1	402	0	0	402
Total	7745	10	1068	8823

4.2 Latency, Average Energy Consumption and Network Scalability

The first and simplest experiment consists of only two nodes that communicate for a long time period so that many secret key re-establishments can take place. This experiment investigates energy consumption and computation times required for the calculation of public and secret shared keys as well as for a full key exchange process.

In this example, key exchange always begins with Alice, while Bob is always the node receiving Alice's request and has to reply. The communication is performed properly as long as the lifetime of the secret key has not reached the value of zero. When the secret key expires a new secret key must be established for further communication.

The time needed for the creation of the public key and the computation of the secret key, as well as the overall time for the key exchange process is given in Table 3(a). Table 3(b) presents the energy Alice and Bob consume to create their public key and the shared secret key. It also presents the overall energy consumption for the full key exchange process.

To estimate energy consumption, we use Contiki's energest module. We measure the time each node spent in each of the following three states: i) Micro-Controller Unit (MCU) active, ii) RF listening / receiving (RX), iii) RF transmitting (TX). Since we are simulating sky motes, we then converted these time

Table 3. Average Time and Energy Consumption of the ECDH Key Exchange Process

(a) Average Time Consumption (seconds)

	Public key	Secret key	Key Exchange Process
Alice	8.560	8.547	25.586
Bob	8.457	8.416	16.873

(b) Average Energy Consumption (mJ)

	Public key	Secret Key	Key Exchange Process	
			MCU	MCU + TX + RX
Alice	47.176	47.226	97.021	113.224
Bob	47.165	47.171	94.353	100.758

values to estimated energy consumption based on typical datasheet power levels at an operating voltage of 3.0V. Energy spent for the creation of the public and the secret key is calculated based solely on microcontroller activity. The total for the key exchange also takes into account consumption attributed to the radio transceiver.

In terms of network scalability, default values configure ACLs to hold a maximum of four concurrent key associations. Each increase to the maximum ACL size by one increases the table's memory footprint by 84 bytes, as illustrated in Fig. 4. With the configuration used for our experiments, we could build working firmware with a table size of up to 36 entries before we started getting linker errors.

This does not mean that the network cannot support more than 36 nodes, but that each sensor can support only up to 36 different secret keys at any time. Bearing in mind that keys are only generated between single-hop neighbors, results demonstrate that it is possible to build very dense networks (each node has 30 or more neighbors) without problems.

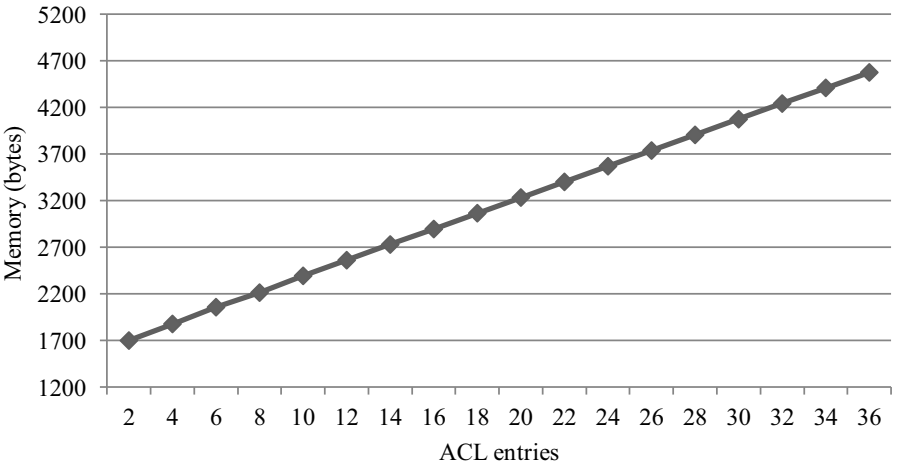


Fig. 4. Scalability with Network Density

4.3 Energy Consumption and Key Lifetime

In this experiment we build a network of ten nodes, where one of them acts as a simple UDP sink node (server) and the rest act as clients. Four of the client nodes are out of the server's transmission range and thus the communication between them and the server is conducted through intermediate routers.

We perform multiple simulation runs, starting with a key lifetime value of 1000 seconds and for each consecutive run the value increases by 500 seconds. Simulations run for a period of two hours so that the keys expire multiple times and many key negotiations take place.

Fig. 5 presents the energy consumption of each network device, for a deployment without key exchange support and for key lifetime values of 1000 and 2500 seconds. The bars in the graph show total energy consumption, with bar portions illustrating consumption attributed to micro-processor activity, RF listening/reception and RF transmission.

Both the microcontroller's and the overall energy consumption of the server device (node ID: 0) is very high in contrast to the estimated energy consumption of the other devices. This happens because the server node has more connections and is required to establish multiple secret keys. Transmission at the server consumes the smallest amount of energy in contrast to the other nodes and reception is the highest. This is due to the functionalities the node implements, as the server transmits only its public key during the key exchange process, while the other nodes transmit their keys as well as application layer data.

Nodes acting as internal routers (node IDs 6-9) consume a large amount of energy in relation to the remaining nodes. The energy consumed by its microprocessor was spent not only to establish a secret key with the server, but also with the client nodes it serves. Similarly, the energy consumed by transmission is due to sending its own application layer messages as well as routing application layer messages towards the server.

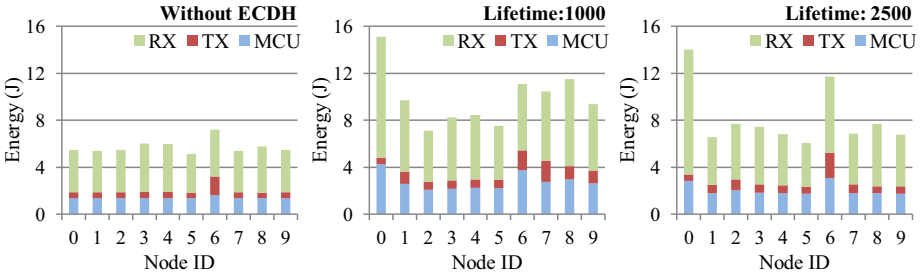


Fig. 5. Average Energy Consumption without ECHD and with ECDH for Key Lifetime Values of 1000 and 2500 Seconds

Results presented in Fig. 6 show that energy consumed by node microprocessors decreases as key lifetime increases. This happens because the keys are valid for a longer time frame and thus, fewer key negotiations are needed. The energy consumption of the server's receiver is fluctuating but remains high, because while the number of the key exchanges decreases, the number of received messages increases and thus the receiver remains busy.

The energy spent by router MCU is lower than the server's MCU energy consumption and higher than the client's average MCU consumption, as it is related to the number of the key exchanges it performs. However, as key lifetime increases, MCU energy consumption decreases.

It is observed that the average energy consumption of the micro-processor of the clients decreases as key lifetime increases. The same phenomenon applies

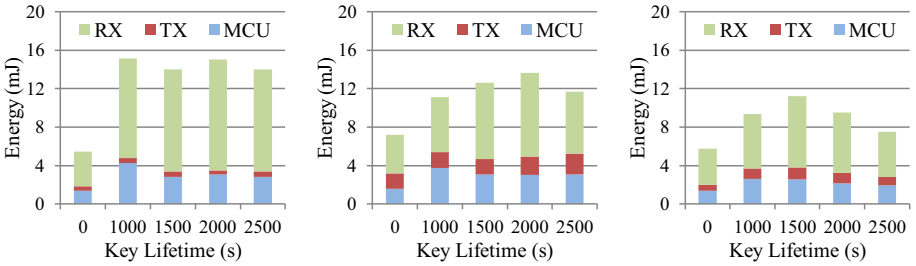


Fig. 6. Energy Consumption vs Key Lifetime.: Server *ID:0* (left), Router *ID:6* (middle); Clients (right).

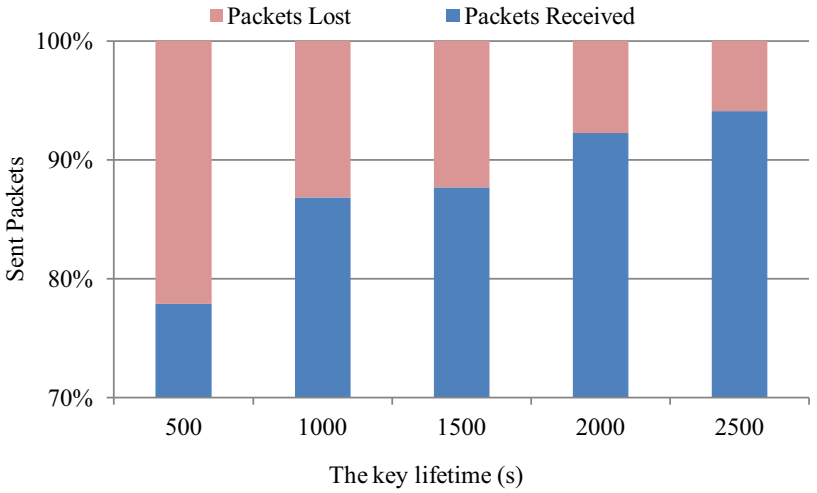


Fig. 7. Packet Loss vs Key Lifetime

to the energy consumed during transmission. However, consumption due to reception increases at the key lifetime of 1500 seconds, for the specific network setup.

4.4 Key Lifetime and Packet Loss

Outgoing packets are dropped at layer 2 when a secret key does not exist in the ACL table and in the case that the key exchange process is in progress. By dropping the outgoing packets the overall performance of the network is affected. However, the existence of the secret key is determined by the lifetime value, which defines whether an entry in the ACL table is valid or not.

To assess how the lifetime affects the network performance we repeat the previous simulations and measure packet loss, with results for various key lifetime values illustrated in Fig. 7. When key lifetime is set at 500 seconds the packets

lost due to the lack of the key are more than 22 percent of the outgoing packets. This happens because key lifetime is short and nodes need to re-negotiate keys relatively frequently. The packet-loss ratio decreases significantly as the key lifetime increases. When the lifetime is set to 1000 and 1500 the packet-loss of the networks is 13% and 12% respectively. Eventually, when the lifetime is set to 2000 and 2500 seconds the packet-loss gets lower than 10% and 5% respectively.

5 Conclusions and Further Work

We developed a software implementation of the ECDH key exchange algorithm in order to assess its applicability and viability for 6LoWPANs and to examine its impact on network performance. Our method performs key exchange between two parties, but it also handles the ACL table and manages the established secret keys. Our implementation requires approximately 7.7 KBytes of code memory when built with the MSP430 GCC toolchain and used with Tmote Sky devices. This indicates that the implementation is somewhat too large for legacy sensor devices. However, contemporary devices incorporate considerably larger flash storage (e.g. 512KB) and 32-bit MCUs, allowing a significantly extended memory space. This alleviates address space restrictions posed by the original MSP430 and subsequent MSP430X architectures.

Average energy consumption attributed to microcontroller activity for the computation of the public and the secret key (elliptic curve 128-bit scalar multiplication) is less than 48 mJ, and about 96 mJ for the entire key exchange negotiation. Total energy consumption is estimated to be approximately 115 mJ, with the increase being primarily attributed to the radio transceiver in listening and frame reception modes.

By simulating a larger network, we investigated the impact of the key lifetime to the overall energy consumption, network performance and scalability. We observe that packet loss decreases as key lifetime increases. This is due to two factors: i) In our current implementation, lack of a key with a neighbor results in an outgoing packet getting dropped and ii) Key negotiation itself is time-consuming, occasionally leading to further losses. Packet delivery ratio can be improved by pro-actively triggering the establishment of a new key between two neighbors before the existing key times out.

Our scalability investigation reveals that each network node is able to keep up to 36 distinct keys in its ACL table. The approach scales well even in very dense networks; this number is sufficiently large if we keep in mind that each node only needs to establish keys with its single-hop neighbors.

The current solution is unauthenticated and is thus vulnerable to man-in-the-middle attacks, similar to the original Diffie Hellman key exchange. To address this, a method to authenticate the two parties before key negotiation would be required, with the Elliptic Curve DSA (ECDSA) algorithm posing as an attractive candidate. This is left as future work.

In terms of the time required to negotiate a shared secret, the algorithm can be further optimised. As part of our future work, we will adjust the ECDH daemon to generate Public-Private key pairs pro-actively, during node idle periods,

instead of on-demand when a new key request is received. As a result, the only computationally intensive operation that will need to be conducted during key negotiation will be the calculation of the shared secret, which can take place in parallel at the two participating parties. We estimate that this will reduce the total negotiation time by approximately 60%.

In the future, ECC hardware acceleration can be employed to make the approach more viable. Hardware acceleration can have a host of positive effects: i) It can decrease code footprint, since it would mean that the implementation of elliptic curve calculations would no longer need to be included in firmware, ii) Key negotiation will be considerably faster, since ECC calculations will not need to be performed by software, iii) Assuming energy-efficient acceleration hardware, total energy consumption for the negotiation will be lower.

On contemporary sensor devices, encryption of link-layer frames is conducted by AES co-processors and is generally considered to be fast and energy-efficient. However, it is impossible to simulate hardware acceleration within the Cooja simulator. Providing a software AES implementation would have had an obfuscating effect on all metrics under investigation and since this work focused on key negotiation, we deliberately removed layer two encryption functionality in its entirety. As part of our future plans, we will evaluate the method in a real testbed with layer two encryption enabled and performed by hardware. This will allow us to draw conclusions on the viability of the complete solution.

References

1. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, Washington, USA, pp. 263–270 (1999)
2. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. 35, 93–104 (2000)
3. Dunkels, A.: Full TCP/IP for 8-bit architectures. In: Proceedings of the 1st International Conference on Mobile systems, Applications and Services, New York, pp. 85–98 (2003)
4. Hui, J.W., Culler, D.E.: IP is dead, long live IP for wireless sensor networks. In: Proc. 6th ACM Conference on Embedded Network Sensor Systems (SenSys 2008), New York, NY, USA, pp. 15–28 (2008)
5. Montenegro, G., Kushalnagar, N., Hui, J., Culler, D.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944 (2007), `http://tools.ietf.org/html/rfc4944`
6. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
7. Zhang, J., Varadharajan, V.: Wireless sensor network key management survey and taxonomy. J. Netw. Comput. Appl. 33, 63–75 (2010)
8. Roman, R., Alcaraz, C., Lopez, J., Sklavos, N.: Key management systems for sensor networks in the context of the Internet of Things. Computers & Electrical Engineering 37, 147–159 (2011)

9. Winter, T. (ed.), Thubert, P. (ed.), Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J.P., Alexander, R.: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (2010), http://tools.ietf.org/html/rfc6550
10. Çamtepe, S.A., Yener, B.: Combinatorial design of key distribution mechanisms for wireless sensor networks. IEEE/ACM Trans. Netw. 15, 346–358 (2007)
11. Bechkit, W., Challal, Y., Bouabdallah, A., Tarokh, V.: A Highly Scalable Key Pre-Distribution Scheme for Wireless Sensor Networks. IEEE Transactions on Wireless Communications 12(2), 948–959 (2013)
12. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM, New York (2002)
13. Wang, H., Li, Q.: Efficient Implementation of Public Key Cryptosystems on Mote Sensors (Short Paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 519–528. Springer, Heidelberg (2006)
14. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
15. Wander, A.S., Gura, N., Eberle, H., Gupta, V., Shantz, S.C.: Energy analysis of public-key cryptography for wireless sensor networks. In: Third IEEE International Conference on Pervasive Computing and Communications - PerCom 2005, pp. 324–328 (2005)
16. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management part 1: General (revision 3). NIST special publication 800, 57 (2011)
17. Bianchi, G., Capossele, A.T., Mei, A., Petrioli, C.: Flexible key exchange negotiation for wireless sensor networks. In: Proceedings of the 5th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, pp. 55–62. ACM, New York (2010)
18. Mzid, R., Boujelben, M., Youssef, H., Abid, M.: Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography. In: 2010 International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), Sousse, pp. 1–8 (2010)
19. Sustainable Computing Research (SCoRe) - ContikiECC, http://score.ucsc.lk/projects/contikiecc
20. Liu, A., Ning, P.: TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN 2008, pp. 245–256 (2008)

URANOS: User-Guided Rewriting for Plugin-Enabled ANdroid ApplicatiOn Security

Daniel Schreckling, Stephan Huber, Focke Höhne, and Joachim Posegga

Institute of IT-Security and Security Law
University of Passau, Innstraße 43, Passau, Germany
{ds,sh,fh,jp}@sec.uni-passau.de

Abstract. URANOS is an Android application which uses syntactical static analysis to determine in which component of an Android application a permission is required. This work describes how the detection and analysis of widely distributed and security critical ad-ware plugins is achieved. We show, how users can trigger bytecode rewriting to (de)activate selected or redundant permissions in Android applications without sacrificing functionality. The paper also discusses performance, security, and legal implications of the presented approach.

1 Introduction

Many Smartphone operating systems associate shared resources with permissions. API calls accessing such resources require permissions to gain the required privileges. Once an application obtains these privileges, it can generally access all the items stored in the respective resource. Additionally, such privileges are often valid until the deinstallation or an update of the application. These properties conflict with the emerging privacy needs of users. Increasing sensitivity encourages the protection of data which helps applications, vendors, or providers to generate individual user profiles. Unfortunately, current coarse grained permission systems only provide limited control or information about an application. Hence, informed consents to the use of permissions are far from being available.

In Android, numerous analyses of permissions requested by an application [3,11,14,20,21] substantiate this problem. Permissions increase the attack surface of an application [4,2,12] and the platform executing it. Thus, granting permissions in excessive manners induces new exploit techniques. Static analysis and runtime monitoring frameworks have been developed to detect permission-based platform and application vulnerabilities. There are also Android core extensions enabling the deactivation of selected permissions. However, such frameworks either interfere with the usability of the application and render it unusable or they only provide permission analysis on separate hosts.

Thus, there is a strong need for flexible security solutions which do not aim at generality and precision but couple lightweight analysis and permission modification mechanisms.

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 50–65, 2013.

We define URANOS, an application rewriting framework for Android which enables the selective deactivation of permissions for specific application contexts, e.g. plugins. The contributions of this paper include an on-device static analysis to detect permissions and their usage, selective on-device rewriting to guarantee user-specific permission settings, and a prototype implementing detection and rewriting in common Android applications.

Our contribution is structured as follows: Section 3 provides a knowledge base for this contribution, Section 2 gives a high-level overview of URANOS. Its components are explained in Section 4. Section 5 discusses performance, limitations, and legal implications. Finally, Section 6 lists related work before Section 7 summarises our conclusions.

2 Approach Overview

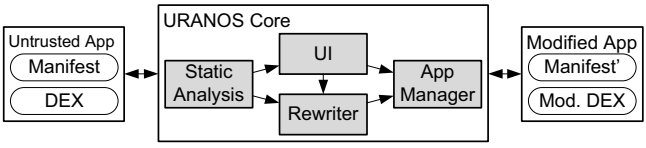


Fig. 1. High-level overview of URANOS

We strive for an efficient on-device framework (see Figure 1) for Android which allows users to selectively disable permissions assigned to an application. To preserve functionality a static analysis infers the permissions required during execution from the bytecode. For efficiency we exploit existing knowledge about the permission requirements of Android API calls, resource access, intent broadcasting etc. Detected permissions are compared with the permissions requested in the application manifest to detect excessive permissions etc. Additionally, we scan the bytecode for plugins using a pre-generated database of API methods and classes used in popular ad-ware. They define context for each bytecode instruction. This allows us to infer the permissions exclusively required for plugins or for the application hosting the plugins. We communicate this information to the user. Depending on his needs, the user can enable or disable permissions for specific application contexts.

Disabled and excessive permissions can be completely removed from the manifest. However, removing an effectively required permission will trigger a security exception during runtime. If these exceptions are unhandled the application will terminate. Therefore, URANOS additionally adapts the application bytecode and replaces the API calls in the respective call context by feasible wrappers.

This combination of analysis and rewriting allows a user to generate operational applications compliant with his security needs. Unfortunately, compliant but rewritten Android applications are neither directly installed nor are they

updated by Android. Therefore, URANOS also delivers an application manager service, replacing applications with their rewritten counterparts and ensuring their updates.

3 Background

This section gives a short overview of the structure of Android applications, their execution environment, and the permission system in Android.

3.1 Android Applications

Applications (Apps) are delivered in zipped packages (`apk` files). They contain multimedia content for the user interface, configuration files such as the manifest, and the bytecode which is stored in a Dalvik executable (`dex` file). Based on the underlying Linux, Android allots user and group IDs to each application.

Four basic types of components can be used to build an App: *activities*, *services*, *content providers*, and *broadcast receivers*. *Activities* constitute the user interface of an application. Multiple activities can be defined but only one activity can be active at a time. *Services* are used to perform time-consuming or background tasks. Specific API functions trigger remote procedure calls and can be used to interact with services. Application can define *content providers* to share their structured data with other Apps. To retrieve this data, so called *ContentResolvers* must be used. They use URIs to access a provider and query it for specific data. Finally, *broadcast receivers* enable applications to exchange *intents*. Intents express an *intent* to perform an action within or on a component. Actions include the display of a picture or the opening of a specific web page.

Developers usually use these components defined in the Android API and the SDK to build, compile, and pack Apps. Their `apks` are signed with the private developer key, distributed via the official Android market, other markets, or it is delivered directly to a Smartphone.

3.2 Dalvik Virtual Machine

Bytecode is stored in Dalvik executables (`dex` files) and is executed in a register based virtual machine (VM) called Dalvik. Each VM runs in its own application process. The system process Zygote starts at boot time and manages the spawning of new VMs. It also preloads and preinitialises the core library classes.

`Dex` files are optimised for size and data sharing among classes. In contrast to standard Java archives, the dex does not store several class files. All classes are compiled into one single `dex` file. This reduces load time and supports efficient code management. Type-specific mappings are defined over all classes and map constant properties of the code to static identifiers, such as constant values, class, field, and method names. The bytecode can also contain developer code not available on the platform, e.g. third-party code, such as plugins (see Figure 2).

Bytecode instructions use numbered register sets for their computations. For method calls, registers passed as arguments are simply copied into new registers only valid during method execution.

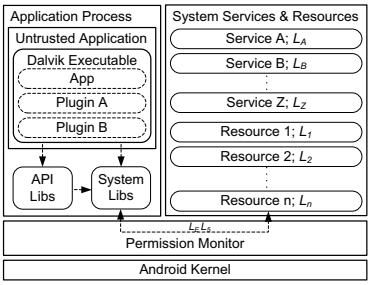


Fig. 2. Plugins and Permissions in Android Applications

3.3 Android Permissions

Android permissions control application access to resources. Depending on their potential impact Android distinguishes three levels: *normal*, *dangerous*, *signature* and *signatureORsystem*. Unlike normal permission which do not directly cause financial harm to users, dangerous and system permission control access to critical resources and may enable access to private data. Granted signature or signaturesORsystem permission grant access to essential system services and data. During installation permissions are assigned as requested in the manifest. The user only approves dangerous permissions. Normal permissions are granted without notification and signature or signatureORsystem permissions verify that the application requesting the permissions has been signed with the key of the device manufacturer.

Resource access can be obtained through API calls, the processing of intents, and through access to content providers and other system resources such as an SD card. Thus, permission enforcement varies with the type of resource accessed. In general, permission assignment and enforcement can be described using a label model as depicted in Figure 2. Each system resource or system service is labeled with the set of permissions it requires to be accessed. An application uses the public API to trigger resource access. This request is forwarded to the system. The system libraries, the binder, and the implementation of the libraries finally execute the resource access. We abstract from the details of the binder-library pair and call this entity a central *permission monitor*. It checks whether an application trying to access a resource with label L_x has been assigned this label. If not, access is forbidden and an appropriate security exception is thrown.

Android also places permission checks in the API and RPC calls [9]. Thus, security exceptions may already occur although the access requests have not reached the permission monitor, yet. As such checks may be circumvented by reflection the actual enforcement happens in the system.

4 The URANOS Framework

This section explains our system in more detail. To ease the understanding we complement our description with Figure 3.

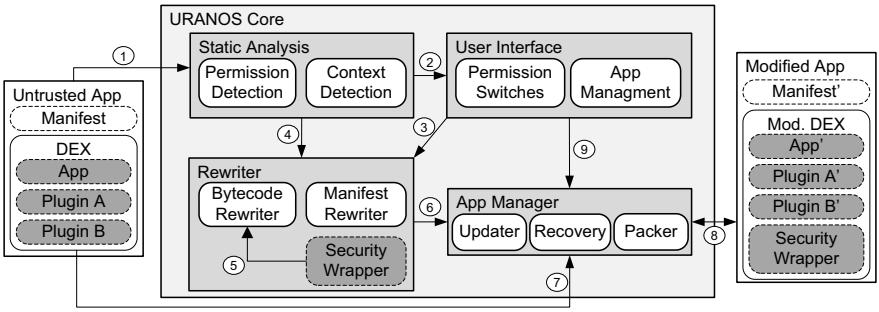


Fig. 3. System Overview

4.1 Application Processing

To process manifest and bytecode of the application, URANOS must obtain access to the `apks`. Depending on how the developer decides how to publish an APK, it is stored in different file system locations: the regular application storage, the application storage on an SD card, or storage which prevents the forwarding (forward-lock) of the application. The `PackageManager` API offered by Android can be used to retrieve the path and filename of the `apks`.

Regular applications are able to obtain reading access to other `apks`. Thus, as a regular application, URANOS can copy `apks` to a local folder and process them. With root permissions, it can also process forward-locked applications.

`Apks` are extracted to obtain access to the manifest and the `dex` file. We enhanced the `dex-tools` from the Android source tree. It directly operates on the bytecode and can extract information required for our analysis. Thus, we avoid intermediate representations. Handles to manifest and bytecode are forwarded (1) to the static analysis and rewriting components of our framework.

4.2 Permission Detection

Next, we parse the manifest and retrieve the set P_{apk} of permissions requested by the App. Afterwards, we scan the bytecode to find all invoke instructions and determine the correct signature of the methods invoked. Invoke instructions use identifiers pointing to entries of a management table which contains complete method signatures. From this table we derive the set I of methods potentially invoked during execution. As this is a syntactical process set I may contain methods which are never called.

We then use function π to compute $P_M = \bigcup_{\forall m \in I} \pi(m)$, i.e. π maps method m to a set of permissions required to invoke m at runtime. Thus, P_M reflects the permissions required by the application to execute all methods in I. Function π is based on the results of Felt et al. [9] which associate actions in an Android App with its required permissions, e.g. method calls.

The use of content providers or intents may also require permissions. However, specific targets of both can be specified using ordinary strings. To keep our

analysis process simple we search the `dex` for strings which map the pattern of a content provider URI or of a activity class name which is defined in the Android API. If a pattern is matched, we add the respective permission to the set P_P of provider permissions or to the set P_I of intent permissions, respectively.

At the end of this process we intersect the permissions specified in the manifest with the permissions extracted from the bytecode, i.e. $P_{val} = P_{apk} \cap (P_M \cup P_P \cup P_I)$ to obtain the validated permissions likely to be required for the execution of the application. Our heuristics induce an over-approximation in this set. Section 5 explains why it does not influence the security of our approach.

4.3 Context Detection

Based on P_{val} we now determine the App components in which the methods requiring these permissions are called. For this purpose we define the *execution context* for an instruction. It is the signature of the method and its class in which the instruction is executed. This definition is generic and can be applied to various detection problems. We focus on widely distributed plugins for Android.

To give users a better understanding on the possible impact of the plugins hosted by the analysed Apps we manually assign each plugin to the following four categories: *passive*, *active*, *audio advertising*, and *feature extensions*.

Passive advertising plugins display advertisements as soon as an activity of the hosting application is active. They are usually integrated into the user interface with banners as placeholders.

Active advertising plugins are similar to pop-up windows and do not require a hosting applications. They use stand alone activities or services, intercept intents, or customise events to become active.

Audio advertising is a rather new plugin category which intercepts control sequences and interferes with the user by playing audio commercials or similar audio content, e.g. while hearing the call signal on the phone.

Feature extensions include features in an application a user or developer may utilise. Among many others, they include in-app billing or developer plugins easing the debugging process.

To detect plugins in an application, we perform the same steps required for archiving the signatures. We scan the application manifest and bytecode for the names listed above and investigate which libraries have to be loaded at runtime. From this information we build a signature and try to match it against our plugin database. This process also uses fuzzy patterns to match the strings inferred from the application. We assume that plugins follow common naming conventions. So, full class names should start with the top Internet domain name, continue with the appropriate company name, and end with the class names. If we do not find matches on full class names, we search for longest common prefixes. If they contain at least two subdomains, we continue searching for the other names to refine the plugin match. In this way we can account for smaller or intentional changes in class or package naming and prevent a considerable decline of the detection rate.

The ability to detect classes of plugins allows us to determine execution contexts. During the bytecode scanning, we track the context C. As soon as our analysis enters the namespace of a plugin class, we change C. It is defined by the name of the plugin N_{Plugin} or by the name N_{apk} of the application if no plugin matches. We generate a map for each method call to its calling context. Together with the function π, this implicitly defines a map γ from permissions to calling contexts. We can now distinguish four types of permissions:

Dispensable permissions $p \in P_{apk} \setminus P_{val}$ are not required by the application,
Application only permissions $p \in P_{apk}$ are exclusively required for the hosting application to run, i.e. $\gamma(p) = \{N_{apk}\}$,
Plugin only permissions $p \in P_{apk}$ are exclusively required for the execution of a plugin, i.e. $\gamma(p) \cap \{N_{apk}\} = \emptyset$, and
Hybrid permissions $p \in P_{apk}$ which are required by both, the hosting application and the plugin, i.e. $\gamma(p)$ does not match the conditions for the other three permission types.

This result is communicated to the user in step (2). He gets an overview of the types of permissions and the context in which they are required. The user can enable or disable them in the entire application, only in the plugin, or only in the hosting application. The next section shows how to support this feature with the help of bytecode rewriting and without modifying Android.

4.4 Rewriter

In general, dispensable permissions are not required for the execution and don't need to be assigned to the application. They can removed from the manifest. The same holds for permissions which should be disabled for the entire application. Thus, the first rewriting step is performed on the application manifest. It revokes the permissions either not required or not desired.

However, withdrawing permissions from an application may render it unusable. Calls to methods which require permissions will throw exceptions. If they are not handled correctly, the runtime environment could finally interrupt execution. To avoid this problem, enable the deactivation of permissions in only specific application components, and to retain an unmodified Android core, the activation or deactivation of permissions triggers a rewriting process (3). It is guided by the results of the syntactical analysis (4). The rewriter, described in this section, adapts the bytecode in such a way that the App can be executed safely even without the permissions it originally requested.

API Methods. For each method whose execution requires a permission, we provide two types of wrappers (5) to replace the original call. Regular API method calls which require a permission, can be wrapped by simple `try` and `catch` blocks as depicted by `WRAPPER1` in Listing 1.1. If the permission required to execute the API call has been withdrawn, we catch the exception and return a feasible default value. In case the permission is still valid, the original method

is called. In contrast, the second wrapper `WRAPPER2` (Listing 1.2) completely replaces the original API call and only executes a default action.

Listing 1.1. Wrapper pattern one

```
public static WRAPPER1 {
  try {
    API_CALL_ACTION;
  } catch (SecurityException se) {
    DEFAULT_ACTION;
  }
}
```

Listing 1.2. Wrapper pattern two

```
public static WRAPPER2 {
    DEFAULT_ACTION;
}
```

Evidently, rewriting could be reduced to only `WRAPPER2`. But, `WRAPPER1` reduces the number of events at which an application has to be rewritten and reinstalled. Assume that a user deactivates a permission for the entire application. The permission is removed from the manifest and all methods requiring it are wrapped. Depending on the wrapper and the next change in the permission settings a rewriting may be avoided because the old wrapper already handles the new settings, e.g. the reactivation of the permission.

Wrappers are static methods and apart from one additional instance argument for non-static methods, they inherit the number and type of arguments from the methods they wrap. This makes it easy to automatically derive them from the API. Additionally, it simplifies the rewriting process as follows.

URANOS delivers a `dex` file which contains the bytecode of all wrappers. This file is merged with the original application `dex` using the `dx` compiler libraries. The new `dex` now contains the wrappers but does not make use of them, yet. In the next step we obtain the list of method calls which need to be replaced from the static analysis component (4). The corresponding invoke instructions are relocated in the new `dex` and the old method identifiers are exchanged with the identifiers of the corresponding wrapper methods.

Here, the rewriting process is finished even if the wrapped method is non-static. At bytecode level, the replacement of a non-static method with a static one simply induces a new interpretation of the registers involved in the call. The register originally storing the object instance is now interpreted as the first method argument. Thus, we pass the instance register to the wrapper in the first argument and can leave all other registers in the bytecode untouched. We illustrate this case in Listing 1.3. It shows bytecode mnemonics for the invocation of the API method `getDeviceId` as obtained by a disassemblers.

Listing 1.3. Regular API call

```
invoke-virtual {v0},
  Landroid/telephony/
      TelephonyManager;
    .getDeviceId:()
       Ljava/lang/String;
```

Listing 1.4. Rewritten API call

```
invoke-static {v0},
  Lde/wrapper/Wrapper;
    ._getDeviceId:(Landroid/telephony/
        TelephonyManager;)
       Ljava/lang/String;
```

The instruction `invoke-virtual` calls the method `getDeviceId` on an instance of class `TelephonyManager`. It is rewritten to a static call in Listing 1.4 and passes the instance as an argument to the static wrapper method.

Reflection. Android supports reflective method calls. They use strings to retrieve or generate instances of classes and to call methods on such instances.

These operations can be constructed at runtime. Hence, the targets of reflective calls are not decidable during analysis and calls to API methods may remain undetected. Therefore, we wrap the methods which can trigger reflective calls, i.e. `invoke` and `newInstance`. During runtime, these wrappers check the `Method` instance passed to `invoke` or the class instance on which `newInstance` is called. Depending on its location in the bytecode the reflection wrapper is constructed in such a way that it passes the invocation to the appropriate wrapper methods (see above) or executes the function in the original bytecode. This does not require dynamic monitoring but can be integrated in the bytecode statically. Reflection calls show low performance and are used very infrequently. Thus, this rewriting will not induce high additional overhead.

Content Providers. Similar to reflective calls, we handle content providers. Providers must be accessed via content resolvers (see Section 3) which forward the operations to be performed on a content provider: `query`, `insert`, `update`, and `delete`. They throw security exceptions if required read or write permissions are not assigned to an application. As these methods specify the URI of the content provider we replace all operations by a static wrapper which passes their call to a monitor. It checks whether the operation is allowed before executing it.

Intents. In general, intents are not problematic as they are handled in the central monitor of Android, i.e. the enforcement does not happen in the application. If an application sends an intent to a component which requires permissions an exception in the error log is generated if the application does not have this permission. The corresponding action is not executed but the application does not crash. Thus, our rewriting must cover situations in which only some instructions in specific execution contexts must not send or receive intents. The control over sending can be realised by wrappers handling the available API methods such as `startActivity`, `broadcastIntent`, `startService`, and `bindService`. The wrappers implement monitors which first analyse the intent to be sent. Depending on the target, the sending is aborted. By rewriting the manifest, we can control which intents a component can receive. This excludes explicit intents which directly address a application component. Here, we assume that the direct access of a system component to an application can be considered legitimate.

4.5 Application Management

We realise permission revocation by repackaging applications. First, our App manager obtains the manifest and `dex` (6) from the rewriter. For recovery, we first backup the old `dex` file and its corresponding manifest. All other resources, such as libraries, images, audio or video files, etc. are not backed up as they remain untouched. They are extracted from the original `apk` (7), signed with the URANOS key together with the new bytecode and manifest. The signed application is then directly integrated into a new `apk`. This process is slow due to the `zip` compression of the archive. In the end, the application manager assists the user to deinstall the old and install the new application (8,9).

In the background we also deploy a dedicated update service. It mimics the update functionality of Android but also operates on the applications resigned by URANOS. We regularly query the application market for updates, inform the user about them, and assists the update process by deinstalling the old App, rewriting the new App, and installing it. Similarly, the App manager provides support for deinstallation and recovery.

5 Discussion

5.1 Performance

To assess the performance of our approach we downloaded over 180 popular applications from the Google Play Store. The URANOS App was adjusted in such a way that it automatically starts analysing and rewriting newly installed applications. Our benchmark measured the analysis time, i.e. the preprocessing of the `dex` (pre) and the execution context detection (det), and the rewriting time, i.e. the merging of wrappers (wrap), the rewriting of the resulting `dex` (rew), and the total time require to generate the final `apk` (tot). The analysis and rewriting phase were repeated 11 times for each App. The first measurement was ignored as memory management and garbage collection often greatly influence the first measurements and hard to reproduce as they heavily depend on the phone state. For the rewriting process, we always selected three random permissions to be disabled. If there were less permissions we disabled all. All measurements were conducted on a Motorola RAZR XT910, running Android 4.0.4 on a 3.0.8 kernel. Due to space restrictions this contribution only discusses a selection of applications and their performance figures. An overview of the complete results, a report on the impact of our rewriting on the App functionality, and the App itself are available at `http://web.sec.uni-passau.de/research/uranos/`.

Apart from the time measurements mentioned above Table 1 enumerates the number of plugins the application contains (#pl), the number of permissions requested (#pm), the number classes (#cl) in the `dex` and the size of the `apk`. In particular the `apk` size has a tremendous impact on the generation of the rewritten application due to APK compression. This provides potential for optimisation in particular if we look at the rather small time required to merge the wrapper file of 81 kB into the complex `dex` structure and redirecting the method calls. This complexity is also reflected in the time for pre-processing the `dex` to extract information required to work on the bytecode.

We can also see that the number of classes and permissions included in an application influence the analysis time. Classes increase the number of administrative overhead in a `dex`. Thus their number also increases the effort to search for the appropriate code locations. Here, Shazam and Instagram are two extreme examples. In turn, the number of permissions increase the number of methods which have to considered during analysis and rewriting.

In our measurements, we do not include execution overhead. The time required for the additional instructions in the bytecode are negligible and within measuring tolerance. Thus, although the generation of the final `apk` is slow, our

Table 1. Selection of analysed and rewritten applications

App	#pl	#pm	#cl	apk[MB]	pre[ms]	det[ms]	wrap[ms]	rew[ms]	tot[ms]
100 Doors	4	3	757	14.4	1421	356	1690	4277	9073
Angry Birds	10	6	873	24.4	1863	640	2308	5767	50408
Bugvillage	13	8	1127	3.1	1819	1214	3425	6832	18092
Coin Dozer	11	6	855	14.7	2028	788	2605	6457	56749
Fruit Ninja	8	8	1472	19.2	2520	1197	3657	7955	144374
Instagram	7	7	2914	12.9	5168	1906	8114	17031	39908
Logo Quiz	3	2	232	9.7	553	96	701	1939	7729
Shazam	8	13	2822	4.4	4098	3214	7837	15182	27263
Skyjumper	3	4	292	0.9	772	257	1222	2991	4106

measurements certify that the analysis and rewriting on Android bytecode can be implemented efficiently on a Smartphone. While other solutions run off-device and focus on precision, such as the web interface provided by Woodpecker [9], URANOS can deliver timely feedback to the user. With this information he can decide about further countermeasures also provided by our system.

5.2 Limitations

As we have already stated above, our analysis uses approximations. In fact, P_M is an over-approximation of the permissions required by method calls, e.g. there may be methods in the bytecode which are never executed. Thus, the mere existence of API calls does not justify a permission assignment to an application. On the other hand $P_P \cup P_I$ is an under-approximation as we only consider strings as a means to communicate via intents or to access resources. There are numerous other ways for such operations, our heuristic does not cover.

Attackers or regular programmers can achieve under-approximations by hiding intent or provider access with various implementation techniques. In this case URANOS will alert the user that a specific permission may not be needed. The user will deactivate the respective permission and no direct damage is caused. Over-approximation can be achieved by simply placing API calls in the bytecode which are never executed. In this case, our analysis does not report the permission mapping to those *dead* API calls to be dispensable. Thus, the over-approximation performed in this round may give the user a wrong sense of security concerning the application. Therefore, URANOS also allows the deactivation of permissions in the hosting application and not only in the plugin.

Attackers may also hide plugin code by obfuscating it, e.g. by renaming all plugin APIs. In this case, URANOS will not detect the plugin. This will prevent the user from disabling permissions for this plugin. In this case, it is still possible to remove permissions for the whole application. Plugin providers which have an interest in the use of their plugins will not aim for obfuscated APIs.

5.3 Legal Restrictions

If software suffices the copyright law's fundamental requirement of originality it is protected by international and national law, such as the Universal Copyright Convention, the WIPO Copyright Treaty grant protection, Article 10 of the international TRIPS agreement of the WTO agreement, and the European Directive 2009/24/EC. In general, these directives prohibit the manipulation, reproduction, or distribution of source code if the changes are not authorised by its rights holder. No consent for modification is required if the software is developed using an open source software licensing model or if *minor* modifications are required for *repair or maintenance*. To achieve *interoperability* of an application even reverse engineering may be allowed. However, any changes must not infringe with the regular *exploitation* of the affected application and the *legitimate interest* of the rights holder.

URANOS cannot satisfy any of the conditions mentioned above. First of all, all actions are performed automatically. Thus, it is not possible to query the rights owner for his permission to alter the software. One may argue that URANOS rewrites the application in order to ensure correct data management. Unfortunately, the changes described above directly infringe with the interest of the rights holder of the application.

On the other hand one argue that a developer must inform the user how the application processes and uses his personal data as highlighted in the *"Joint Statement of Principles"* of February 22^{nd} 2012 and signed by global players like Amazon, Apple, Google, Hewlett-Packard, Microsoft and Research In Motion. However, current systems only allow an informed consent of insufficient quality. In particular when using plugins, a developer would need to explain how user data is processed. But developers only use APIs to libraries without knowing internal details. To provide adequate information about the use of data a developer would have to understand and/or reverse engineer the plugin mechanisms he uses. So, for most plugins or libraries, the phrasing of a correct terms of use is impossible. Yet, this fact does not justify application rewriting. The user can still refuse the installation. If, despite deficient information, he decides to install the software he must stick to the legal restrictions and use it as is.

In short: URANOS and most security systems which are based on application rewriting conflict with international and most national copyright protection legislation. This situation is paradoxical as such systems try to protect private data from being misused by erroneous or malicious application logic. Thus, they try to enforce data protection legislation but are at the same time limited by copyright protection laws.

6 Related Work

This section focuses on recent work addressing permission problems in Android. We distinguish two types of approaches: Analysis and monitoring mechanisms.

6.1 Permission Analysis

One of the first publications analysing the Android permission system is Kirin [8]. It analyses the application manifest and compares it with simple and pre-defined rules. In contrast to URANOS, rules can only describe security critical combinations of permissions and simply prevent an application from being installed.

The off-device analysis in [4] is more sophisticated. It defines attack vectors which are based on design flaws and allow for the misuse of permissions. Chin et al. describe secure coding guidelines and permissions as a means to mitigate these problems. Their tool, ComDroid, can support developers to detect such flaws but it does not help App users in detecting and mitigating such problems.

This lack of user support also holds Stowaway [9]. This tool is focused on permissions which are dispensable for the execution of an application. Comparable to URANOS, Stowaway runs a static analysis on the bytecode. However, this analysis is designed for a server environment. While it provides better precision through a flow analysis, it can not correct the detected problems and the analysis times exceed those of URANOS by several magnitudes.

Similar to Stowaway, AndroidLeaks [11] uses an off-device analysis which detects privacy leaks. Data which is generated by operations which are subject to permission checks are tracked through the application to data sinks using static information flow analysis. AndroidLeaks supports the human analyst. The actual end user can not directly benefit from this system.

DroidChecker [3] and Woodpecker [12] use inter-procedural control flow analyses to look for permission vulnerabilities, such as the confused deputy (CD) vulnerability. However, DroidChecker additionally uses taint tracking to detect privilege escalation vulnerabilities in single applications while Woodpecker targets system images. Techniques applied in Woodpecker were also used to investigate the functionality of in-app advertisement [13]. URANOS is based on an extended collection of advertisement libraries used in this work. Similar analytical work with a less comprehensive body has been conducted in [20].

6.2 Enhanced Permission Monitoring

An early approach which modifies the central security monitor in Android to introduce an enriched permission system of finer granularity is Saint [17]. However, Saint mainly focuses on inter-application communication. CRePE [5] goes one step further and extends Android permissions with contextual constraints such as time, location, etc. However, CRePE does not consider the execution context in which permissions are required. Similar holds for Apex [16]. It manipulates the Android core implementation to modify the permissions framework and also introduces additional constraints on the usage of permissions.

Approaches such as QUIRE [7] or IPC inspection of Felt et al. [10] focus on the runtime prevention of CD attacks. QUIRE defines a lightweight provenance system for permissions. Enforcement in this framework boils down to the discovery of chains which misuse the communication to other apps. IPC inspection solves this problem by reinstantiating apps with the privileges of their callers.

Both approaches require an OS manipulation and consider an application to be monolithic. This prevents them from recognising execution contexts for permissions. The same deficiencies hold for XManDroid [2] which extends the goal of QUIRE and IPC inspection by also considering colluding applications.

Similar to IPC inspection and partially based on QUIRE is AdSplit [19]. It targets advertisement plugins and also uses multi-instantiation. It separates the advertisement and its hosting application and executes it in independent processes. Although mentioned in their contribution Shekhar does not aim at deactivating permissions in one part of the application or at completely suppressing communication between the separated application components.

Leontiadis et al. also do not promote a complete deactivation of permissions and separate applications and advertising libraries to avoid over-privileged execution [15]. A trade-off between user privacy and advertisement revenue is proposed. A separated permission and monitoring system controls the responsible processing of user data and allows to interfere with IPC if sufficient revenue has been produced. URANOS could be coupled with such a system by only allowing the deactivation of permissions if sufficient revenue has been produced. However, real-time monitoring would destroy the lightweight character of URANOS.

Although developed independently, AdDroid [18] realises a lot of the ideas proposed by Leontiadis et al. AdDroid proposes specific permissions for advertisement plugins. Of course, this requires modifications to the overall Android permission system. Further, to obtain a privilege separation, AdDroid also proposes a process separation of advertisement functionalities from the hosting application. Additionally, the Android API is proposed to be modified according to the advertisement needs. It remains unclear how such a model should be enforced. The generality of URANOS could contribute to such an enforcement.

Two approaches which are very similar to URANOS are I-Arm Droid [6] and AppGuard [1]. Both systems rewrite Android bytecode to enforce user defined policies. I-Arm Droid does not run on the Android device and is designed to enforce developer defined security policies enforced by inlined reference monitors at runtime. The flexibility of the inlining process is limited as all method calls are replaced by monitors. Selective deactivation of permissions is not possible. The same holds for AppGuard. While it can be run directly on the device the rewriting process replaces all critical method calls. AppGuard compares to URANOS as it uses a similar resource and user friendly deployment mechanism which do not require root access on the device.

7 Conclusions

The permission system and application structure in today's Smartphones do not provide a good foundation for an informed consent of users. URANOS takes a first step into this direction by providing enhanced feedback. The user is able to select which application component should run with which set of permissions. Thus, although our approach can not provide detailed information about its functionality the user benefits from a finer granularity of permission assignment.

If in doubt, he is not confronted with a all or nothing approach but can selectively disable critical application components. The execution contexts we define in our work are general and can describe many different types of application components. Further, we neither require users to manipulate or root their Smartphones. Instead we maintain the regular install, update, and recovery procedures.

Our approach is still slow when integrating the executable code into a fully functional application. However, this overhead is not directly induced by our efficient analysis or rewriting mechanisms. In fact, we highlighted the practical and security impact of a trade-off between a precise and complete flow analysis and a lightweight but fast and resource saving syntactical analysis which can run on user device without altering its overall functionality.

Acknowledgements. The research leading to these results has received funding from the European Union's FP7 project COMPOSE, under grant agreement 317862.

References

1. Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: AppGuard - Real-time policy enforcement for third-party applications. Tech. Rep. A/02/2012, Saarland University (2012)
2. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.R.: XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. Technical Report TR-2011-04, Technische Universität Darmstadt (April 2011)
3. Chan, P.P., Hui, L.C., Yiu, S.M.: Droidchecker: analyzing android applications for capability leak. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC 2012, pp. 125–136. ACM, New York (2012)
4. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.: Analyzing inter-application communication in android. In: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys 2011, pp. 239–252. ACM, New York (2011)
5. Conti, M., Nguyen, V.T.N., Crispo, B.: CRePE: context-related policy enforcement for android. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 331–345. Springer, Heidelberg (2011)
6. Davis, B., Sanders, B., Khodaverdian, A., Chen, H.: I-arm-droid: A rewriting framework for in-app reference monitors for android applications. In: IEEE Mobile Security Technologies (MoST), San Francisco, CA
7. Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., Wallach, D.S.: QUIRE: Lightweight Provenance for Smart Phone Operating Systems. CoRR abs/1102.2445 (2011)
8. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: CCS 2009: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235–245. ACM, New York (2009)
9. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: ACM Conference on Computer and Communications Security, pp. 627–638 (2011)
10. Felt, A.P., Wang, H.J., Moshchuk, A., Hanna, S., Chin, E.: Permission redelegation: attacks and defenses. In: Proceedings of the 20th USENIX Conference on Security, SEC 2011, p. 22. USENIX Association, Berkeley (2011)

11. Gibler, C., Crussell, J., Erickson, J., Chen, H.: AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. In: Katzenbeisser, S., Weippl, E., Camp, L.J., Volkamer, M., Reiter, M., Zhang, X. (eds.) TRUST 2012. LNCS, vol. 7344, pp. 291–307. Springer, Heidelberg (2012)
12. Grace, M., Zhou, Y., Wang, Z., Jiang, X.: Systematic Detection of Capability Leaks in Stock Android Smartphones. In: Proceedings of the 19th Network and Distributed System Security Symposium, NDSS (February 2012)
13. Grace, M.C., Zhou, W., Jiang, X., Sadeghi, A.R.: Unsafe exposure analysis of mobile in-app advertisements. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC 2012, pp. 101–112. ACM, New York (2012)
14. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, pp. 639–652. ACM, New York (2011)
15. Leontiadis, I., Efstratiou, C., Picone, M., Mascolo, C.: Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In: Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications, HotMobile 2012, pp. 2:1–2:6. ACM, New York (2012)
16. Nauman, M., Khan, S., Zhang, X.: Apex: extending android permission model and enforcement with user-defined runtime constraints. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, pp. 328–332. ACM, New York (2010)
17. Ongtang, M., McLaughlin, S.E., Enck, W., McDaniel, P.D.: Semantically Rich Application-Centric Security in Android. In: ACSAC, pp. 340–349. IEEE Computer Society (2009)
18. Pearce, P., Felt, A.P., Nunez, G., Wagner, D.: AdDroid: Privilege separation for applications and advertisers in Android. In: Proceedings of AsiaCCS (May 2012)
19. Shekhar, S., Dietz, M., Wallach, D.S.: AdSplit: separating smartphone advertising from applications. In: Proceedings of the 21st USENIX Conference on Security Symposium, Security 2012, p. 28. USENIX Association, Berkeley (2012)
20. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in android ad libraries. In: IEEE Mobile Security Technologies (MoST), San Francisco, CA
21. Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: IEEE Symposium on Security and Privacy, pp. 95–109 (2012)

Online Banking with NFC-Enabled Bank Card and NFC-Enabled Smartphone

Max Günther and Bernd Borchert

Department of Computer Science, University of Tübingen, Germany

Abstract. Banks want to use their genuine strong credential for online banking transaction authorization - the debit card. Customers nowadays are usually equipped with a Smartphone and prefer to not carry a card reader in addition. Methods where developed that use the Smartphone to authorize online banking transactions. These methods are vulnerable to Smartphone malware. We present NFC-TAN as a Smartphone method that combines the two requirements: Strong credential debit card and no additional device. We discuss to what extend this solution decreases vulnerability. Moreover, we consider usability, cost, and integration aspects of NFC-TAN.

Keywords: Online Banking, NFC, Smartphone, Smartcard, Signing.

1 Introduction

Modern online banking methods are secured against content-manipulation attacks with transaction-signing solutions: a separate device computes a signature depending on the details of a transaction. The necessary characteristics of a secure signature creation device have been worked out and explained [21,32] and corresponding online banking systems have been implemented (see Sect. 2.2) which can be considered secure against content-manipulation attacks.

However, academic approval of security is not the only success factor for online banking methods [25,32]. Other important factors are usability, system integration complexity, trust, bank-internal policies or business strategies, external regulations, and - maybe most importantly - costs and benefits. That is why

Fig. 1. Basic steps of the NFC-TAN method

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 66–81, 2013.

there are many systems with different security, usability, and cost trade-offs in use today.

In this paper, we present NFC-TAN as a method that uses the genuine bank-owned credential debit card (also known as bank card) to authorize PC-based online banking transactions in conjunction with a Smartphone - and nothing else. NFC-TAN works the following way [5]: the Smartphone reads the transaction via 2D code from the PC, shows the transaction data for confirmation on its display, contacts the debit card via NFC and receives a confirmation code from the card; the customer finally transfers this code to the PC (see Fig. 1 and Fig. 2).

We think this approach is well-founded given two assumptions: Banks want to have control over the client side credential and customers do not want to carry any additional devices for authorization purposes. Debit cards offer an array of advantages: They are convenient, well known, copy proof, and already integrated into a banks credential management system.

The enabling technology for this approach is NFC (Near Field Communication): NFC-Smartphones are available and debit cards will be soon NFC-enabled. The NFC-TAN method offers a trade-off between security, usability and cost that was not possible before. This paper contributes an analysis of the security situation and a discussion of the other success factors usability and costs.

2 State of the Art and Related Work

Apart from credential stealing attacks, the customers' insecure Internet clients have been identified as the main vulnerability of online banking systems. Sophisticated attacks such as man-in-the-browser attacks [12,30,32] were developed and already anticipated for example in [18]. These content-manipulation attacks can be only defeated if an online banking transaction is authorized by a signature depending on the transaction data and computed by a separate device that also displays the transaction data. If a nonce is used as additional input, also replay and pre-play attacks can be ruled out [4,28]. A signature in the online banking context may be symmetric or asymmetric and is termed TAN (transaction-authorization-number) or, to underline their dynamic and transaction dependent nature, dynamic-TAN or TAC (transaction-authorization-code [32]). In this paper, we assume traditional username/password authentication at login time as this is used in most online banking systems at least in central Europe - in fact, strong login authentication seems rather unnecessary given transaction-signing solutions.

2.1 Basic Steps of Transaction Signing Solutions

Most modern online banking solutions use two devices in tandem: the PC-browser for convenient input of transaction details and a separated device which displays those details and computes a signature for it. This approach requires two communication paths: the transaction details must be somehow transferred to the signature creation device and the computed signature must be somehow transferred to the PC (or server). The basic steps in most systems therefore are:

1. Enter transaction on the PC-browser (and send it to the server).
2. Transfer transaction details to the separated device.
3. Check (and confirm) the transaction on the display of the separated device, which computes a signature for the transaction.
4. Transfer the signature to the server (possibly via PC-browser).

2.2 Secure Signature Creation Device Solutions

In order to be considered immune against content-manipulation attacks, a signature creation device must fulfill some basic requirements. Various secure devices and corresponding online banking solutions have been proposed and implemented. From our understanding the basic properties of such devices are:

- A display that is guaranteed to show the transaction that will be signed.
- A crypto-module that is guaranteed to sign the displayed transaction.
- A signature can leave the device only if the user decides so.
- Secure storage of the credential (tamper-proof and challenge-response).

One approach to enforce the first three properties is to allow only very limited IO-capabilities (number pad + LCD display) that render malware infection virtually impossible. Especially to allow for more sophisticated interaction with the customers PC, the operating system of the signature creation device must be built securely from ground up and a button to trigger the signature computation may be necessary (if the device also has an out-channel for the signature other than the display). Laurie and Singer define a larger set of requirements to ensure the abovementioned basic properties and to add additional value (e.g. non-repudiation, updateable) [21]. The requirement to securely store a customers credential is mostly considered to be sufficiently satisfied by dedicated hardware like smartcards - laboratory-demonstrated attacks on such hardware are too expensive and therefore considered to be irrelevant for online banking systems [32]. Likewise, we consider vulnerabilities arising from implementation deficits like those reported in [3,13] to be out of the scope of this paper.

We briefly describe six characteristic external-device solutions in use today which all fulfill the abovementioned requirements for a secure signature creation device, further discussions of such and other solutions can be found in [32,19,26]. In all solutions, the transaction details are entered on the PC-browser and double-checked on a separate device that also computes a signature but they greatly vary in terms of usability resp. required customer interaction. We distinguish the solutions in terms of communication/user interaction and the used crypto-module:

ChipTAN [10] An offline smartcard reader (aka TAN-Generator) with photodiodes and display. This solution is of special interest for this paper, because our implementation is based on the same specifications, see Section 3.3. The so called TAN-Generator has a rudimentary optical channel consisting of five photodiodes that serially read the transaction data from a flickering code shown on the PC-display. Once the transaction is transmitted, the

TAN-Generator sends it to the plugged in debit card which in turn computes and returns a TAN as a function of the transaction and a secret. The TAN is then shown on the TAN-Generators display.

Manual Smartcard Reader [10] Manual variant of chipTAN without photodiodes. Communication with browser: number pad / display. Credential: secret stored on debit card.

Camera Token [8] An offline token with camera and display. Communication with browser: 2D code and camera / display. Credential: stored on device.

Offline Token [31] An offline token with a number pad and a display. Communication with browser: number pad and display. Credential: secret stored on the device.

ZTIC [32] A smartcard reader with USB connection and display. Communication with server via special protocol. Credential secret stored on smartcard.

USB-Token [24] Token with USB connection and display. Communication with server via special protocol. Credential stored on device.

2.3 Smartphone-Only Solutions

Solutions were proposed and implemented that use Smartphones instead of secure signature creation devices. For a customer the main advantage of those solutions is convenience: He does not have to carry an additional device but can instead use his Smartphone as a second display in order to check the transaction details before confirming them. The Smartphone solutions also provide communication channels for the transfer of transaction data to the Smartphone and for the transfer of the signature to the PC or server.

Photo-TAN. This class of solutions uses 2D codes to conveniently transfer the transaction data to the Smartphone via camera. A Smartphone app then displays the transaction data and computes a signature using a key stored on the Smartphone. The signature is then either sent to the server via mobile Internet or entered manually on the PC browser by the user. Photo-TAN was suggested by several researchers [6,11,17,27] and was recently implemented at some banks [1,7,9]. Some of these implementations work in a reversed fashion: The server encrypts the transaction and a TAN and the Smartphone decrypts the message and displays both components. Given the rise of mobile malware, Photo-TAN has severe vulnerabilities because none of the basic requirements for secure signature creation devices introduced above is fulfilled. Smartphone malware is possibly able to steal the customers signature key and send it to any recipient. Moreover it may be able to manipulate the display and the communication with the crypto-module and send signatures to any recipient, see Sect. 4.1.

mobileTAN. Another well-known solution is mTAN. The customer enters the transaction details in the PC browser and sends them to the server. The server sends the transaction and a TAN to the customers mobile phone via text/SMS. In order to confirm the transaction, the customer enters the TAN in the PC browser

(or cancels the operation if the transaction details have been manipulated). Various attack vectors against mTAN have been exploited [29], see Sect. 4.1.

2.4 Trusted-Smartphone Plus Smartcard

Smartphone based solutions that assume the existence of a special execution mode sufficiently shielded against mobile malware can of course meet the above-mentioned requirements for a secure signature creation device.

Alpár et al. [2] present one such solution using an NFC smartcard as an external crypto module and envision a user triggered *trusted mode* that enables the user to verify the transaction details on a then-trustworthy Smartphone display; Ortiz-Yepes [22] proposed a similar approach with a NFC smartcard as external crypto module. In addition, both works suggested a protocol based on the industry standard EMV-CAP (Chip Authentication Program) that avoids some of the flaws (e.g. not including the transaction data as signature challenge) found by Drimer et al. in other implementations [13]. The results are similar to the HHD protocol which was developed without those flaws from the outset by the German banking industry and that is used by our solution, see Sect. 3.3.

The main difference to our work is the assumption about having a Smartphone with a *trusted mode*. This distinction greatly influences the threat model and consequently mobile malware attacks are not covered in [2] while these are a major part of our security analysis. Another difference is that we focus on two-channel PC-based online banking, while Alpár et al. discuss the natural scenario given a trusted Smartphone: single-channel mobile banking. Although trusted execution technologies for Smartphones are being developed, to our knowledge currently no commercially available device supports the full functionality.

3 NFC-TAN Method

3.1 Motivation

The NFC-TAN method presented below allows for online banking transaction authorization with the customers debit card as credential and his Smartphone as communication device [5,16,23]. We belief this approach is justified given two assumptions:

- Banks prefer the debit card over other client side credentials. The credential debit card is already integrated into a banks credential management system: reliable processes for development, deployment and revocation are in use in any bank. A further benefit is the already implemented industry standard EMV that can be used for transaction authorization, see Sect. 3.3.
- Customers do not want an additional device. An additional device is always a burden - one thing more that costs money and can get stolen, lost, or broke. In contrast, customers have their Smartphone and debit card with them anyway. Apart from these physical aspects, the credential debit card is well-known and convenient because the bank is responsible for initialization, deployment and replacement.

3.2 Description

The steps in the NFC-TAN method are quite simple and familiar to a customer because they resemble the four steps of the basic procedure described in Sect. 2.1:

1. Log in on the PC browser, enter transaction and submit it unconfirmed to the bank server.
2. Scan the 2D code shown on the PC screen with the Smartphone.
3. Double-check the transaction on the Smartphone display and confirm with the debit card.
4. Transfer the TAN to the PC browser and submit it to the bank server.

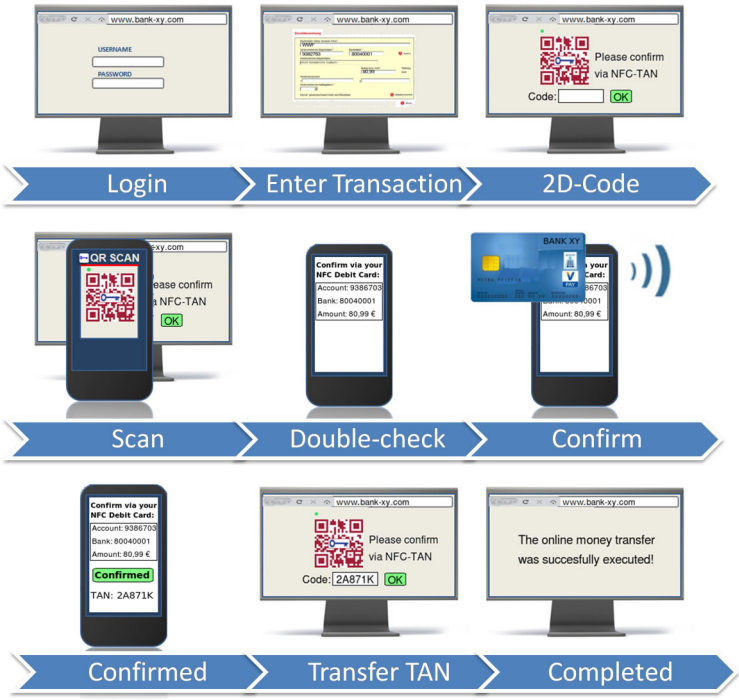


Fig. 2. NFC-TAN procedure from a customer's perspective

Background Interaction. Under the hood, the following interaction and computation is done:

- After receiving the transaction, the bank server generates a 2D code containing the transaction details and a nonce. This 2D code is included in the server response.

- The Smartphone app builds a challenge from the transaction details and the nonce and sends it to the debit card. The debit card calculates the response as a function of the challenge and a secret key and returns it to the Smartphone app. The Smartphone app selects the TAN from this response.
- The server can compute the correct TAN since it knows the transaction details, nonce, and secret key and therefore is able to check the TAN received.

Process Requirement. We require that the server has only one open session and only one open transaction at a time for one account. Our security discussion explains why this rather easy-to-implement requirement is important.

Familiarity. For many customers the NFC-TAN method may be familiar because it is similar to the abovementioned chipTAN method, the differences are: a 2D code is shown instead of a flickering code, the device is a Smartphone not a special purpose device, and debit card and device communicate contactless. NFC-TAN can be seen an extension of Photo-TAN for which the credential is stored and used on the debit card instead of the Smartphone.

Performance. Modern Smartphones are able to scan 2D-code within a second and smartcards are able to compute the response within less than a second.

3.3 Debit Card, EMV and HHD

Debit cards, or more general bank cards, are bank issued smartcards that host an array of different applications. NFC-TAN utilizes the debit cards ability to compute a signature as a function of a given challenge and a secret stored on the card. The inter-operation of terminal and card for this functionality is specified by EMV [14], a generic industry standard based on ISO7816 (contact) resp. ISO14443 (contactless). On the basis of EMV, card issuers developed standards for applications like ATM, POS, and online banking - with MasterCards CAP being a prominent example. The German banking industry committee *Central Credit Committee*, developed the standard HHD (hand held device) [33] which is used for online banking transactions. As the EMV protocol has no notion of transaction details like beneficiary's account number and amount, HHD specifies how reader devices aggregate those details into a challenge for the smartcard. The abovementioned chipTAN solution is one implementation of HHD, where the transaction is transmitted to the TAN-Generator, which aggregates the transaction and hands it over to the plugged in debit card.

We based our implementation of NFC-TAN also on the HHD specification. Consequently, the Smartphone app of NFC-TAN contains a software implementation of the TAN-Generator. NFC-TAN is therefore currently compatible with the chipTAN solution but is adaptable to many solutions using EMV based or other specifications since the only strong requirement towards the smartcard is the ability to compute a response for a challenge using a stored secret.

4 Security

Attacks against online banking services can be classified into three main categories: software attacks, physical attacks, and social attacks [32]. Software attacks are the most common and the most threatening attack vectors, because they can be launched against a large number of customers with comparatively small effort and are harder to trace back than social or physical attacks [32].

4.1 Software Attacks

Credential-Stealing and Credential-Abusing Attacks. The credentials of the NFC-TAN method are: password for login authentication and the signature key for transaction authorization. The password could be stolen by key loggers, phishers or social engineers. In addition, mobile malware can steal the password on the Smartphone from balance-checking bank apps which are usually protected by the same password. With NFC-TAN, the signature key is stored on the debit card and therefore out of reach of both malware and tampering.

However, malware on the Smartphone may not have to steal the signature key at all: it may be enough to be able to send a challenge to the debit card and obtain a signature. We describe how this attack is prevented in the NFC-TAN method by two countermeasures the server can implement, assuming state of the art server side session handling: (i) Only one open login-session for one customer is allowed at a given time, (ii) the server-challenge includes a nonce. Because the customer only brings the card to the Smartphone in order to confirm a transaction (and therefore has an open session) an attacker cannot initiate a transaction on its own (he would need to have an open session). The nonce prevents pre-play attacks, because a challenge obtained earlier is invalidated when the associated session is closed (at the latest), and the customer cannot open a session if the attackers is still open.

In a variant of this attack - with a collaborating malware on the PC that hijacked the customers session - Smartphone malware could try to forward an additional, malicious challenge to the debit card while the user is holding his card close to the Smartphone. This attack is prevented by allowing only one transaction at a given time and introducing a pause of some seconds between transactions.

We can conclude that the use of the debit card as a credential protects the NFC-TAN method against credential-stealing attacks. Credential-abusing attacks by Smartphone malware alone are much harder to conduct because the debit card is not permanently connected to the Smartphone. We assume a responsible and attentive customer and the server side countermeasures described above, namely (1) allowing only one session per user, and (2) one open transaction per session at a given time, (3) using a nonce, and (4) enforcing a delay of a few seconds between the completion of one transaction and the start of another one.

Credential-Stealing and Credential-Abusing Attacks Are More Dangerous without Debit Card. Photo-TAN is an example of a method in which both credentials password and signature key can be stolen by malware, actually by malware on the Smartphone alone. The signature key is stored on the Smartphone and any attempt to secure it (e.g. key encryption via a pin or another key) is ultimately vain with respect to powerful malware, because malware could just imitate the steps executed preceding the signing algorithm, and finally will find the key. Even a pin will not prevent this because the pin is first tapped and then used in the imitating process. Malware doing this has to be sophisticated and it is not clear what the success expectation of such an attack is in practice. Nevertheless, if such an attack is successful the consequences are fatal: Malware could send the credentials to a remote attacker who can execute any banking transaction at any time! One way to store a key copy-proof on the Smartphone is a Secure Element that never exposes the key but only has a challenge-response interface. For example, Photo-TAN could be extended with such a Secure Element provided by the bank and integrated physically into the Smartphone [27]. Malware then cannot steal the signature key but abuse it by communicating with the Secure Element and conduct an attack in the following way: First the password is tapped when the customer enters it on the Smartphone to check his account balance. Afterwards malware can open an online banking session at any time and can abuse the Secure Element to compute a signature for an arbitrary transaction without knowing the key. Restricting access to the SE for all but some signed apps may hinder this attack considerably.

This example demonstrates that using a Secure Element with permanent contact to the Smartphone can lead to vulnerabilities not present in the proposed NFC-TAN solution. It also shows that the variant of the NFC-TAN method with an NFC chip stuck on or plugged into the Smartphone is a severe degradation in terms of security though its usability is desirable. Another security issue is the distribution of the credential to the users Smartphone. A practical disadvantage for a bank is the administration effort of yet another credential.

The mobileTAN solution uses the phone number as credential. Regarding Smartphone malware that approach suffers from the fact that text/SMS messages are not considered deserving special protection on Smartphones and are generally accessible by apps, once the user granted a permission to do so – quite a common permission to ask for by all sorts of apps offered. One attack that exploits this situation is described in [29]. Another possible attack is executed by Smartphone malware alone: after stealing the account password mobile malware logs into the customers bank account and initiates a malicious transaction, intercepts the text message on the mobile phone, picks the tan and confirms the transaction. This attack can be conducted anytime the mobile is online and completely without notice to the user as no user interaction is required.

Content-Manipulation Attacks. A man-in-the-browser can manipulate the communication between bank server and PC-browser showing the transaction the customer entered, but communicating a forged one - a second display sufficiently separated from the PC defeats this attack. If a Smartphone with Internet

access is used as second display, a content-manipulation attack can only be conducted by collaborating malware on PC and Smartphone, because both displays have to be manipulated:

PC malware cannot manipulate the transaction because the Smartphone will display it again.

Mobile malware cannot manipulate the transaction because the correct transaction was displayed on the PC and is already known to the server.

Only collaborating malware on PC and Smartphone can perform a distributed men-in-the-middle attack by manipulating both displays. This remains a vulnerability of the NFC-TAN solution and is another example of the fact that two insecure systems do not result in a secure system.

Both malware instances need to know the original transaction (to be able to display it) and the forged transaction (in order to send it to the server resp. the debit card). They may communicate the transactions via 2D code, Wi-Fi, or an Internet server. The attack preparation includes finding and infiltrating both devices of one customer with malware. The actual independence of PC and Smartphone therefore is an important measure for the estimation about how complicated and how probable such attacks are. For example, sharing a Wi-Fi network or establishing a cable connection to synchronize calendars may increase the likelihood of a double infection.

Double infection is only a necessary prerequisite for an attack on NFC-TAN the attack itself still has to be conducted as a real-time manipulation attack. The difficulty of the attack step against the Smartphone app is probably comparable to the key theft from a Photo-TAN Smartphone app. The coordination of two distributed malware instances on PC and Smartphone alone is neither a real obstacle to attackers. Nevertheless, the attack has to involve the customers debit card and therefore the customer's interaction. Firstly, this greatly reduces the opportunities of an attack from 'nearly always' to about 'once every few days'. Secondly, the attack has to be executed in the moment the user wants to confirm a transaction, that is, within a few seconds. Therefore this attack is more difficult than the double infection attack on mobileTAN reported in [29].

4.2 Physical Attacks

Physical attacks are often considered to be less important in the online banking context because they cannot be launched against a large number of customers. In general we agree with that estimation, therefore we only discuss one aspect particularly interesting for NFC-TAN:

Wireless Attack. Enabling access to the TAN computation functionality of the debit card via NFC introduces new vulnerabilities and customers' concern because an attacker could contact the debit card unnoticed by the owner.

However, the attacker still would have to know the account number and account password to complete his attack. Depending on the implementation neither the account number nor the card number can be read via NFC. Even if an attacker can associate a card with an account number and has stolen the password, a real-time attack is necessary because the server-challenge includes a nonce that prevents replay and pre-play attacks: the attacker would have to have simultaneously: an open bank session, a valid challenge and a NFC-connection within a range of 25cm [20]. Summarizing, the wireless attack is a negligible threat to NFC-TAN.

4.3 Social Attacks

For NFC-TAN customers, one reasonable briefing is important: "Only introduce your debit card to the Smartphone if you want to confirm an online banking transaction you initiated!" Then the abovementioned credential-abusing attack can be defeated because the customers open online banking session on the PC prevents Smartphone malware from opening a parallel covert one. The NFC-TAN user has a chance to prevent such an attack without an in-depth notion of the underlying security.

The user could still be tricked into signing a bogus transaction (e.g. for "service test reasons") with the NFC-TAN solution like with almost all other solutions including secure signature creation devices. This kind of attacks is easily prevented by reasonable instructions and customers attentiveness but very hard to prevent without.

4.4 Protocol / Cryptography

Since our implementation does not introduce a new protocol but reuses the specification for transaction authorization with debit cards HHD, we did not perform a security analysis of the protocol or the involved algorithms. The Smartphone does neither store a credential nor perform critical computation exactly like the TAN-generator in the HHD implementation. Since the secret on the debit card is symmetric our implementation of NFC-TAN is also symmetric. A public key implementation of NFC-TAN would be possible with suitable debit cards.

5 Usability

One obvious usability disadvantage as compared to other Smartphone solutions is the additional step of holding the debit card close to the Smartphone. This is balanced by the following advantages.

No Additional Device. This is especially important in a mobile scenario (e.g. in an Internet cafe) because the user does not want to carry an additional device he can lose or forget.

Convenient Communication and Display. The transaction is transferred from the PC to the Smartphone via a 2D code resp. a 2D code scanner on the Smartphone. This is usually completed in less than a second and familiar to users, as 2D codes are quite widespread. The user checks the transaction on the Smartphone display where it is possible to show a transaction completely and comfortably. In fact, the customer may check the transaction only on the Smartphone, because a manipulation by malware on the PC alone would not be a threat.

Offline Availability. The NFC-TAN method does not require an active mobile Inter-net connection on the Smartphone and therefore can be used abroad or in areas with bad network connection. This is a usability advantage compared to other solutions like mobileTAN.

Exchangeable Smartphone. As the Smartphone is a mere communication device, it is exchangeable, for example by a newly bought one or a friends one. Likewise, the credential can easily be delegated since it is stored on the debit card - this may be of benefit in a company account scenario. However, customers may prefer if Smartphone and debit card are paired, e.g. because they may consider this as an additional prevention of the abovementioned wireless relay attack.

Increasing Penetration of NFC-enabled Smartphones. In 2012 over 26% of all Smartphones sold in Europe were NFC-enabled [15], global numbers should be similar. It is to be expected that a significant part of the customers will have NFC-Smartphones in the near future.

6 Complexity: Implementation, Integration and Administration

Minimal Credential Administration Changes on the Server. Banks already maintain debit cards as credentials. The NFC-TAN solution can be integrated into this system without changes to server-side credential management and distribution. Especially, no further credential is introduced and has to be administrated in parallel on the server side. If a bank already uses chipTAN/HHD for online banking, our NFC-TAN implementation can be integrated with minimal server-side efforts: only the flicker-code has to be replaced by a 2D code. This is only a user interface change, whereas business logic on the background servers remains unchanged. Consequently, both methods can be offered simultaneously to the customer.

Smartphone Application Development. In order to support NFC-TAN, a bank's Smartphone app needs to be extended by the following functionality: 2D code scanner, software implementation of the smartcard reader (TAN-Generator), and NFC communication. Although this is not trivial, there is no implementation risk because the technologies are well-known.

NFC-enabled Debit Cards. NFC-TAN needs only minimal software changes on the EMV/HHD debit card - probably only the activation of NFC is necessary (*Dual Interface*). In Germany, debit cards may be NFC-enabled in the medium term for this functionality - currently an electronic purse application on the debit card of some banks can be accessed via NFC.

7 Costs

As online banking providers are profit-oriented companies, their primary attention clearly lays on cost. Other factors are important mainly because they affect costs and revenue. This means that investments for better security measures including implementation, integration and administration have to pay off at the end of the day.

Non-recurring Expenses. The implementation of the NFC-TAN method requires development and integration costs for server and Smartphone software, see Section 6. The distribution of the NFC-enabled cards will most likely occur within the regular exchange of bank cards. Moreover, no additional software needs to be developed and installed on the debit card, see Section 6. In other words, the requirement of NFC-enabled debit cards adds no expense - neither for banks nor customers.

Recurring Expenses. The NFC-TAN method adds no direct recurring costs for a bank like device costs or per-transaction costs. Nevertheless, there will be special indirect costs like customer support and Smartphone app maintenance and indirect costs like server administration and credential management. These costs are comparable to those of other methods.

Expenses for the Customer. The customer - if he wants to use NFC-TAN - has to buy an NFC-Smartphone. This may be interpreted in the way that banks move parts of their costs to their customer.

8 Variants and Extensions

8.1 NFC-TAN Mobile Banking (Single-Channel)

Customers may demand the following single-channel mobile banking variant of NFC-TAN: the user logs into his bank account on the Smartphone with an app or on the mobile browser and enters a transaction, and confirms it by holding the NFC debit card close to the Smartphone. This raises security issues comparable to those already discussed in Sect. 4.1. The main problem is that mobile malware alone can conduct a transaction manipulation attack. NFC-TAN mobile banking should therefore be protected by further server-side mechanisms like beneficiary white-lists, transaction limits, etc., or a *trusted mode* like suggested in [2].

8.2 OCR Instead of 2D Code

In the NFC-TAN method the transaction is transferred from the PC to the Smartphone via a 2D code. Instead, the Smartphone app could take a snapshot of the filled transaction form and recognize the transaction via OCR. This What-You-See-Is-What-You-Sign variant increases usability because the customer does not have to check the transaction on the Smartphone. Moreover, this increases security against a man-in-the-PC-browser which may speculate that user does not double-check the transaction on the Smartphone and therefore manipulate the transaction. This OCR extension of NFC-TAN may also allow a convenient and secure way to allow for collective transfers - no double-checking on the Smartphone is necessary.

8.3 Online NFC-TAN

In the NFC-TAN method, the TAN is manually transferred to the PC and then sent to the server. As a variant, the Smartphone could also send the TAN to the server via mobile Internet. This increases usability because the customer does not have to type the TAN. However, the Smartphone would have to have mobile Internet connection. Another disadvantage is increased implementation complexity on the server-side because the TAN receiving script has to be integrated.

8.4 Secure Displays

The main vulnerability of the NFC-TAN is a result from the fact that common Smartphones do not have a secure display. However, NFC-TAN could be combined with some secure display approaches. One solution would be a smartcard having a display [27] and additional NFC-functionality. Another solution could be a Smartphone with a secure display (ARM TrustZone® / G&D MobiCore®) and still use the debit card as a bank owned external credential in case there are reasons for not storing the credential on the Smartphone, like complexity, integration and administration overhead for a new credential.

9 Conclusion

We present NFC-TAN as a solution for online banking transaction authorization that uses the customers debit card as credential and his Smartphone for communication. The NFC-TAN solution is more secure than pure Smartphone solutions without sacrificing much usability. It is less secure than the secure signature creation device solutions but more convenient. Integration complexity and costs on the bank side are comparatively low since no additional credential needs to be managed. Summarizing, NFC-TAN is an online banking method with a trade-off between security, usability, and costs which was not possible until the recently beginning ubiquity of NFC-enabled devices.

Acknowledgement. We would like to thank Klaus Reinhardt and Matthias Sattler for many interesting discussions and the reviewers for their valuable hints and remarks.

References

1. 1822direkt GmbH: Modernes Online-Banking mit QR-TAN (2012), https://www.1822direkt.com/service/zugangsmoeglichkeiten/
2. Alpár, G., Batina, L., Verdult, R.: Using NFC phones for proving credentials. In: Schmitt, J.B. (ed.) MMB & DFT 2012. LNCS, vol. 7201, pp. 317–330. Springer, Heidelberg (2012)
3. Blom, A., de Koning Gans, G., Poll, E., de Ruiter, J., Verdult, R.: Designed to Fail: A USB-Connected Reader for Online Banking. In: Jøsang, A., Carlsson, B. (eds.) NordSec 2012. LNCS, vol. 7617, pp. 1–16. Springer, Heidelberg (2012)
4. Bond, M., Choudary, O., Murdoch, S.J., Skorobogatov, S., Anderson, R.: Chip and Skim: cloning EMV cards with the pre-play attack (2012)
5. Borchert, B.: Sichere Verschlüsselung für Online Accounts durch ein Gerät mit Kamera, Display und Nahfunk als Mittler zwischen Rechner und Geheimnis. German patent DE102009040009B4 (2009)
6. Borchert, B., Reinhardt, K.: Vorrichtung und Verfahren zur abhör- und manipulationssicheren Verschlüsselung für Online-Accounts. German patent DE2007052734B4 (2007)
7. Commerzbank AG: photoTAN, http://www.commerzbanking.de/
8. Cronto Limited: CrontoSign Device (2012), http://www.cronto.com/crontosign-transaction-authentication-device.htm
9. Cronto Limited: CrontoSign Mobile App (2012), http://www.cronto.com/crontosign-transaction-authentication-mobile.htm
10. Die Deutsche Kreditwirtschaft: chipTAN, http://www.die-deutsche-kreditwirtschaft.de/dk/zahlungsverkehr/electronic-banking/chiptan.html
11. Dodson, B., Sengupta, D., Boneh, D., Lam, M.S.: Secure, consumer-friendly web authentication and payments with a phone. In: Gris, M., Yang, G. (eds.) MobiCASE 2010. LNICST, vol. 76, pp. 17–38. Springer, Heidelberg (2012)
12. Dossogne, J., Markowitch, O.: Online banking and man in the browser attacks, survey of the belgian situation. In: Goseling, J., Weber, J.H. (eds.) Proceedings of the 31st Symposium on Information Theory in the Benelux, WICSITB 2010, pp. 19–26 (2010)
13. Drimer, S., Murdoch, S.J., Anderson, R.: Optimised to Fail: Card Readers for Online Banking. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 184–200. Springer, Heidelberg (2009)
14. EMVCo LLC: EMV Integrated Circuit Card Specifications for Payment Systems, Book 1-4 (2008), http://emvco.com/
15. GfK SE: Smartphones Bring Fresh Boost (2013), http://www.gfk.com/news-and-events/press-room/press-releases/Pages/Smartphones-bring-fresh-boost.aspx
16. Günther, M.: Sicheres Online Banking via Smartphone mit Nahfunk (NFC). Master's thesis, Universität Tübingen (2011)
17. Handelsblatt: Online-Banking wird sicherer (2008), http://www.handelsblatt.com/technologie/it-tk/it-internet/internet-online-banking-wird-sicherer/3064982.html

18. Hiltgen, A., Kramp, T., Weigold, T.: Secure Internet banking authentication. IEEE Security Privacy 4(2), 21–29 (2006)
19. Hisamatsu, A., Pishva, D., Nishantha, G.G.D.: Online banking and modern approaches toward its enhanced security. In: The 12th International Conference on Advanced Communication Technology (ICACT), vol. 2, pp. 1459–1463 (2010)
20. Kirschenbaum, I., Wool, A.: How to Build a Low-Cost, Extended-Range RFID Skimmer. In: 15th USENIX Security Symposium, pp. 43–57 (2006)
21. Laurie, B., Singer, A.: Choose the red pill and the blue pill: a position paper. In: Proceedings of the 2008 Workshop on New Security Paradigms, pp. 127–133. ACM, New York (2008)
22. Ortiz-Yepes, D.A.: Enhancing Authentication in eBanking with NFC-Enabled Mobile Phones. In: ERCIM News. No. 76, European Research Consortium for Informatics and Mathematics (2009)
23. Sattler, M.: Einbinden der neuen NFC-Debitkarte in das Fotohandy. Master's thesis, Universität Tübingen (2012)
24. Seal One AG: Seal One USB (2011), `http://www.seal-one.com/products-list.en-UK.html`
25. Shah, M.H., Braganza, A., Morabito, V.: A survey of critical success factors in e-Banking: an organisational perspective. European Journal of Information Systems 16(4), 511–524 (2007)
26. Singh Brar, T.P., Sharma, D., Singh Khurmi, S.: Vulnerabilities in e-banking: A study of various security aspects in e-banking. International Journal of Computing & Business Research (2012)
27. Starnberger, G., Froihofer, L., Goeschka, K.M.: QR-TAN: Secure Mobile Transaction Authentication. In: International Conference on Availability, Reliability and Security, ARES 2009, pp. 578–583 (2009)
28. Syverson, P.: A Taxonomy of Replay Attacks. In: Proceedings of the Computer Security Foundations Workshop VII, CSFW 7, pp. 187–191 (1994)
29. The H Security: Millions stolen with mTAN fraud (2012), `http://www.h-online.com/security/news/item/Millions-stolen-with-mTAN-fraud-1763923.html`
30. Utakrit, N.: Review of Browser Extensions, a Man-in-the-Browser Phishing Techniques Targeting Bank Customers. In: Australian Information Security Management Conference (2009)
31. Vasco Inc.: DIGIPASS 260 (2013), `http://www.vasco.com/products`
32. Weigold, T., Hiltgen, A.: Secure confirmation of sensitive transaction data in modern Internet banking services. In: World Congress on Internet Security (WorldCIS), pp. 125–132 (2011)
33. Zentraler Kreditausschuss: Schnittstellenspezifikation für die ZKA-Chipkarte - HandHeldDevice, HHD (2010)

A Defensive Virtual Machine Layer to Counteract Fault Attacks on Java Cards

Michael Lackner, Reinhard Berlach, Wolfgang Raschke,
Reinhold Weiss, and Christian Steger

Institute for Technical Informatics,
Graz University of Technology, Graz, Austria
{michael.lackner,reinhard.berlach,wolfgang.raschke,
rweiss,steger}@tugraz.at

Abstract. The objective of Java Cards is to protect security-critical code and data against a hostile environment. Adversaries perform fault attacks on these cards to change the control and data flow of the Java Card Virtual Machine. These attacks confuse the Java type system, jump to forbidden code or remove run-time security checks. This work introduces a novel security layer for a defensive Java Card Virtual Machine to counteract fault attacks. The advantages of this layer from the security and design perspectives of the virtual machine are demonstrated. In a case study, we demonstrate three implementations of the abstraction layer running on a Java Card prototype. Two implementations use software checks that are optimized for either memory consumption or execution speed. The third implementation accelerates the run-time verification process by using the dedicated hardware protection units of the Java Card.

Keywords: Java Card, Defensive Virtual Machine, Countermeasure, Fault Attack.

1 Introduction

A Java Card enables Java applets to run on a smart card. The primary purpose of using a Java Card is the write-once, run-everywhere approach and the ability of post-issuance installation of applets [21]. These cards are used in a wide range of applications (e.g., digital wallets and transport tickets) to store security-critical code, data and cryptographic keys. Currently, these cards are still very resource-constrained devices that include an 8- or 16-bit processor, 4kB of volatile memory and 128kB of non-volatile memory. To make a Java Card Virtual Machine run on such a constrained device, a subset of Java is used [19]. Furthermore, special Java Card security concepts, such as the Java Card firewall [18] and a verification process for every applet [15], were added. The Java Card firewall is a run-time security feature that protects an applet against illegal access from other applets. For every access to a field or method of an object, this check is performed. Unfortunately, the firewall security mechanism can be circumvented by applets

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 82–97, 2013.

that do not comply with the Java Card specification. Such applets are called malicious applets.

To counteract malicious applets, a bytecode verification process is performed. This verification is performed either on-card or off-card for every applet [15]. Note that this bytecode verification is a static process and not performed during applet execution. The reasons for this static approach are the high resource needs of the verification process and the hardware constraints of the Java Card. This behavior is now abused by adversaries. They upload a valid applet onto the card and perform a fault attack (FA) during applet execution. Adversaries are now able to create a malicious applet out of a valid one [5].

A favorite time for performing a FA is during the fetching process. At this time, the virtual machine (VM) reads the next Java bytecode values from the memory. An adversary that performs an FA at this time can change the readout values. The VM then decodes the malicious bytecodes and executes them, which leads to a change in the control and data flow of the applet. A valid applet is mutated by such an FA to a malicious applet [5,17,11] and gains unauthorized access to secret code and data [16,2].

To counteract an FA, a VM must perform run-time security checks to determine if the bytecode behaves correctly. In the literature, different countermeasures, such as control-flow checks [23], double checks [4], integrity checks [8] and method encryption [20], have been proposed. Barbu [3] proposed a dynamic attack countermeasure in which the VM executes either standard bytecodes or bytecodes with additional security checks.

All these works do not concentrate on the question of how these security mechanisms can be smoothly integrated into a Java Card VM. For this integration, we propose adding an additional security layer into the VM. This layer abstracts the access to internal VM resources and performs run-time security checks to counteract FAs. The primary contributions of this paper are the following:

- Introduction of a novel defensive VM (D-VM) layer to counteract FAs during run-time. Access to security-critical resources of the VM, such as the operand stack (OS), local variables (LV) and bytecode area (BA), is handled using this layer.
- Usage of the D-VM layer as a dynamic countermeasure. Based on the actual security level of the card, different implementations of the D-VM layer are used. For a low-security level, the D-VM implementation uses fewer checks than for a high-security level. The security level depends on the credibility of the currently executed applet and run-time information received by hardware or software modules.
- A case study of a defensive VM using three different D-VM layer implementations. The API of the D-VM layer is used by the Java Card VM to perform run-time checks on the currently executing bytecode.
- The defensive VMs are executed on a smart card prototype with specific HW security features to speed up the run-time verification process. The resulting run-time and main memory consumption of all implemented D-VM layers are presented.

Section 2 provides an overview of attacks on Java Cards and the current countermeasures against them. Section 3 describes the novel D-VM layer presented in this work and its integration into the Java Card design. Furthermore, the method by which the D-VM layer enables the concept of dynamic countermeasures is presented. Section 4 presents implementation details regarding how the three D-VM implementations are inserted into the smart card prototype. Section 5 analyzes the additional costs for the D-VM implementations based on the execution and main memory overhead. Finally, the conclusions and future work are discussed in Section 6.

2 Related Work

In this section, the basics of the Java Card VM and work related to FA on Java Cards are presented. Then, an analysis of work regarding methods of counteracting FAs and securing the VM are presented. Finally, an FA example is presented to demonstrate the danger posed by such run-time attacks for the security of Java Cards.

2.1 Java Card Virtual Machine

A Java Card VM is software that is executed on a microprocessor. The VM itself can be considered a virtual computer that executes Java applets stored in the data area of the physical microprocessor. To be able to execute Java applets, the VM uses internal data structures, such as the OS or the LV, to store interim results of logical and combinatorial operations. All of these internal data structures are general objects for adversaries that attack the Java Card [4,20,24].

For every method invocation performed by the VM, a new Java frame [19] is created. This frame is pushed to the Java stack and removed from it when the method returns. In most VM implementations, this frame internally consists of three primary parts. These parts have static sizes during the execution of a method. The first frame part is the OS on which most Java operations are performed. The OS is the source and destination for most of the Java bytecodes. The second part is the LV memory region. The LV are used in the same manner as the registers on a standard CPU. The third part is the frame data, which holds all additional information needed by the VM and Java Card Runtime Environment (JCRE) [18]. This additional information includes, for example, return addresses and pointers to internal VM-related data structures.

2.2 Attacks on Java Cards

Loading an applet that does not conform to the specification defined in [19] onto a Java Card is a well-known problem called a logical attack (LA). After an LA, different applets on the card are no longer protected by the so-called Java

sandbox model. Through this sandbox, an applet is protected from illegal write and read operations of other applets. To perform an LA, an adversary must know the secret key to install applets. This key is known for development cards, but it is highly protected for industrial cards and only known by authorized companies and authorities. In conclusion, LAs are no longer security threats for current Java Cards.

Side-channel analyses are used to gather information about the currently executing method or instructions by measuring how the card changes environment parameters (e.g., power consumption and electromagnetic emission) during run-time. Integrated circuits influence the environment around them but can also be influenced by the environment. This influence is abused by an FA to change the normal control and data flow of the integrated circuit. Such FAs include glitch attacks on the power supply and laser attacks on the cards [2,24]. By performing side-channel analyses and FAs in combination, it is possible to break cryptographic algorithms to receive secret data or keys [16].

In 2010, a new group of attacks called combined attacks (CA) was introduced. These CAs combine LAs and FAs to enable the execution of ill-formed code during run-time [5]. An example of a CA is the removal of the *checkcast* bytecode to cause type confusion during run-time. Then, an adversary is able to break the Java sandbox model and obtain access to secret data and code stored on the card [5,17]. In this work work, we concentrate on countering FAs during the execution of an applet using our D-VM layer.

2.3 Countermeasures against Java Card Attacks

Since approximately 2010, an increasing number of researchers have started concentrating on the question of what tasks must be performed to make a VM more robust against FAs and CAs. Several authors [22,8] suggest adding an additional security component to the Java Card applet. In this component, they store checksums calculated over basic blocks of bytecodes. These checksums are calculated off-card in a static process and added to a new component of the applet. During run-time, the checksum of executed bytecodes is calculated using software and compared with the stored checksums. If these checksums are not the same, a security exception is thrown.

Another FA countermeasure is the use of control-flow graph information [23]. To enable this approach, a control-flow graph over basic blocks is calculated off-card and stored in an additional applet component. During run-time, the current control-flow graph is calculated and compared with the stored control graph.

In [20], the authors propose storing a countermeasure flag in a new applet component to indicate whether the method is encrypted. They perform this encryption using a secret key and the Java program counter for the bytecode of every method. Through this encryption, they are able to counteract attacks that change the control-flow of an applet to execute illegal code or data.

Another countermeasure against FAs that target the data stored on the OS is presented in [4]. In this work, integrity checks are performed when data are pushed or popped onto the OS. Through this approach, the OS is protected against FAs that corrupt the OS data.

Another run-time check against FAs is proposed in [10,14], in which they create separate OSes for each of the two data types, *integralValue* and *reference*. With this approach of splitting the OS, it is possible to counteract type-confusion attacks. A drawback is that in both works, the applet must be preprocessed.

In [3], the authors propose a dynamic countermeasure to counteract FAs. Bytecodes are implemented in different versions inside the VM, a standard version and an advanced version that performs additional security checks. The VM is now able to switch during run-time from the standard to the advanced version. By using unused Java bytecodes, an applet programmer can explicitly call the advanced bytecode versions.

The drawbacks of current FA countermeasures are that most of them add an additional security component to the applet or rely on preprocessing of the applet. This has different drawbacks, such as increased applet size or compatibility problems for VMs that do not support these new applet components. In this work, we propose a D-VM layer that performs checks on the currently executing bytecode. These checks are performed based on a run-time policy and do not require an off-card preprocessing step or an additional applet component.

2.4 EMAN4 Attack: Jump Outside the Bytecode Area

In 2011, the run-time attack EMAN4 was found [6]. In this work a laser was used to manipulate the read out values from the EEPROM to 0x00. By this laser attack an adversary is able to change the Java bytecode of post-issuance installed applets during their execution.

The target time of the attack is when the VM fetches the operands of the *goto_w* bytecode from the EEPROM. Generally the *goto_w* bytecode is used to perform a jump operation inside a method. The *goto_w* bytecode consists of the operand byte 0xa8 and two offset bytes for the branch destination [19]. This branch offset is added to the actual Java program counter to determine the next executing bytecode. An adversary which changes this offset is able to manipulate the control flow of the applet.

With the help of the EMAN4 attack it is possible to jump with the Java program counter outside the applet bytecode area (BA), as illustrated in Figure 1. This is done by changing the offset parameters of the *goto_w* bytecode from 0xFF20 to 0x0020 during the fetch process of the VM. The jump destination address of the EMAN4 attack is a data array outside the bytecode area. This data array was previously filled with adversary defined data. After the laser attack the VM executes the values of the data array. This execution of adversary definable data leads to considerably more critical security problems, such as memory dumps [7]. In this work we counteract the EMAN4 attack by our control flow policy. This policy only allows to fetch bytecodes which are inside the bytecode area.

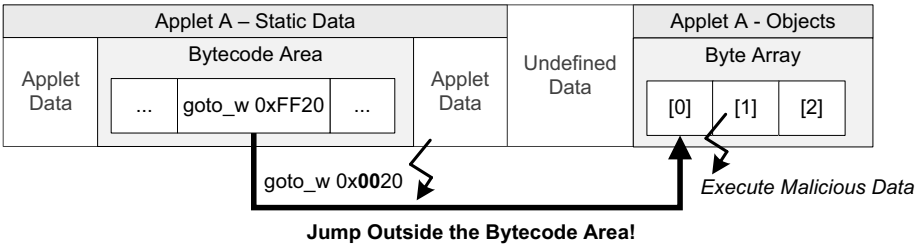


Fig. 1. The EMAN4 run-time attack changes the jump address 0xFF20 to 0x0020, which leads to the security threat of executing bytecode outside the defined BA of the current applet [6]

3 Defensive VM Layer

In this work, we propose adding a novel security layer to the Java Card. Through this layer, access to internal structures (e.g., OS, LV and BA) of the VM is handled. In reference to its defensive nature and its primary use for enabling a defensive VM, we name this layer the defensive VM (D-VM) layer. An overview of the D-VM layer and the D-VM API, which is used by the VM, is depicted in Figure 2 and is explained in detail below.

Functionalities offered by the D-VM API include, for example, pushing and popping data onto the OS, writing and reading from the LV and fetching Java bytecodes. It is possible for the VM to implement all Java bytecodes by using these API functions. The pseudo-code example in Listing 1.1 shows the process of fetching a bytecode and the implementation of the *sadd* bytecode using our D-VM API approach. The *sadd* bytecode pops two values of *integral data* type from the OS and pops the sum as an *integral data* type back onto the OS.

Listing 1.1. Pseudo-code of the VM using the API functions of the newly introduced D-VM layer.

```
 //use the D-VM API to fetch the next bytecode from the BA
switch(dvm_fetch_bytecode())
{
  ...
  case sadd: //implementation of the sadd bytecode.
  {
     //use the D-VM API to obtain the two values from the OS
    result = dvm_pop_integralData() + dvm_pop_integralData();
     //use the D-VM API to write the sum back onto the OS
    dvm_push_integralData(result);
  }
  ...
}
```

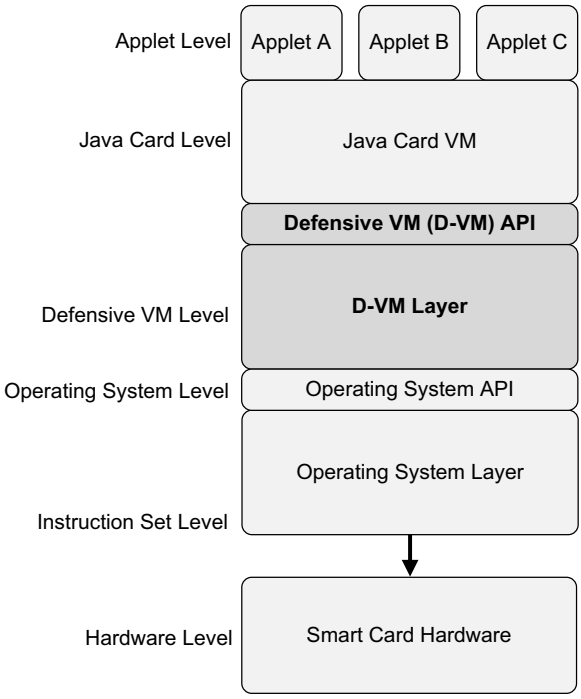


Fig. 2. The VM executes Java Card applets and uses the newly introduced D-VM layer to secure the Java Card against FAs

The security mechanisms within the security layer intended to protect the VM from FAs are hidden from the VM programmer. A security architect, specialized for VM security, is able to implement and choose the appropriate countermeasures within the D-VM layer. These countermeasures are based on state-of-the-art knowledge and the hardware constraints of the smart card architecture. Programmers implementing the VM do not need to know these security techniques in detail but rather just use the D-VM API functions.

If HW features are used, the D-VM layer communicates with these units and configures them through specific instructions. Through this approach, it is also very easy to alter the SW implementations by changing the D-VM layer implementation without changing specific Java bytecode implementations. It is possible to fulfill the same security policy on different smart card platforms where specific HW features are available.

On a code size-constrained smart card platform, an implementation that has a small code size but requires more main memory or execution time is used. The appropriate implementations of security features within the D-VM API are used without the need to change the entire VM.

Dynamic Countermeasures: The D-VM layer is also a further step to enable dynamic fault attack countermeasures such as that proposed by Barbu in [3]. In this work, he proposes a VM that uses different bytecode implementations depending on the actual security level of the smart card. If an attack or malicious behavior is detected, the security level is decreased. This decreased security leads to an exchange of the implemented bytecodes with more secure versions. In these more secure bytecodes, different additional checks, such as double reads, are implemented, which leads to decreased run-time performance.

Our D-VM layer further advances this dynamic countermeasure concept. Depending on the actual security level, an appropriate D-VM layer implementation is used. Therefore, the entire bytecode implementation remains unchanged, but it is possible to dynamically add and change security checks during run-time. An overview of this dynamic approach is outlined in Figure 3.

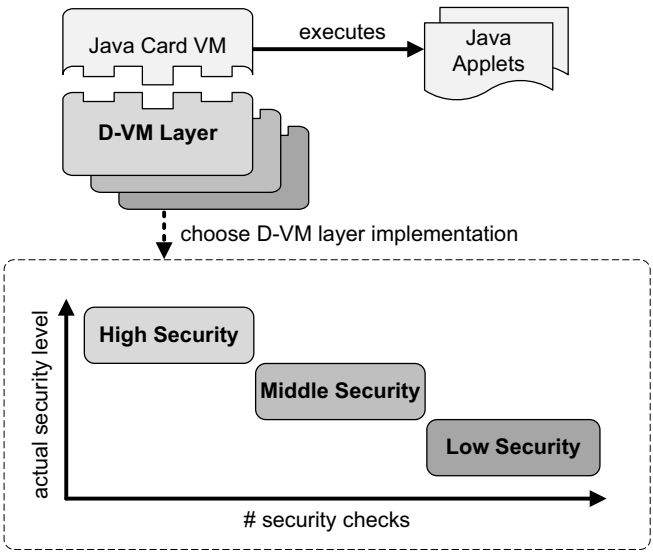


Fig. 3. Based on the current security level of the VM, an appropriate D-VM layer implementation is chosen

The actual security level of the card is determined by HW sensors (e.g., brightness and supply voltage) and the behavior of the executing applet. For example, at a high security level, the D-VM layer can perform a read operation after pushing a value into the OS memory to detect an FA. At a lower security level, the D-VM layer performs additional bound, type and control-flow checks.

Security Context of an Applet: Another use case for the D-VM layer is the post-issuance installation of applets on the card. We focus on the user-centric ownership model (UCOM) [1] in which Java Card users are able to load their own

applets onto the card. For the UCOM approach, each newly installed applet is assigned a defined security level at installation time. The security level depends on how trustworthy the applet is. For example, the security level for an applet signed with a valid key from the service provider is quite high, which results in a high execution speed. Such an applet should be contrasted with an applet that has no valid signature and is loaded onto the card by the Java Card owner. This applet will run at a low security level with many run-time checks but a slower execution speed. Furthermore, access to internal resources and applets installed on the card could be restricted by the low security level.

3.1 Security Policy

This chapter introduces the three security policies used in this work. With the help of these policies, it is possible to counteract the most dangerous threats that jeopardize security-critical data on the card. The type and bound policies are taken from [14] and are augmented with a control-flow policy. The fulfillment of the three policies on every bytecode is checked by three different D-VM layer implementations using our D-VM API.

Control-Flow Policy: The VM is only allowed to fetch bytecodes that are within the borders of the currently active method's BA. Fetching of bytecodes that are outside of this area is not allowed. The actual valid method BA changes when a new method is invoked or a return statement is executed. Because of this policy, it is no longer possible for control-flow changing bytecodes (e.g., *goto_w* and *if_scmp_w*) to jump outside of the reserved bytecode memory area. This policy counters the EMAN4 attack [6] on the Java Card and all other attacks that rely on the execution of a data array or code of an-other applet that is not inside the current BA.

Type Policy: Java bytecodes are strongly typed in the VM specification [19]. This typing means that for every Java bytecode, the type of operand that the bytecode expects and the type of the result stored in the OS or LV are clearly defined. An example is the *sastore* bytecode, which stores a *short* value in an element of a *short* array object. The *sastore* bytecode uses the top three elements from the OS as operands. The first element is the address of the array object, which is of type *reference*. The second element is the index operand of the array, which must be of type *short*. The third element is the value, which is stored within the array element and is of type *short*.

Type confusion between values of integral data (*boolean*, *byte* or *short*) and object references (*byte[]*, *short[]* or *class A*, for example) is a serious problem for Java Cards [24,17,13,25,6,11]. To counter these attacks, we divide all data types into the two main types, *integralData* and *reference*. Note that this policy does not prevent type confusion inside the main type *reference* between array and class types.

Bound Policy: Most Java Card bytecodes push and pop data onto the OS or read and write data into the LV, which can be considered similar to registers. The OS is the main component for most Java bytecode operations. Similar to buffer overflow attacks in C programs [9], it is possible to overflow the reserved memory space for the OS and LV. An adversary is then able to set the return address of a method to any value. Such an attack was first found in 2011 by Bouffard [6,7]. An overflow of the OS happens by pushing or popping too many values onto the OS. An LV overflow happens when an incorrect LV index is accessed. This index parameter is decoded as an operand for several LV-related bytecodes (e.g., *sstore*, *sload* and *sinc*). This operand is therefore stored permanently in the non-volatile memory. Thus, changing this operand through an FA gives an attacker access to memory regions outside the reserved LV memory region. These memory regions are created for every method invoked and are not changed during the method execution. Therefore in this work, we permit Java bytecodes to operate only within the reserved OS and LV memory regions.

4 Java Card Prototype Implementation

In this work three implementations of the D-VM layer are proposed to perform run-time security checks on the currently executing bytecode. Two implementations perform all checks in SW to ensure our security policies. One implementation uses dedicated HW protection units to accelerate the run-time verification process. The implementations of the D-VM layer were added into a Java Card VM and executed on a smart card prototype. This prototype is a cycle-accurate SystemC [12] model of an 8051 instruction set-compatible processor. All software components, such as the D-VM layer and the VM, are written in C and 8051 assembly language.

4.1 D-VM Layer Implementations

This section presents the implementation details for the three implemented D-VM layers used to create a defensive VM. All three implemented D-VM layers fulfill our security policy presented in Chapter 3 but differ from each other in the detailed manner in which the policies are satisfied. The key characteristic of the two SW D-VM implementations is that they use a different implementation of the type-storing approach to counteract type confusion. The run-time type information (*integralData* or *reference*) used to perform run-time checks can be stored either in a type bit-map (memory optimization) or in the actual word size of the microprocessor (speed optimization). The HW Accelerated D-VM uses a third approach and stores the type information in an additional bit of the main memory. Through this approach, the HW can easily store and check the type information for every OS and LV entry. An overview of how the type-storing policy is ensured by our D-VM implementations and a memory layout overview are shown in Figure 4 and explained in detail in the next sections.

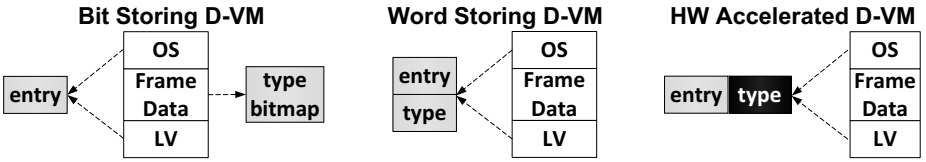


Fig. 4. The Bit Storing D-VM stores the type information for every OS and LV entry in a type bitmap. The Word Storing D-VM stores the type information below the value in the reserved OS and LV spaces. The HW Accelerated D-VM holds the type information as an additional type bit, which increases the memory size of a word from 8 bits to 9 bits.

Bit Storing D-VM: This D-VM layer implementation stores the type information for every element on the OS and LV in a type bitmap. The type information for every entry of the OS and LV is now represented by a one-bit entry. A problem with this approach is that the run-time overhead is quite high because different shift and modulo operations must be performed to store and read the type information from the type bitmap. These operations (shift and modulo) are, for the 8051 architecture, computationally expensive operations and thus lead to longer execution times. An advantage of the bit-storing approach is the low memory overhead required to hold the type information in the type bitmap.

Word Storing D-VM: The run-time performance of the type storing and reading process is increased by storing the type information using the natural word size of the processor and data bus on which the memory for the OS and LV is located. Every element in the OS and LV is extended with a type element of a word size such that it can be processed very quickly by the architecture. By choosing this implementation, the memory consumption of the type-storing process increases compared with the previously introduced SW Bit Storing D-VM. Pseudo-codes for writing to the top of the stack of the OS for the bit- and word-storing approach are shown in Listings 1.2 and 1.3.

Listing 1.2. Operations needed to push an element on the OS by the Bit Storing D-VM

```
dvm_push_integralData(value)
{
  //push value onto OS and
  //increase OS size
 OS[size++] = value;
  //store type information
  //into type bitmap,
  //INT->integralData type
 bitmap[size/8] = INT<<(size%8);
}
```

Listing 1.3. Operations needed to push an element on the OS by the Word Storing D-VM

```
dvm_push_integralData(value)
{
  //push value onto OS
  //increase OS size
 OS[size++] = value;
  //store type information
  //into next memory word,
  //INT->integralData type
 OS[size++] = INT;
}
```

HW Accelerated D-VM: Performing type and bound checks in SW to fulfill our security policy consumes a lot of computational power. Types must be loaded, checked and stored for almost every bytecode. The bounds of the OS and LV must be checked such that no bytecode performs an overflow. The HW Accelerated D-VM layer uses specific HW protection units of the smart card to accelerate these security checks. New protection units (bound protection and type protection) are able to check if the current memory move (MOV) operation is operating in the correct memory bounds. The type information for the OS and LV entries is stored as an additional type bit for every main memory word. The information is decoded into new assembly instructions to specify which memory region (OS, LV or BA) and with which data type (*integralData* or *reference*) the MOV operation should write or read data. An overview of the HW Accelerated D-VM is shown in Figure 5. Depending on the assembly instruction, the HW protection units perform four security operations:

- Check if the Java opcode is fetched from the current active BA.
- Check if the destination address of the operation is within the memory area of the OS or LV. If the operation is not within these two bounded areas, a HW security exception is thrown.
- For every write operation write the type decoded in the CPU instruction into the accessed memory word.
- For every read operation, check if the stored type is equal to the type decoded in the CPU instruction. If they are not equal, throw a hardware security exception.

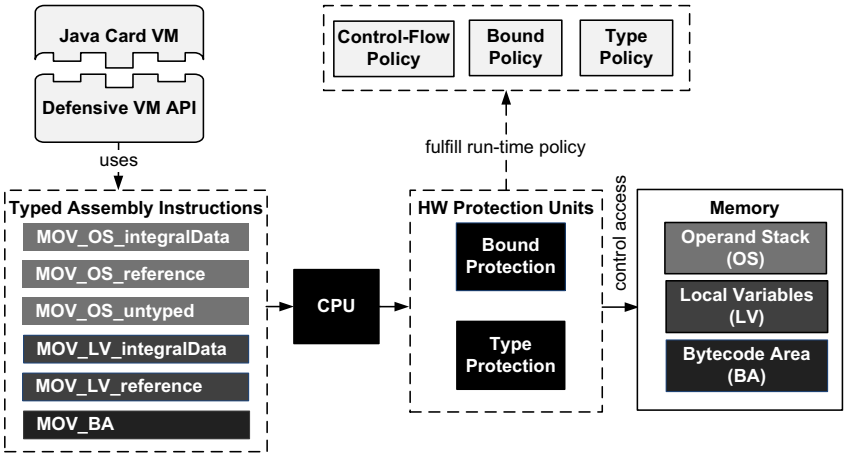


Fig. 5. Overview of the HW Accelerated D-VM implementation using new typed assembly instructions to access VM resources (OS, LV and BA). Malicious Java bytecodes violating our run-time policy will be detected by new introduced HW protection units.

5 Prototype Results

In this section, we present the overall computational overhead of the three implemented D-VM layers and their main memory consumption. All of them are compared with a VM implementation without the D-VM layer. The speed comparison is performed for different groups of bytecodes by self written micro-benchmarks where all bytecodes under test are measured. These test programs first perform an initialization phase where the needed operands for the bytecode under test are written into the OS or LV. After the execution of the bytecode under test the effects on the OS or LV are removed. Note that our smart card platform has no data or instruction cache. Therefore, no caching effects must be taken into account for all test programs.

5.1 Computational Overhead

Speed comparisons for specific bytecodes are shown in Figure 6. For example, the Java bytecode *sload* requires 148% more execution time for the Word Storing D-VM. For the Bit Storing D-VM, the execution overhead is 212%. The increased overhead is because of the expensive calculations used to store the type information in a bitmap. For the HW Accelerated D-VM, the execution speed decreases by only 4% because all type and bound checks are performed using HW. Additional run-time statistics for groups of bytecodes are listed in Table 1. As expected, the Bit Storing D-VM consumes the most overall run-time, with an increase of 208%. The Word Storing D-VM needs 142% more run-time. The HW Accelerated D-VM has only 6% more overhead.

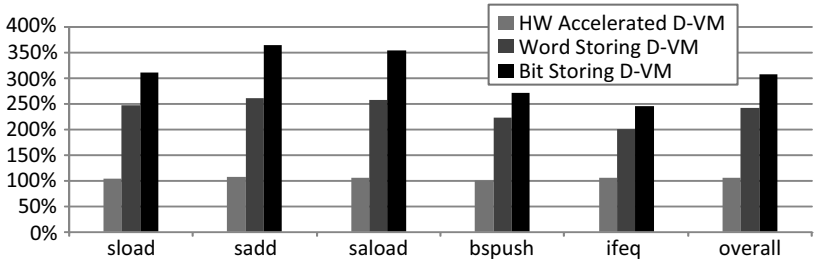


Fig. 6. Speed comparison of individual bytecodes for the different D-VM layer implementations proposed in this work. The results are compared with a VM without the D-VM layer.

5.2 Main Memory Consumption

The HW Accelerated D-VM requires one type bit per 8 bits of data to store the type information during run-time. This results in an overall main memory increase of 12.5%. The Word Storing D-VM requires in the worst case 33% more memoy because one type byte holds the type information for two data bytes.

Table 1. Speed comparison for different groups of bytecodes compared with a VM without the D-VM layer

Bytecode Groups	HW Accelerated D-VM	Word Storing D-VM	Bit Storing D-VM
Arithmetic/Logic	+7%	+146%	+240%
LV Access	+5%	+185%	+243%
OS Manipulation	+5%	+151%	+231%
Control Transfer	+7%	+113%	+173%
Array Access	+5%	+130%	+166%
Overall	+6%	+142%	+208%

The Bit Storing D-VM requires approximately 6.25% more memory in the case in which the entire memory is filled with OS and LV data. This is because the Bit Storing D-VM requires one type bit per 16 bits of data.

6 Conclusions and Future Work

This work presents a novel security layer for the virtual machine (VM) on Java Cards. Because it is intended to defend against fault attacks (FAs), it is called the defensive VM (D-VM) layer. This layer provides access to security-critical resources of the VM, such as the operand stack, local variables and the bytecode area. Inside this layer, security checks, such as type checking, bound checking and control-flow checks, are performed to protect the card against FAs. These FAs are executed during run-time to change the control and data flow of the currently executing bytecode.

By storing different implementations of the D-VM layer on the card, it is possible to choose the appropriate security implementation based on the actual security level of the card. Through this approach, the number of security checks can be increased during run-time by switching among different D-VM implementations. Furthermore, it is possible to assign a trustworthy applet a low security level, which results in high execution performance, and vice versa. One D-VM layer implementation can be, for example, low security with high execution speed or high security with low execution speed. Another advantage is the concentration of the security checks inside the layer.

To demonstrate this novel security concept, we implemented three D-VM layers on a smart card prototype. All three layers fulfill the same security policy (control-flow, type and bound) for bytecodes but differ in their implementation details. Two D-VM layer implementations are fully implemented in software but differ in the manner in which the type information is stored. The Bit Storing D-VM has the highest run-time overhead, 208%, but the lowest memory increase, 6.25%. The Word Storing D-VM decreases the run-time overhead to 142% but consumes approximately 33% more memory. The HW Accelerated D-VM uses dedicated Java Card HW to accelerate the run-time verification process and has an execution overhead of only 6% and a memory increase of 12.5%.

In future work, we will focus on the question of which sensor data should be used to increase the internal security of the Java Card. Another question is how many security states are required and how much they differ in their security needs.

Acknowledgments. The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the CoCoon project under the FIT-IT contract FFG 830601. We would also like to thank our project partner NXP Semiconductors Austria GmbH.

References

1. Akram, R., Markantonakis, K., Mayes, K.: A Paradigm Shift in Smart Card Ownership Model. In: 2010 International Conference on Computational Science and Its Applications (ICCSA), pp. 191–200 (March 2010)
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE 94(2), 370–382 (2006)
3. Barbu, G., Andouard, P., Giraud, C.: Dynamic Fault Injection Countermeasure. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 16–30. Springer, Heidelberg (2013)
4. Barbu, G., Duc, G., Hoogvorst, P.: Java Card Operand Stack: Fault Attacks, Combined Attacks and Countermeasures. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 297–313. Springer, Heidelberg (2011)
5. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 148–163. Springer, Heidelberg (2010)
6. Bouffard, G., Iguchi-Cartigny, J., Lanet, J.-L.: Combined Software and Hardware Attacks on the Java Card Control Flow. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 283–296. Springer, Heidelberg (2011)
7. Bouffard, G., Lanet, J.-L.: The Next Smart Card Nightmare. In: Naccache, D. (ed.) Cryphtography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 405–424. Springer, Heidelberg (2012)
8. Bouffard, G., Lanet, J.-L., Machemie, J.-B., Poichotte, J.-Y., Wary, J.-P.: Evaluation of the Ability to Transform SIM Applications into Hostile Applications. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 1–17. Springer, Heidelberg (2011)
9. Cowan, C., Wagle, P., Pu, C., Beattie, S., Walpole, J.: Buffer overflows: attacks and defenses for the vulnerability of the decade. In: Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems], pp. 227–237 (2003)
10. Dubreuil, J., Bouffard, G., Lanet, J.-L., Cartigny, J.: Type Classification against Fault Enabled Mutant in Java Based Smart Card. In: 2012 Seventh International Conference on Availability, Reliability and Security (ARES), pp. 551–556 (August 2012)
11. Hamadouche, S., Bouffard, G., Lanet, J.-L., Dorsemaine, B., Nouhant, B., Magloire, A., Reygnaud, A.: Subverting Byte Code Linker service to characterize Java Card API. In: Proceedings of the 7th Conference on Network and Information Systems Security (SAR-SSI), pp. 122–128 (2012)

12. IEEE: Open SystemC Language Reference Manual IEEE Std 1666-2005, IEEE
13. Iguchi-Cartigny, J., Lanet, J.-L.: Developing a Trojan applets in a smart card. Journal in Computer Virology 6, 343–351 (2010)
14. Lackner, M., Berlach, R., Loinig, J., Weiss, R., Steger, C.: Towards the Hardware Accelerated Defensive Virtual Machine – Type and Bound Protection. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 1–15. Springer, Heidelberg (2013)
15. Leroy, X.: Bytecode verification on Java smart cards. Software: Practice and Experience 32(4), 319–340 (2002)
16. Markantonakis, K., Mayes, K., Tunstall, M., Sauveron, D., Piper, F.: Smart card security. In: Nedjah, N., Abraham, A., de Macedo Mourelle, L. (eds.) Computational Intelligence in Information Assurance and Security. SCI, vol. 57, pp. 201–233. Springer, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-71078-3_8
17. Mostowski, W., Poll, E.: Malicious Code on Java Card Smartcards: Attacks and Countermeasures. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 1–16. Springer, Heidelberg (2008)
18. Oracle: Runtime Environment Specification. Java Card Platform, Version 3.0.4, Classic Edition (2011)
19. Oracle: Virtual Machine Specification. Java Card Platform, Version 3.0.4, Classic Edition (2011)
20. Razafindralambo, T., Bouffard, G., Thampi, B.N., Lanet, J.-L.: A Dynamic Syntax Interpretation for Java Based Smart Card to Mitigate Logical Attacks. In: Thampi, S.M., Zomaya, A.Y., Strufe, T., Alcaraz Calero, J.M., Thomas, T. (eds.) SNDS 2012. CCIS, vol. 335, pp. 185–194. Springer, Heidelberg (2012)
21. Sauveron, D.: Multiapplication smart card: Towards an open smart card? Information Security Technical Report 14(2), 70–78 (2009); Smart Card Applications and Security
22. Séré, A.A.K., Iguchi-Cartigny, J., Lanet, J.-L.: Checking the Paths to Identify Mutant Application on Embedded Systems. In: Kim, T.-H., Lee, Y.-H., Kang, B.-H., Ślęzak, D. (eds.) FGIT 2010. LNCS, vol. 6485, pp. 459–468. Springer, Heidelberg (2010)
23. Séré, A.A.K., Iguchi-Cartigny, J., Lanet, J.-L.: Evaluation of Countermeasures Against Fault Attacks on Smart Cards. International Journal of Security and Its Applications 5(2), 49–61 (2011)
24. Vertanen, O.: Java Type Confusion and Fault Attacks. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 237–251. Springer, Heidelberg (2006)
25. Vetillard, E., Ferrari, A.: Combined Attacks and Countermeasures. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 133–147. Springer, Heidelberg (2010)

A Forward Privacy Model for RFID Authentication Protocols

Daisuke Moriyama, Miyako Ohkubo, and Shin'ichiro Matsuo

NICT, 4-2-1, National Institute of Information and Communication s Technology,
2-2-1 Nukui-Kitamachi, Koganei-shi, Tokyo 184-8795 Japan
{dmoriyam,m.ohkubo,smatsuo}@nict.go.jp

Abstract. In this paper, we propose a new variant of indistinguishability-based security model for the RFID authentication protocol, which allows an adversary to obtain an authentication result and secret key of a target tag. Ng et al. showed that symmetric-key based RFID authentication protocols cannot be resilient to the above information leakage simultaneously in the Paise-Vaudenay security model. We review the existing result and extend the Juels-Weis security model to satisfy these properties by using a suitable restriction. Moreover, we give two example protocols that satisfy the modified security model.

Keywords: RFID, authentication, security model, forward-privacy.

1 Introduction

The RFID authentication protocol is an authentication protocol in which an RFID reader communicates with RFID tags and authenticates them. This protocol is considered to be one of the most important techniques for construct the world of "Internet of Things", in which all objects automatically interact with each other with wireless communication [19]. However, many people worry about their privacy with RFID attached objects (e.g., CASPIAN). Therefore, the RFID authentication protocol requires that the identity of RFID tags should be kept secret (anonymity) and any transaction should not be linked (unlinkability), except from legitimate RFID readers.

Since 2003, many RFID authentication protocols have been investigated and several protocols have been shown to be insecure (see [5]). The security of these protocols is evaluated by using a cryptographic security model and one of the goals of the RFID authentication protocol is to satisfy this model. The Paise-Vaudenay security model [17] divides the privacy level into eight categories and these are roughly divided into four groups: whether a malicious adversary can obtain an authentication result or not (wide/narrow), and whether the internal secret key of the target tag is finally revealed or not (forward/weak). However, Ng et al. classified symmetric-key based RFID authentication protocols into four types and showed that these protocols only satisfy either narrow-forward privacy or wide-weak privacy [8]. In particular, their result showed that authentication protocols which have a key update mechanism and resynchronization property,

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 98–111, 2013.

satisfy only wide-weak privacy, which can be achieved by protocols in which the secret key is always fixed. Coisel and Martin suspected that the provable security level of the SK-based protocol [9] (which does not require any secret key update) and O-FRAP (key update mechanism is clearly shown) are equivalent [5].

In this paper, we propose a variant of the Juels-Weis security model [11] to investigate a suitable security model for RFID authentication protocol. Our model allows an adversary to obtain an authentication result and the secret key of a target tag. Note that we cannot simply send the secret key to the adversary since the resulting model is not achievable. Instead, we add the rule that the reader and the target tags must properly execute a session before the adversary obtains the secret key of the target tag. If the current secret key is appropriately updated in an honest execution, the adversary cannot distinguish which tag communicated with the reader in the past executions. We show that O-FRAP [7] and a variant of the OSK protocol [15] satisfy the modified security model. Furthermore, we show another separation based on the classification in [8] in which the RFID tag can interact with the reader concurrently.

2 Preliminaries

2.1 Notations

We denote a set of k-bit string as $\{0,1\}^k$. 1^k is a k-bit string of 1. $x \xleftarrow{\mathsf{U}} X$ means that variable x is randomly chosen from set X. If f is a probabilistic algorithm, $b \xleftarrow{\mathsf{R}} f(a)$ denotes that the output from f on input a is assigned to b. $\Pr[f(a) \to b]$ evaluates the probability that the algorithm f outputs b on input a. We say that a probability $P(k)$ is negligible in k if for any polynomials f it holds that $P(k) \leq 1/f(k)$ for sufficient large k.

2.2 Pseudo-random Generator

Let k be a security parameter. The pseudo-random generator is a deterministic algorithm g that takes as input 1^k and truly random secret $x \in \{0,1\}^k$ and outputs $g(x)$ which is computationally indistinguishable from random string y ($|g(x)| > |x|$). In this paper, we treat the expansion factor as k and consider pseudo-random generator $g : \{0,1\}^k \to \{0,1\}^{2k}$. The advantage of a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ for pseudo-random generator g is defined by $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},g}(k) := |\Pr[\mathcal{A}(1^k, g(x)) \to 1 \mid x \xleftarrow{\mathsf{U}} \{0,1\}^k] - \Pr[\mathcal{A}(1^k, y) \to 1 \mid y \xleftarrow{\mathsf{U}} \{0,1\}^{2k}]|$.

Definition 1. *A deterministic algorithm $g : \{0,1\}^k \to \{0,1\}^{2k}$ is pseudo-random generator if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},g}(k)$ is negligible in k.*

We split the above pseudo-random generator g and consider two functions (g_1, g_2) such that the output of these functions are k-bit where $y_1 = g_1(x), y_2 = g_2(x)$ and $y_1 \| y_2 := g(x)$. Let $x_i := g_2^i(x_0)$ be i-round iterated function with input x_0. When we consider $y_i := g_1(x_i)$, Berbain et al. proved that the function G :

$\{0,1\}^k \to \{0,1\}^{2kn+k}$ s.t. $(\{(x_i, y_i)\}_{0 \le i \le n}, x_{n+1}) := G(x_0)$ is also the pseudo-random generator [2]. In particular, they showed that for any PPT adversary $\mathcal{A}$, there exists a PPT time algorithm $\mathcal{B}$ such that $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},G}(k) \le n^2 \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},g}(k)$.

2.3 Pseudo-random Function

The pseudo-random function is a function that takes as input a truly random secret (seed) $x \in \{0,1\}^k$ and k-bit string, which output is computationally indistinguishable from truly random function RF. The advantage of an adversary $\mathcal{A}$ against the pseudo-random function $f : \{0,1\}^k \to \{0,1\}^k \to \{0,1\}^k$ is defined as $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},f}(k) := |\Pr[\mathcal{A}^{f(x,\cdot)}(1^k) \to 1 \mid x \xleftarrow{\mathsf{U}} \{0,1\}^k] - \Pr[\mathcal{A}^{\mathsf{RF}}(1^k) \to 1]|$.

Definition 2. *We say that $f : \{0,1\}^k \to \{0,1\}^k$ is pseudo-random function if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},f}(k)$ is negligible in k.*

3 Security Model for RFID Authentication

The RFID authentication protocol is an authentication protocol between the RFID reader $\mathcal{R}$ and RFID tag $t \in \mathcal{T}$. The RFID reader runs a setup algorithm Setup and generates a public parameter and secret key $(pk, sk) \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^k)$. The reader and each tag share a secret key in the symmetric-key based protocol. After the initialization, the reader and each tag communicate with each other in a wireless setting during the authentication phase. Finally, they output "accept" or "reject" as a result. Following the previous security models [11,17], we assume that the tag only performs the sequential session with the reader, though the reader can interact with a lot of tags concurrently. We consider an active adversary as one who can modify any communication message between the reader and tags.

The RFID authentication protocol requires correctness, security and privacy and many security models are provided to formalize them [1,2,4,6,7,10,11,14,17,18]. In this paper, we omit detailed definitions for correctness and security since almost all security models commonly define them. Correctness means that the reader accepts the RFID tag when the session is completed and the communication message is not modified. Security requires that the reader reject the session when the communication is modified by the adversary. In the following, we concentrate on the definition of *privacy* in the security model, which we call the "privacy model".

3.1 Juels-Weis Privacy Model

Juels and Weis introduced an indistinguishability based privacy definition for the RFID authentication protocol in 2007 [10,11].

Consider the following experiment $\mathsf{Exp}^{\mathsf{IND}\text{-}b}_{\Pi,\mathcal{A}}(k)$ between adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ and a challenger in an RFID authentication protocol Π.

$\underline{\mathsf{Exp}^{\mathsf{IND}\text{-}b}_{\Pi,\mathcal{A}}(k)}$
$(pk, sk) \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^k)$;
$(t_0^*, t_1^*, st_1) \xleftarrow{\mathsf{R}} \mathcal{A}_1^{\mathsf{ReaderInit},\mathsf{Send},\mathsf{Corrupt},\mathsf{Result}}(pk, \mathcal{R}, \mathcal{T})$;
$\mathcal{T}' := \mathcal{T} \setminus \{t_0^*, t_1^*\}$;
$b' \xleftarrow{\mathsf{R}} \mathcal{A}_2^{\mathsf{ReaderInit},\mathsf{Send},\mathsf{Corrupt},\mathsf{Result}}(\mathcal{R}, \mathcal{T}', \mathcal{I}(t_b^*), st_1)$;
Output b'

The adversary $\mathcal{A}$ can interact with the reader and many tags in $\mathcal{T}$ by using the oracle query $\{\mathsf{ReaderInit}, \mathsf{Send}, \mathsf{Corrupt}, \mathsf{Result}\}$. $\mathsf{ReaderInit}(1^k)$ activates the session by the reader and $\mathcal{A}$ sends an arbitrary message m by using $\mathsf{Send}(t_i, m)$ $(t_i \in \mathcal{T})$. In addition, the secret key of the tag can be obtained by $\mathsf{Corrupt}(t_i)$ and $\mathsf{Result}(\mathsf{sid})$ outputs the authentication result (namely, accept or reject) of the session sid. When $\mathcal{A}$ outputs two tags (t_0^*, t_1^*) in $\mathcal{T}$, the challenger flips a coin $b \xleftarrow{\mathsf{U}} \{0, 1\}$ and $\mathcal{A}$ interacts with t_b^* anonymously ($\mathcal{I}(t_b^*)$ means the anonymous access to the tag). $\mathcal{A}$ cannot issue $\mathsf{Corrupt}$ query to (t_0^*, t_1^*) in this model. The advantage of $\mathcal{A}$ in this experiment is defined by $\mathsf{Adv}^{\mathsf{IND}}_{\Pi,\mathcal{A}}(k) = |\Pr[\mathsf{Exp}^{\mathsf{IND}\text{-}0}_{\Pi,\mathcal{A}}(k) \to 1] - \Pr[\mathsf{Exp}^{\mathsf{IND}\text{-}1}_{\Pi,\mathcal{A}}(k) \to 1]|$.

Definition 3. *An RFID authentication protocol Π satisfies Juels-Weis privacy model if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}_{\Pi,\mathcal{A}}(k)$ is negligible in k.*

3.2 Paise-Vaudenay Privacy Model

Paise and Vaudenay defined a simulation based privacy model for the RFID authentication protocol. In this model, the adversary can issue a $\{\mathsf{CreateTag}, \mathsf{DrawTag}, \mathsf{Free}\}$ query in addition to $\{\mathsf{ReaderInit}, \mathsf{Send}, \mathsf{Corrupt}, \mathsf{Result}\}$ in the Juels-Weis privacy model. The $\mathsf{CreateTag}$ query registers a new free tag but this tag still cannot communicate with the reader. Instead, the $\mathsf{DrawTag}$ query converts the free tag into a virtual tag, which can interact with the reader (the adversary can input arbitrary tags and a distribution to transform the tag) and the Free query converts the virtual tag into the original free tag. This model considers a 4×2 matrix for the adversary's capabilities in order to classify the privacy level as follows.

1. For the Result query:
   (a) *Wide* — The adversary can issue the result query.
   (b) *Narrow* — The adversary cannot issue the result query.
2. For the $\mathsf{Corrupt}$ query:
   (a) *Strong* — The adversary can issue the corrupt query at any time.
   (b) *Destructive* — Corrupted tags execute no session.
   (c) *Forward* — The adversary cannot issue any other queries after the corrupt query is sent to a tag.
   (d) *Weak* — The adversary cannot issue the corrupt query.

When we consider $\mathcal{O}_1 := \{\mathsf{CreateTag}, \mathsf{DrawTag}, \mathsf{Free}, \mathsf{Corrupt}\}$ and $\mathcal{O}_2 := \{\mathsf{Launch}, \mathsf{Send}, \mathsf{Result}\}$, wide-strong privacy is described by the following experiment against an RFID authentication protocol Π.

$\underline{\mathsf{Exp}^{\mathsf{SIM\text{-}0}}_{\Pi,\mathcal{A}}(k)}$	$\underline{\mathsf{Exp}^{\mathsf{SIM\text{-}1}}_{\Pi,\mathcal{A},\mathcal{S}}(k)}$
$(pk, sk) \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^k)$;	$(pk, sk) \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^k)$;
$b \xleftarrow{\mathsf{R}} \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2}(pk, \mathcal{R})$:	$b \xleftarrow{\mathsf{R}} \mathcal{A}^{\mathcal{O}_1,\mathcal{S}(pk)}(pk)$:
Output b	Output b

The adversary $\mathcal{A}$ can access the reader and tags with $\mathcal{O}_2$ directly in the experiment on the left-side, but the right-side experiment requires a simulator $\mathcal{S}$ to simulate its interaction. Note that $\mathcal{S}$ can obtain arbitrary information corresponding to the $\mathcal{O}_1$ query issued by $\mathcal{A}$. The advantage of the adversary is defined by $\mathsf{Adv}^{\mathsf{SIM}}_{\Pi,\mathcal{A},\mathcal{S}}(k) = |Pr[\mathsf{Exp}^{\mathsf{SIM\text{-}0}}_{\Pi,\mathcal{A}}(k) \to 1] - \Pr[\mathsf{Exp}^{\mathsf{SIM\text{-}1}}_{\Pi,\mathcal{A},\mathcal{S}}(k) \to 1]|$.

Definition 4. *An RFID authentication protocol Π satisfies the Paise-Vaudenay privacy model if for any PPT adversary $\mathcal{A}$, there exists an algorithm $\mathcal{S}$ such that $\mathsf{Adv}^{\mathsf{SIM}}_{\Pi,\mathcal{A},\mathcal{S}}(k)$ is negligible in k.*

Note that we can define the other privacy levels in a similar way. In particular, we focus mainly on wide-forward privacy and narrow-weak privacy in the Paise-Vaudenay privacy model.

Recently, Moriyama et al. showed several relationships between Juels-Wies and Paise-Vaudenay privacy model [13]. But the strongest privacy level requires public key cryptography [16] and the weakest privacy model does not include any tag's corruption. The motivation of the paper is to investigate a suitable privacy model which a symmetric-key based protocol can satisfy the provably security.

4 Desynchronization Problem in RFID Authentication

In 2009, Ng et al. revisited the Paise-Vaudenay privacy model from the perspective of key synchronization [8]D They classified the symmetric key based RFID authentication protocols on the basis of the key update and showed the limitation of archiving the privacy level in the Paise-Vaudenay privacy model.

Type 0. Any secret key of the protocol is not updated.
Type 1. The tag always updates the secret key whenever the session is executed. The tag does not authenticate the reader in these types of protocols.
Type 2a. After the reader authenticates the tag, the secret key of the tag is updated when the tag authenticates the reader.
Type 2b. Before the reader authenticates the tag, the secret key of the tag is updated. Different from Type 1, the tag authenticates the reader in this type of protocol. If the tag downgrades its secret key when the reader authentication fails (e.g., the tag keeps the early state of the secret key for this property), we call it the "type 2b′ protocol".

Type 0: Type 0 protocols do not support narrow-forward privacy since there is no key update mechanism. Consider the following steps to break narrow-forward privacy: Register two tags (t_0, t_1) with the $\mathsf{CreateTag}$ query and generate one virtual tag with the $\mathsf{DrawTag}$ query with input (t_0, t_1) and uniformly random

distribution. Observe a session between the virtual tag and the reader, convert the virtual tag to the free tag with the Free query and obtain the secret keys with the Corrupt query . Then, the adversary easily checks which tag is communicated to the reader, which no simulator can do without a probability higher than 1/2.

Type 1: Type 1 protocols do not support wide-weak privacy since the tag always updates its secret key. From its property, this kind of protocol generally defines $q_{\max}$ which is the upper bound for resynchronizing the secret key between the reader and tag. Therefore, Ng et al. described the following steps:

1. Generate two tags (t_0, t_1) with the CreateTag query.
2. Convert t_0 to $vtag$ with the DrawTag query.
3. Issue the SendTag query to $vtag$ to execute $q_{\max} + 1$ sessions, but interfere with the message being sent to the reader after the tag updates its secret key.
4. Convert $vtag$ back to t_0 with the Free query.
5. Generate one virtual tag with the DrawTag query with input (t_0, t_1) and a uniformly random distribution.
6. Execute a session between the virtual tag and the reader normally (the adversary only eavesdrops on the communication).
7. Obtain the authentication result of the above session with the Result query.

When t_0 is chosen by the second DrawTag query, the reader cannot resynchronize with the tag, so it outputs "reject". However, the reader accepts the virtual tag when t_1 is chosen by the query. Since the adversary can learn which tag is chosen, type 1 protocols cannot satisfy wide-weak privacy.

Type 2a: A key update algorithm for the tag is executed only if the tag authenticates the reader in the type 2a protocols. When the adversary modifies the communication messages from the reader and forces the tag to reject all sessions, the tag cannot update its secret key, so it executes all sessions with a fixed secret key. The adversary can learn which tag executes the session when the adversary issues the Corrupt query in the final step of the experiment. Therefore these protocols do not satisfy narrow-forward privacy.

Type 2b and 2b′: In these types, the tag runs the key update algorithm before the reader. Type 2b protocols cannot satisfy wide-weak privacy since the secret key transition for these protocols is equivalent to that for the type 1 protocols, regardless of the reader authentication. When the authentication protocol is classified in type 2b′, the adversary can fix the secret key of the tag when the communication message from the reader is erased and the adversary sends a random message. Eventually, the privacy level of the type 2b′ protocols is the same as that of the type 2a protocols (narrow-forward privacy failure).

This result shows that any symmetric key based RFID authentication protocol cannot satisfy both wide-weak privacy and narrow-forward privacy in the Paise-Vaudenay privacy model. However, we consider that the privacy level of the type 2a and 2b′ protocols are not equivalent to type 0 protocols. The type 2a and 2b′

protocols clearly specify the key update/downgrade procedure to resynchronize the secret key and provide the notion of "forward-privacy".

5 The Modified Forward Privacy Model

In this section, we illustrate a variant of the Juels-Weis privacy model that allows the adversary to issue the Corrupt query to the challenge tags. Note that the Paise-Vaudenay privacy model does not explicitly define which and when a tag is considered to be the target. In contrast, the Juels-Weis privacy model is easy to modify since the interaction between the challenge tag and adversary is clear. The classification described in Section 4 is generally applied to the RFID authentication protocol. If we purposely add the Corrupt query during the anonymous access phase of this model, no symmetric key based RFID authentication protocol satisfies the modified model because the Result query is given to the adversary.

Instead, if the RFID tag can normally interact with the reader (this means the adversary only eavesdrops on the communication) and resynchronization is completed, the adversary cannot trace which tag interacts with the reader even when the current secret key of the tag is revealed (of course, we assume that the updated key is hard to invert).

Consider the following game between the challenger and adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ based on the Juels-Weis privacy model. Let $\pi \xleftarrow{R} \mathsf{Execute}(\mathcal{R}, t)$ be one normal execution of the session between the reader and tag t, and π denotes the communication message.

$\underline{\mathsf{Exp}^{\mathsf{IND}^*\text{-}b}_{\Pi,\mathcal{A}}(k)}$
$(pk, sk) \xleftarrow{R} \mathsf{Setup}(1^k)$;
$(t_0^*, t_1^*, st_1) \xleftarrow{R} \mathcal{A}_1^{\mathsf{ReaderInit},\mathsf{Send},\mathsf{Corrupt},\mathsf{Result}}(pk, \mathcal{R}, \mathcal{T})$;
$\mathcal{T}' := \mathcal{T} \setminus \{t_0^*, t_1^*\}$;
$st_2 \xleftarrow{R} \mathcal{A}_2^{\mathsf{ReaderInit},\mathsf{Send},\mathsf{Result}}(\mathcal{R}, \mathcal{T}', \mathcal{I}(t_b^*), st_1)$;
$\pi_b^* \xleftarrow{R} \mathsf{Execute}(\mathcal{R}, t_b^*), \pi_{1-b}^* \xleftarrow{R} \mathsf{Execute}(\mathcal{R}, t_{1-b}^*)$;
$b' \xleftarrow{R} \mathcal{A}_3^{\mathsf{ReaderInit},\mathsf{Send},\mathsf{Corrupt},\mathsf{Result}}(\mathcal{R}, \mathcal{T}, \pi_b^*, \pi_{1-b}^*, st_2)$:
Output b'

The advantage of the adversary in the modified model is defined by $\mathsf{Adv}^{\mathsf{IND}^*}_{\Pi,\mathcal{A}}(k) = |\Pr[\mathsf{Exp}^{\mathsf{IND}^*\text{-}0}_{\Pi,\mathcal{A}}(k) \to 1] - \Pr[\mathsf{Exp}^{\mathsf{IND}^*\text{-}1}_{\Pi,\mathcal{A}}(k) \to 1]|$.

We add the honest execution which the reader and tag communicate without the adversary's interruption. Instead, the adversary can issue the Corrupt query to target tags t_0^* and t_1^* after the honest execution from the original Juels-Weis privacy model. If the normal execution *securely* updates the secret key of the reader and tag, this results in the privacy of the challenge tags for the type 2a and 2b′ protocols. The adversary $\mathcal{A}_3$ can obtain the secret key of the target tag and easily break the privacy for type 0 protocols (e.g., SK-based protocol [9,5]).

Definition 5. *An RFID authentication protocol Π satisfies the modified Juels-Weis privacy model if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}^*}_{\Pi,\mathcal{A}}(k)$ is negligible in k.*

In this paper, we only concentrate on the modification of the Juels-Weis privacy model because the simulation-based privacy model does not explicitly define which is the actual target from the malicious adversary. So it is difficult to insert the honest execution to the Paise-Vaudenay privacy model Note that if we omit the honest execution in the modified model, we can easily describe an attack scenario for any symmetric key based RFID authentication protocols as in Section 4. Also note that the honest communication between the reader and tag must be executed *after* the anonymous communication phase is finished. Billet, Etrog and Gilbert formalized the forward privacy model such that the honest execution occurs before the anonymous communication [3]. However, their model is not achievable since such a honest execution is useless and does not prevent any of the attacks described in Section 4 (Specifically, their proposed protocol PEPS falls into a type 2a protocol and does not satisfy their privacy model).

6 Suitable RFID Authentication Protocols

6.1 O-FRAP

O-FRAP is an RFID authentication protocol proposed by Li, Burmester and de Medeiros in 2007 [7]. This protocol falls into type 2a and does not hold narrow-forward privacy [17]. In this paper, we show that O-FRAP satisfies the modified privacy model. Remark that O-FRAP satisfies universally composable (UC) security, but [7] does not provide any relationship between their model and other security models. So the adequateness of the UC definition and its protocol described in [7] is not investigated. Our result shows that a UC-secure protocol also holds provable security in the indistinguishability-based privacy model.

Let k be a security parameter, $f : \{0,1\}^k \times \{0,1\}^{2k} \to \{0,1\}^{4k}$ be pseudo-random function and ℓ be the total number of tags in the protocol. Each tag contains its secret key sk_i and nonce r_2. The reader keeps its database $\{sk_i, sk_{i.old}\}_{1 \le i \le \ell}$ which contains the current and previous secret keys of each tag. The reader and a tag t_i execute the following authentication.

1. The reader chooses $r_1 \xleftarrow{\mathsf{U}} \{0,1\}^k$ and sends it to the tag.
2. When the tag t_i obtains r_1, it computes $(r_{temp}, s_2, s_3, sk'_i) := f(sk_i, r_1 \| r_2)$ and sends (r_2, s_2) to the reader. Then the nonce r_2 is updated as $r_2 := r_{temp}$.
3. Upon receiving (r_2, s_2), the reader performs the following:
   - Compute $(\cdot, s''_2, s''_3, sk''_i) := f(sk_i, r_1 \| r_2)$ for $1 \le i \le \ell$ and check if $s''_2 = s_2$. If so, set $sk_{i.old} := sk_i, sk_i := sk''_i, s'_3 := s''_3$ and accept the tag.
   - If $s''_2 = s_2$ where $(\cdot, s''_2, s''_3, sk''_i) := f(sk_{i.old}, r_1 \| r_2)$ for some $1 \le i \le \ell$, the tag sets $sk_i := sk''_i, s'_3 := s''_3$ and accepts the tag.
   - Otherwise, select $s'_3 \xleftarrow{\mathsf{U}} \{0,1\}^k$ and reject all tags.

Finally, the reader sends s'_3 to the tag[1].

4. When the tag receives s'_3, it checks whether $s'_3 = s_3$. If the equation holds, then the tag updates the secret key as $sk_i := sk'_i$ and accepts the reader. Otherwise, the tag outputs "reject".

The main building block of this protocol is the challenge-response authentication with the pseudo-random function f. In addition, the reader keeps the previous secret key of each tag to resynchronize the secret key to the tag when the desynchronization occurs.

Theorem 1. *Assume that f is a pseudo-random function. Then, O-FRAP satisfies the modified Juels-Weis privacy model.*

Proof. Let q be a upper bound by which the RFID tag performs the key update procedure. Note that q is bounded by the adversarial oracle query and it is at most polynomial in k. S_i is the event that the adversary outputs 1 in Game i.

Game 0. This is the original game between the challenger and adversary in the modified Juels-Weis privacy model.
Game 1-(i,j). We gradually change the output of all the sessions executed in this game:

1. For any session executed between the reader and tag $t_{i'}$ $(t' < i)$, the output of the pseudo-random function is changed with a uniformly random string over $\{0,1\}^{4k}$.
2. For i-th tag t_i,
   2-1. When the number of key update is less than j $(j' < j)$, the output of the pseudo-random function is changed with a uniformly random string $(r_{temp}, s_2, s_3, sk'_{j'}) \xleftarrow{\cup} \{0,1\}^{4k}$.
   2-2. If the number of key update j' holds $j' \geq j$, the output of the session is computed by the pseudo-random function f.
3. For any session executed between the reader and tag $t_{i'}$ $(i' > i)$, all output and update keys are computed by the pseudo-random function.

Lemma 1. $\Pr[S_0] = \Pr[S_{1\text{-}(1,0)}]$.

This transformation is purely conceptual and there is no difference between these games.

Lemma 2. *For all $1 \leq i \leq \ell$ and $0 \leq j \leq q$, there exists an algorithm $\mathcal{B}$ for pseudo-random function f such that $|\Pr[S_{1\text{-}(i,j)}] - \Pr[S_{1\text{-}(i,j+1)}]| \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},f}(1^k)$.*

[1] We slightly modify O-FRAP so that the reader does not abort the session and output a random variable even when the search procedure is failed. Otherwise, the Result query becomes meaningless since the adversary can trivially know the authentication result.

Proof. If the adversary can distinguish the difference between Game 1-(i,j) and Game 1-$(i,j+1)$ with non-negligible probability, then there exists an algorithm $\mathcal{B}$ that breaks the security of the pseudo-random function. The only difference between these games is that the outputs come from the pseudo-random function or truly random string when the tag t_i executes the sessions with j-th updated key.

The oracle queries which algorithm $\mathcal{B}$ can issue is either the pseudo-random function $f(sk_i,\cdot)$ or truly random function RF. $\mathcal{B}$ generates secret key of the tag except t_i in the set up and simulates the session between the reader and tags. If the number of key update on t_i is less than j, we set uniformly random string over $\{0,1\}^{4k}$ as the output of the session. Similarly, if the key update on t_i is executed more than j, we computes the output with the pseudo-random function. If the number of key update on t_i is j, $\mathcal{B}$ computes the output as follows. When $\mathcal{A}$ issues $\mathsf{ReaderInit}$, $\mathcal{B}$ generates $r_1 \xleftarrow{\mathsf{U}} \{0,1\}^k$ and sends r_1 to $\mathcal{A}$. Upon receiving $\mathsf{SendTag}(t_i, r_1')$ from the adversary, $\mathcal{B}$ chooses $r_2 \xleftarrow{\mathsf{U}} \{0,1\}^k$ and issues $r_1'\|r_2$ to the oracle query. The output of the tag in this session consists of the response from the oracle $(r_{temp}, s_2, s_3, sk'_{j'}) \in \{0,1\}^{4k}$. When the adversary sends (r_1, r_2', s_2') to the reader, $\mathcal{B}$ issues $r_1\|r_2'$ to the oracle and verifies the message as the protocol specification. If the verification is accepted, $\mathcal{B}$ set s_3 as the output of the reader. Otherwise, the output of the reader is set as $s_3 \xleftarrow{\mathsf{U}} \{0,1\}^k$. Finally, when $\mathcal{A}$ outputs a bit b, $\mathcal{B}$ outputs the same bit.

If the pseudo-random function is given to $\mathcal{B}$, the above game is equivalent to Game 1-(i,j) from the view point of the adversary $\mathcal{A}$. Otherwise, the distribution of the message is the same as Game 1-$(i,j+1)$. Therefore we have $|\Pr[S_{1\text{-}(i,j)}] - \Pr[S_{1\text{-}(i,j+1)}]| \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},f}(1^k)$.

Lemma 3. *For any* $1 \leq i \leq \ell$, *we have* $\Pr[S_{1\text{-}(i,q+1)}] = \Pr[S_{1\text{-}(i+1,0)}]$.

It is clear that 3 hold since the game transformation between them is purely conceptual (the distribution is equivalent).

From these lemmas, we can transform Game 0 to Game 1-$(\ell+1,0)$. This time, all the output of the session in the last game is independently chosen and truly random. That is, the probability that $\mathcal{A}$ can guess which tag is selected in the anonymous communication phase is $1/2$. Thanks to $\mathsf{Execute}$, the secret key of the tag is completely updated and its secret key is uniformly chosen and independent from the previous sessions. Therefore the adversary can obtain no information about the past session of the tag.

Finally, we obtain $\mathsf{Adv}^{\mathsf{IND}^*}_{\Pi,\mathcal{A}}(k) \leq \ell(q+1) \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}}(1^k)$.

6.2 The Modified OSK Protocol

The OSK Protocol is an RFID authentication protocol provided by Ohkubo et al. in 2003 and this is one of the type 1 protocols [15]. In this paper, we add the reader authentication and the roll back property for the secret key to the OSK protocol. We show that the modified protocol satisfies IND*-privacy.

Consider that k is a security parameter, $g : \{0,1\}^k \to \{0,1\}^k \times \{0,1\}^k$ is a pseudo-random generator and $f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^{2k}$ is a pseudo-random function. The reader keeps $\{sk_i\}_i$ and each tag contains its secret key sk'_i where $(\cdot, sk'_i) = g(sk_i)$.

1. The reader chooses $r_1 \xleftarrow{\cup} \{0,1\}^k$ and sends it to the tag.
2. When the tag t_i obtains r_1, it performs the following:
   (a) Compute $(u'_1, u'_2) := g(sk'_i)$.
   (b) Select $r_2 \xleftarrow{\cup} \{0,1\}^k$ and compute $s_2 := f(u'_1, r_1 \| r_2)$.
   (c) Set $sk_{i.temp} := sk_i$ and $sk_i := u'_2$.
   (d) Send (r_2, s_2) to the reader.
3. Upon receiving (r_2, s_2), the reader computes $(u_1, u_2) := g(sk_i)$ for $1 \le i \le \ell$ and performs the following:
   – Compute $(u''_1, u''_2) := g(u_2)$ and check $s_2 = f(u''_1, r_1 \| r_2)$. If so, compute $s'_3 := f(u''_1, r_2 \| r_1)$, set $sk_i := u_2$ and accept the tag.
   – If $s_2 = f(u_1, r_1 \| r_2)$ compute $s'_3 := f(u_1, r_2 \| r_1)$ and accept the tag.
   – Otherwise, select $s'_3 \xleftarrow{\cup} \{0,1\}^k$ and reject all tags.
   Finally, the reader sends s'_3 to the tag.
4. When the tag receives s'_3, it checks whether $s'_3 = f(sk_{i.temp}, r_2 \| r_1)$ or $s'_3 = f(sk_i, r_2 \| r_1)$. If either equation holds, then the tag accepts the reader. Otherwise, the tag sets $sk_i := sk_{i.temp}$ and output "reject".

Similar to the original OSK protocol, the updated secret key is deterministically defined by the current secret key. Thus the reader can precompute this value if the reader has enough resources. Therefore the reader can find the tag's identity from its database.

Theorem 2. *If g is a pseudo-random generator and f is a pseudo-random function, the above protocol satisfies the modified Juels-Weis privacy model.*

Proof. Let q be an upper bound by which the RFID tag performs the key update procedure. Note that q is bounded by the adversarial oracle query and it is at most polynomial in k. S_i is the event that the adversary outputs 1 in Game i.

Game 0. This is the original attack game in the modified Juels-Weis privacy model.
Game 1-i. We gradually change the output of the pseudo-random generator g as follows:

1. For any session executed between the reader and tag $t_{i'}$ $(t' < i)$, the output of the pseudo-random generator g is changed with a uniformly random string over $\{0,1\}^{2k}$.
2. For any session executed between the reader and tag $t_{i'}$ $(t' \ge i)$, all output and update keys are computed by the pseudo-random generator.

Game 2-(i,j). We modify the output of the pseudo-random function f in the same fashion as O-FRAP.

Lemma 4. *We have* $\Pr[S_0] = \Pr[S_{1\text{-}0}]$.

This change is purely conceptual and it is clear that $\Pr[S_0] = \Pr[S_{1\text{-}0}]$.

Lemma 5. *For any* $0 \leq i \leq \ell - 1$*, there exists an algorithm* $\mathcal{B}$ *such that* $|S_{1\text{-}i} - S_{1\text{-}(i+1)}| \leq \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}(k)$.

Proof. If the adversary can distinguish between Game 1-i and Game 1-$(i+1)$, we construct an algorithm $\mathcal{B}$ that breaks the security of the pseudo-random generator G (see Section 2.3).

$\mathcal{B}$ obtains $\delta := (\{x_i, y_i\}_{0 \leq i \leq q}, x_{q+1})$, where $\delta := G(x_1)$ or $\delta \xleftarrow{\mathsf{U}} \{0,1\}^{nk+k}$. $\mathcal{B}$ computes all the secret keys of the tag except t_{i+1} and simulates the session as Game 1-i. The initial secret key of the tag t_i is set as x_1 and the reader's secret key corresponding to the tag is x_0. $\mathcal{B}$ sets (u_1, u_2) as $(u_1, u_2) := (y_{j+1}, x_{j+2})$ when t_i updated the secret key j times. When $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ outputs the same bit.

If $\mathcal{B}$ is given pseudo-random variables, then the above game is equivalent to Game 1-i from the view point of the adversary. Otherwise, the distribution of the above game is the same as Game 1-$(i+1)$. Therefore we have $|S_{1\text{-}i} - S_{1\text{-}(i+1)}| \leq \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}(k)$.

Lemma 6. *We have* $\Pr[S_{1\text{-}\ell}] = \Pr[S_{2\text{-}(1,0)}]$.

Lemma 7. *For any* $1 \leq i \leq \ell$*, there exists an algorithm* $\mathcal{B}$ *such that* $|\Pr[S_{1\text{-}(i,j)}] - \Pr[S_{1\text{-}(i,j+1)}]| \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},f}(1^k)$ *(*$0 \leq j \leq q$*).*

Lemma 8. *For any* $1 \leq i \leq \ell$*, we have* $\Pr[S_{1\text{-}(i,q+1)}] = \Pr[S_{1\text{-}(i+1,0)}]$.

It is clear that Lemma 4 and 6 hold since these game transformations are purely conceptual. We can easily prove Lemma 7 and 8 on the basis of the proof for Lemma 2 and Lemma 3 in a similar fashion.

Finally, we obtain

$$\mathsf{Adv}^{\mathsf{IND}^*}_{\mathcal{A},\Pi}(k) \leq \ell \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}(k) + \ell(q+1) \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},f}(k).$$

7 Further Separation in Concurrent Execution

The above privacy definition and other previous privacy models [11,17,18] assume that the RFID tag runs only one session at a time (e.g., sequential execution). However, an adversary can issue many oracle queries and interrupt the communication, so the tag may receive the message to start a new session during the execution of another session. Thus we consider the case that the RFID tag can perform the concurrent execution in this section. Of course, this situation may not practical since the resource of cheap tags is limited. Nonetheless, this situation is useful to show the gap between the type 2a and 2b′ protocols.

While these two types of protocols contain the key update mechanism and the secret key of the tag is synchronized to the reader, the secret key input to

the computation of the output message on the tag is different in the concurrent setting. Consider that (m_0, m_1) is the output from the tag in the concurrent execution. In the type 2a protocols, both messages are computed with fixed secret key since the tag does not update the secret key before the reader authentication. Even when the adversary obtains an updated secret key, it is difficult to distinguish the challenge tag from these messages. On the other hand, the secret key is always updated and m_1 is computed by the updated secret key in type 2b$'$ protocols. If the adversary responds with a random message to the tag in the concurrent session, the secret key of the tag is roll backed and the adversary obtains the secret key which is used to compute m_1. Therefore the adversary can distinguish which tag is selected in the anonymous communication phase in the type 2b$'$ protocols if we consider the concurrent setting.

8 Concluding Remarks

In this paper, we proposed a new variant of Juels-Wies privacy model that allows an adversary to issue the result and corrupt queries on the basis of the Juels-Weis privacy model. The RFID tag is quite cheap device and it is hard to implement secure module (e.g., Trusted Platform Module). Thus the secret key leakage is critical issue for RFID authentication protocol. Independently, we can observe the authentication result in many situations (automatic ticket gate, entrance of the private sector, etc). Though Ng et al. showed the separation result described in Section 4, there is a achievable security model that the adversary can obtain these information. We showed two examples and provide concrete security proof for these protocols.

References

1. Akgün, M., Çağlayan, M.U.: Extending an RFID security and privacy model by considering forward untraceability. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) STM 2010. LNCS, vol. 6710, pp. 239–254. Springer, Heidelberg (2011)
2. Berbain, C., Billet, O., Etrog, J., Gilbert, H.: An efficient forward private RFID protocol. In: ACMCCS 2009, pp. 43–53. ACM (2009)
3. Billet, O., Etrog, J., Gilbert, H.: Lightweight privacy preserving authentication for RFID using a stream cipher. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 55–74. Springer, Heidelberg (2010)
4. Burmester, M., Le, T.V., Medeiros, B.D., Tsudik, G.: Universally composable RFID identification and authentication protocols. ACM TISSEC 12(4(21)) (2009)
5. Coisel, I., Martin, T.: Untangling RFID privacy models. ePrint Archive, 2011/636 (2011)
6. Hermans, J., Pashalidis, A., Vercauteren, F., Preneel, B.: A new RFID privacy model. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 568–587. Springer, Heidelberg (2011)
7. Le, T.V., Burmester, M., Medeiros, B.D.: Universally composable and forward-secure RFID authentication and authenticated key exchange. In: ASIACCS 2007, pp. 242–252. ACM (2007)

8. Ng, C.Y., Susilo, W., Mu, Y., Safavi-Naini, R.: New privacy results on synchronized RFID authentication protocols against tag tracing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 321–336. Springer, Heidelberg (2009)
9. International organization for standardization. ISO/IEC 9798: Information technology – Security techniques – Entity authentication, 1991-2010
10. Juels, A., Weis, S.A.: Defining strong privacy for RFID. In: PerCom 2007, pp. 342–347. IEEE (2007)
11. Juels, A., Weis, S.A.: Defining strong privacy for RFID. ACM TISSEC 12(1(7)) (2009)
12. Lim, C.H., Kwon, T.: Strong and robust RFID authentication enabling perfect ownership transfer. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 1–20. Springer, Heidelberg (2006)
13. Moriyama, D., Matsuo, S., Ohkubo, M.: Relations among notions of privacy for RFID authentication protocols. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 661–678. Springer, Heidelberg (2012)
14. Ouafi, K., Phan, R.C.-W.: Traceable privacy of recent provably-secure RFID protocols. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 479–489. Springer, Heidelberg (2008)
15. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic approach to privacy-friendly tags. In: RFID Privacy Workshop (2003)
16. Ouafi, K., Vaudenay, S.: Strong privacy for RFID systems from plaintext-aware encryption. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 247–262. Springer, Heidelberg (2012)
17. Paise, R., Vaudenay, S.: Mutual authentication in RFID: security and privacy. In: ASIACCS 2008, pp. 292–299. ACM (2008)
18. Vaudenay, S.: On privacy models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)
19. Welbourne, E., Battle, L., Cole, G., Gould, K., Rector, K., Raymer, S., Balazinska, M., Borriello, G.: Building the internet of things using RFID: The RFID ecosystem experience. IEEE Internet Computing (2009)

On Secure Embedded Token Design

Quasi-looped Yao Circuits and Bounded Leakage

Simon Hoerder[1], Kimmo Järvinen[2], and Daniel Page[1]

[1] University of Bristol, Department of Computer Science,
Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK
{hoerder,page}@cs.bris.ac.uk
[2] Aalto University, Department of Information and Computer Science,
P.O. Box 15400, FI–00076 Aalto, Finland
kimmo.jarvinen@aalto.fi

Abstract. Within a broader context of mobile and embedded computing, the design of practical, secure tokens that can store and/or process security-critical information remains an ongoing challenge. One aspect of this challenge is the threat of information leakage through side-channel attacks, which is exacerbated by any resource constraints. Along these lines, this paper extends previous work on use of Yao circuits via two contributions. First, we show how careful analysis can fix the maximum number of leakage occurrences observed during a DPA attack, effectively bounding leakage from a Yao-based token. To achieve this we use modularised Yao circuits, which also support our second contribution: the first Yao-based implementation of a secure authentication payload, namely HMAC based on SHA-256.

1 Introduction

A vast range of challenges and opportunities has emerged as a result of the (ongoing) proliferation of mobile and embedded computing. We now routinely and fundamentally depend on a broad range of complex, highly integrated mobile devices and applications and supporting technologies and techniques need to keep pace with such developments. We consider a motivating example within this context, namely the establishment of a secure communication channel between some token and another party. Although we limit our remit to tokens that are more capable than a (basic) smart-card (e.g., a mobile telephone or USB token, with concrete examplars including the IBM ZTIC[1]), the secure, efficient implementation of such a device is clearly of central importance.

Although well studied cryptographic techniques can satisfy varied requirements at a high level, real-world security guarantees are still difficult to achieve: the field of physical security in particular, including passive side-channel attacks, represents a central concern. In general, a typical side-channel adversary acquires execution profiles by observing a device and, in an ensuing analysis phase, has

[1] http://www.zurich.ibm.com/ztic/

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 112–128, 2013.

to recover a security-critical target value (e.g., key material) from the profiles. Although practical bounds on the number of profiles collected or processed may exist, attacker capability in this respect scales over time (e.g., with Moore's law); ideally this should be catered for by any countermeasure.

Our focus is the threat of Simple (SPA) and Differential Power Analysis (DPA) [10], including variations thereof such as electro-magnetic emission analysis. We cater for timing side-channels, but explicitly deem (semi-)invasive or active attacks as outside the scope of this paper. In our scenario, profiles acquired take the form of traces that describe power consumption by the token. One broad class of countermeasures aims to produce an implementation of some primitive, and secure it by applying attack-specific countermeasures at an algorithmic, software and hardware level. Selected examples include schemes for hiding [16, Chapter 7] and masking [16, Chapter 10] input and/or intermediate values. An alternative class aims to formulate then implement a primitive which is secure-by-design. Following a philosophy that says security should be treated as a first-class goal [9,23], this is an attractive direction in that robust guarantees can be provided. However, as the emerging field of leakage-resilient primitives (see [25] for example) illustrates, difficulties remain. Most importantly, leakage-resilient cryptography has focused on assuring security provided leakage entropy remains below a certain bound; unfortunately, no practical means of (provably) verifying such a bound for a given device is currently available.

We extend work on Yao circuits [28,29,15,1,5,6,14], especially their implementation on tokens [8,7] as a means of performing leakage-resilient computation within the motivating scenario above. Our focus is practicality: we aim to keep the entire architecture as simple as possible in order to reduce manufacturing and verification cost. Careful analysis of said architecture places a bound τ on the number of useful leakage occurrences an attacker can observe. For a given signal-to-noise ratio, this forces an attacker to develop more effective attacks rather than simply using more traces; when combined with conventional countermeasures, it can effectively prevent such attacks. One can view this process as playing a similar role to key refresh [18,17], but without the need for synchronization. In addition, we add the first secure authentication payload, HMAC [20], to the list of applications implemented as Yao circuits. The overall result are leakage-*bounded* implementations of both AES_{128} and HMAC.

2 Background

An important aspect of formalising the ability of a side-channel attack(er) is the selection of a model that describes the form of leakage from a token (and thus exploitable by the attacker). The model proposed by Standaert et al. [24] can be directly applied to our scenario with just one minor modification. In said model, adversarial success depends, among other factors, on the number of oracle queries allowed per primitive independent of updates to secret values (e.g., use of key refresh schemes). We therefore replace the number of oracle queries with the number of observable leakage occurrences per secret value.

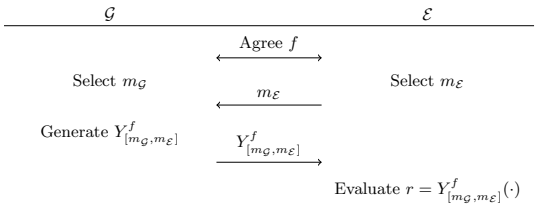


Fig. 1. A high-level, generic description of Yao circuit generation and evaluation. Note that all inputs and outputs are implicitly defined wrt. f, and that depending on the scenario a) one or more of f, $m_{\mathcal{G}}$ and $m_{\mathcal{E}}$ could alternatively be provided as input to the protocol, and b) subsequent use of r could be included.

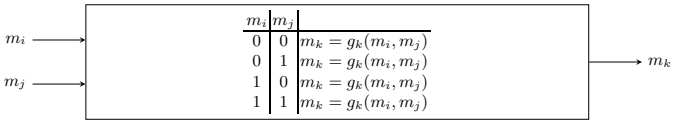
2.1 Yao Circuits at a High Level

An implementation of a functionality f as a standard Boolean circuit, say B^f, can be evaluated using an input x to give an output $r = B^f(x)$. We use B^f_x to denote an implementation with an embedded fixed input and retain a similar form for evaluation $r = B^f_x(\cdot)$. At a high level, Yao circuits simply represent a non-standard implementation of f, say Y^f, which allows the associated evaluation to be secure.

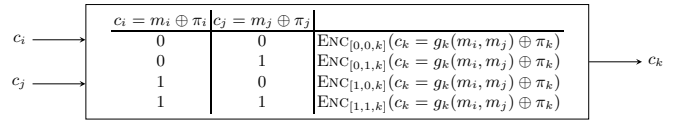
Use of a given Yao circuit can be described formally as a secure two-party computation protocol. The parties involved are a circuit generator $\mathcal{G}$ who (given f and x) produces Y^f_x, and a circuit evaluator $\mathcal{E}$ who (given Y^f_x) computes $r = Y^f_x(\cdot)$; both are illustrated in Figure 1. Thus the side-channel attack surface is shifted away from individual primitives f onto the task of generating a Yao circuit $Y^f_{[m_{\mathcal{G}},m_{\mathcal{E}}]}$ where we can give strong bounds on leakage. In contrast to the original usage as a two-party computation protocol, the token is trusted and we do not need oblivious transfer to communicate the circuit inputs $m_{\mathcal{E}}, m_{\mathcal{G}}$. Note that in theory $\mathcal{G}$ could evaluate $Y^f_{[m_{\mathcal{G}},m_{\mathcal{E}}]}$ as well as generating it, but we deem it more economic in most cases to let $\mathcal{E}$ do the evaluation.

Imagine g_k refers to some k-th truth table wlog. describing a 2-input, 1-output Boolean function or gate instance within B^f. Both the inputs to and outputs from said gate are provided on wires indexed using a unique wire identifier (or wire ID): we write m_i for the value carried by the wire with wire ID i. Note that the output wire ID can act to identify a given gate instance (i.e., acts as a gate ID). Figure 2a is a trivial starting point outlining how such a gate can be evaluated.

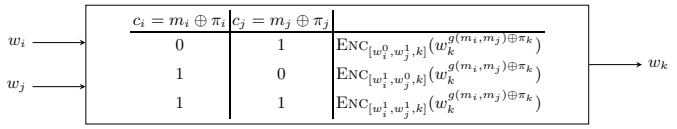
Yao circuits are constructed by forming a "garbled" Yao gate instance in Y^f for each Boolean gate instance in B^f. Both inputs to and outputs from the Yao gate are altered to mask their underlying value: this means each m_i is replaced by c_i, an encryption of the former. Given $\textsc{Enc}_x(y)$ denotes the encryption of y under the key x using some symmetric cipher (with a κ-bit key and β-bit block size), Figure 2b illustrates a Yao gate corresponding to the above. Note that



(a) An example Boolean gate.



(b) An illustrative example of the corresponding Yao gate.



(c) The corresponding Yao gate with wire labelling and Garbled Row Reduction (GRR) [19] applied (noting that each c_i forms part of the associated w_i per Equation 1).

Fig. 2. A step-by-step comparison of a 2-input, 1-output Boolean gate (whose function is described by g_k), and the associated Yao gate construction before and after optimisation

- the public c_i and c_j inputs (whose secret underlying values are m_i and m_j) are provided on wires with public indices i and j,
- the public c_k output (whose secret underlying value is m_k) is provided on a wire with public index k,
- the standard gate functionality is g_k, and
- π_i, π_j and π_k act as secret permutation bits on the rows of the truth table.

During evaluation of the gate, $\mathcal{E}$ gets c_i and c_j meaning it cannot recover the underlying values of m_i and m_j since it does not know π_i and π_j. However, c_i and c_j index *one* entry of the truth table and allow *only* this entry to be decrypted (since they determine the associated cipher key) and yield c_k. The central idea is that a Yao gate reveals nothing about a) the gate functionality nor b) underlying values, iff. it is evaluated at most once.

2.2 Abstract Realisation of Yao Circuits

Wire Labels. The illustrative example above has a major shortcoming: the effective cipher key size is just two bits (since all wire IDs are public), meaning the key is inherently susceptible to exhaustive search. To combat this, given a security parameter λ one replaces the Boolean value communicated on each wire with a randomised λ-bit wire label. Let

$$w_i^{c_i} = \rho_i \parallel (m_i \oplus \pi_i) = \rho_i \parallel c_i \tag{1}$$

denote the i-th such wire label in the general case where $m_i \oplus \pi_i = c_i$ for $c_i \in \{0, 1\}$, $\rho_i \xleftarrow{\$} \mathbb{Z}_2^{\lambda-1}$ and $\pi_i \xleftarrow{\$} \mathbb{Z}_2$. Note that the combination of ρ_i and π_i

could be viewed as a λ-bit ephemeral key, implying $\kappa = 2\lambda + \epsilon$ where ϵ is the number of bits used to encode wire IDs, and that Equation 1 caters for two optimisations outlined below.

The Garbled Row Reduction (GRR) optimisation was introduced by Naor et al. [19]. In short, by carefully selecting

$$w_k^{c_k} = w_k^{g(m_i,m_j)\oplus\pi_k} = \text{ENC}_{[w_i^0,w_j^0,k]}(\gamma) \tag{2}$$

for $c_i = c_j = 0$ and a suitable public constant γ, the first truth table entry can be eliminated (as illustrated by Figure 2c). This is attractive since it permits up to a 25% reduction in communication of the Yao circuit from $\mathcal{G}$ to $\mathcal{E}$, plus reduces the amount of storage required.

The "free XOR" optimisation introduced by Kolesnikov and Schneider [11] realises XOR gates (almost) for free. Given a global (i.e., one for each instance of Y^f), secret constant $\Delta \in \mathbb{Z}_2^{\lambda-1}$ they select w_i^1 as in Equation 1, then define

$$w_i^0 = w_i^1 \oplus (\Delta \parallel 1) = (\rho_i \oplus \Delta) \parallel (\pi_i \oplus (m_i \oplus 1)) \tag{3}$$

to allow computation of XOR gates via the relationships

$$\begin{aligned} w_i^0 \oplus w_i^1 &= w_i^1 \oplus (\Delta \parallel 1) \oplus w_j^1 &&= w_i^1 \oplus w_j^0 = w_k^1 \\ w_i^0 \oplus w_i^0 &= w_i^1 \oplus (\Delta \parallel 1) \oplus w_j^1 \oplus (\Delta \parallel 1) &&= w_i^1 \oplus w_j^1 = w_k^0 = w_k^1 \oplus (\Delta \parallel 1) \end{aligned}$$

2.3 Previous Token Implementations of Yao Circuits

As far as cryptographic primitives are concerned, previous work (with the exception of [12]) focus on use of AES_{128} as the functionality f. Other functionalities considered relate instead to higher level applications, e.g., database search [5,14] or bioinformatics [6]. Pinkas et al. [21] provide the first feasibility results (using software) of Yao-based constructions; since they relate more directly to the chosen scenario, we detail work by Järvinen et al. [8,7] below which both implement Yao circuits on tokens but do not use modularisation.

Secure computation via One-Time Programs (OTPs): In [8], Järvinen et al. consider a scenario wherein $\mathcal{E}$ is a hardware token, and $\mathcal{G}$ is a trusted party during a setup stage. The idea is that $\mathcal{G}$ as token issuer stores One-Time Memories (OTMs) for a fixed number x of OTPs represented by Y^f on the token. At run-time, the token owner uses one set of OTMs to form the input labels corresponding to his data, and the token evaluates the Yao circuit before finally releasing the result.

The advantage of this scenario is that very little protection against side-channel attacks is required. However, the major disadvantage is the limited number of Yao circuits: in most real-life scenarios this is unacceptable. In addition, a generic framework without this disadvantage is given at the cost of loosing the leakage-resilient circuit generation. Our work can be seen as a leakage-resilient, more flexible version of the framework.

Secure computation via out-sourcing: In [7], Järvinen et al. consider a scenario wherein $\mathcal{E}$ is a cloud computing provider; the role of $\mathcal{G}$ is split between a secure

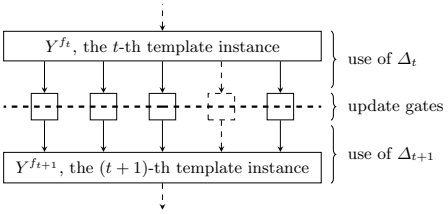


Fig. 3. An example of updating Δ between two template instances, with the heavy dashed line denoting the boundary between use of Δ_t and Δ_{t+1}

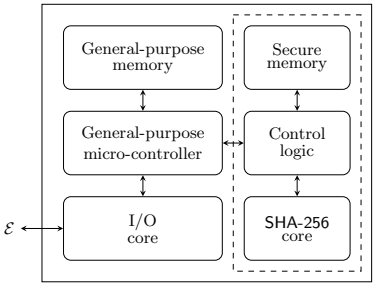


Fig. 4. A block diagram of our proposed token architecture. Only the Yao core (within the dashed box) has requirements for conventional side channel countermeasures

hardware token $\mathcal{G}_S$ and some other party $\mathcal{G}_U$ (e.g., a desktop workstation). The idea is for $\mathcal{G}_U$ to generate B^f, which is passed to $\mathcal{G}_S$ and translated (securely) into Y^f. The Y^f can then be evaluated by $\mathcal{E}$, with the overall effect of securely out-sourcing computation from $\mathcal{G}_U$ to $\mathcal{E}$ (and the token $\mathcal{G}_S$).

This scenario is advantageous in the sense it allows a flexible choice of f (wrt. the token) and is very speed- and memory-efficient. However, it has a relatively high hardware requirement: in addition to the SHA-256 core it requires at least one AES_{128} core, [7] uses two, which all have to be free from leakage.

3 Supporting Alterations to Traditional Yao Circuits

To support our design in Section 4, we first outline two supporting concepts: detail relating to their realisation and utility is deferred until later. While neither represents a significant change to underlying theory, we posit that both significantly ease the practical task of constructing and using Yao circuits.

3.1 Circuit Modularisation

Existing Limitations. Consider a typical iterative block cipher design, with s rounds in total (e.g, AES_{128} with $s = 10$). The functionality for round i is described by f_i, which implemented as a Boolean circuit is B^{f_i}. The s different round functionalities can be the same or different as required, with the overall cipher thus described by the functionality

$$f = f_{s-1} \circ \cdots \circ f_1 \circ f_0.$$

One can implement this either combinatorially, whereby instances of each B^{f_i} are "unrolled" to form the resulting monolithic circuit, or iteratively, whereby instances of each unique B^{f_i} are "looped over" with a need for only one circuit instance and some control logic. Traditional Yao circuits *must* adopt the former

approach: no sequential elements (e.g., latches, clock signals) are allowed because each gate can be evaluated at most once (to ensure security). One cannot reuse the resulting Yao circuit unless rerandomisable constructions [4] are considered; typically these incur a prohibitive overhead.

One impact of this restriction is that previous work almost exclusively focuses on AES_{128} (as far as cryptographic primitives are concerned) which a) has a fixed s and can hence be unrolled, and b) has a fairly compact hardware implementation. We know of only one implementation of another cryptographic primitive [12], wherein an (insecure) implementation of 256-bit RSA is described. Even with such small operands, the resulting Yao circuit is ≈ 8500 times larger than their AES_{128} circuit, in part as a result of the requirement to unroll the loop representing a binary, modular exponentiation.

Modularised Yao Circuits. To address this issue, we make use of modular Yao circuits. Similar concepts have been used recently[2] in [6,14] to achieve efficiency gains for large circuits, but only [14] mentions the possibility of using run-time parameters to control circuit assembly.

Traditionally (right) each monolithic Boolean circuit $B^f = B^{f_{s-1} \circ \cdots \circ f_1 \circ f_0}$ is stored and must be translated into the corresponding Yao circuit by $\mathcal{G}$ each time the latter needs to be evaluated. In our alternative (left), $\mathcal{G}$ holds only the static description of each template B^{f_i} (and associated meta-data), instantiating them to form Y^f without holding the entirety of the (potentially large) B^f. This technique permits a quasi-loop: given s, e.g. the number of blocks to be hashed, at run-time (rather than being fixed), the token unrolls one circuit template B^{f_i}, to form the resulting Yao circuit. The resulting reduction in resource requirement and increased flexibility allows us to implement HMAC. Generation of the corresponding Y^f can be streamed in that $\mathcal{G}$ communicates one part at a time to $\mathcal{E}$.

3.2 Updating Δ between Template Instances

In previous works [21,8,7] using the free XOR trick [11], the authors argue that for correctness Δ must remain constant within a given Yao circuit. Indeed, if an XOR gate is evaluated using constants $\Delta_1 \neq \Delta_2$, the result is incorrect as

$$\begin{aligned} w_i^0 \oplus w_j^1 &= (w_i^1 \oplus (\Delta_1 \parallel 1)) \oplus w_j^1 \neq w_i^1 \oplus (w_j^1 \oplus (\Delta_2 \parallel 1)) = w_i^1 \oplus w_j^0 \\ w_i^0 \oplus w_j^0 &= (w_i^1 \oplus (\Delta_1 \parallel 1)) \oplus (w_j^1 \oplus (\Delta_2 \parallel 1)) \neq \qquad\qquad w_i^1 \oplus w_j^1 \end{aligned}$$

shows. Despite this, a crucial observation is that Δ *can* be changed *if* said change is applied consistently. Although doing so has no functional benefit, we use it to directly formulate a leakage bound within Section 5.

In some modularised Yao circuit, imagine the t-th template instance uses Δ_t. The next, $(t+1)$-th instance can then use Δ_{t+1} (as illustrated by Figure 3) iff.

[2] A year earlier, [5] proposed a different kind of modularity, namely mixing Yao circuits and homomorphic operations. The cost associated with additional hardware required to support homomorphic operations means we do not adopt this approach.

each connecting wire is updated appropriately. The simplest approach is to consider a dedicated 1-input, 1-output update gate with the identity functionality, i.e., $g_k(x) = x$. Given

$$w_k^{c_k} = w_k^{m_i \oplus \pi_k} = \textsc{Enc}_{[w_i^0, w_i^0, k]}(\gamma)$$

by definition, and that $m_i = m_k$ since the gate updates Δ rather than change the underlying input value, Equation 2 means the associated wire labels are

$$\begin{aligned} w_i^0 &= w_i^1 \oplus (\ \Delta_t \ \ \|\ 1\ ) \\ w_k^0 &= w_k^1 \oplus (\ \Delta_{t+1} \ \|\ 1\ ) \end{aligned}$$

However, this suggests that *any* non-XOR gate can be used to perform the update without cost: the existing GRR-optimised Yao gate truth table only needs to have Δ_{t+1} folded into the label $w_k^{g_k(m_i, m_j) \oplus \pi_k}$ instead of Δ_t where appropriate.

4 Token Design

We consider a scenario wherein $\mathcal{G}$ is a hardware token that needs to be secured against SPA and DPA attacks, and $\mathcal{E}$ is some other party to which computation is outsourced. The idea is to put a bound on the number of times secret values are used for any computation and leakage can be observed before the value – akin to key refresh – is updated. The known, residual leakage can then be accommodated by careful use of conventional countermeasures.

4.1 Cipher Construction

To instantiate the symmetric cipher required to encrypt wire labels, we follow existing work [21,7,8] by using a one-time pad like construction

$$\textsc{Enc}_{[w_i^{c_i}, w_j^{c_j}, k]}(w_k^{c_k}) = \mathsf{SHA\text{-}256}(w_i^{c_i} \ \|\ w_j^{c_j} \ \|\ k) \oplus w_k^{c_k}$$

where the SHA-256 output is implicitly truncated to λ bits to match the wire label size. We reuse SHA-256 as a secure Pseudo-Random Number Generator (PRNG), splitting the SHA-256 into two 128-bit values

$$\begin{aligned} [x, y] &\leftarrow \mathsf{SHA\text{-}256}(seed_{t-1}) \\ seed_t &\leftarrow seed_{t-1} \oplus y \\ rand &\leftarrow x \end{aligned}$$

so $x = rand$ is used as a wire label for example, while y is used exclusively to update the seed. Note SHA-256 is therefore the *only* significant cryptographic core required by the token. This construction is certainly secure if the PRNG is modelled as a random oracle which is a weaker model than the one we have for our Yao circuits. Intuitively however, some form of correlation robustness should be sufficient. Research on the correlation robustness of hash functions is

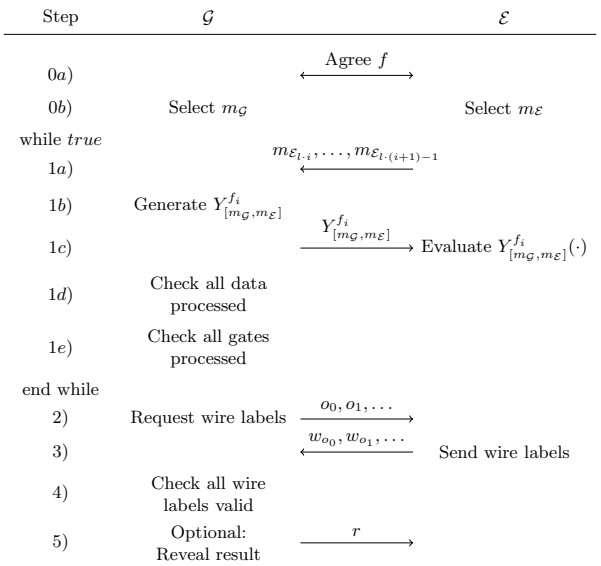


Fig. 5. The two-party computation protocol reflecting circuit modularisation. Note that $\mathcal{E}$ does not communicate $m_{\mathcal{E}}$ in one block, but rather in multiple l-bit blocks. Output wires of the Yao circuit (i.e. the wires carrying results from the functionality) have wire IDs $o_0, o_1, \ldots$.

still developing, see [11,3] for example, and therefore we defer the exact security requirements to future work.

Previous work has used domain-specific languages (e.g. SFDL [15,1]) to implement the payloads. While this *may* ease implementation, it reduces control over the actual circuit layout. Standard Hardware Description Languages (HDLs) are, in fact, well suited to payload description: they simply lack the protocol aspect of Yao circuits. Therefore we aim to reap benefits of familiarity, design and code portability by using a VHDL dialect and associated compiler, both of our own design, to describe structural and behavioural circuit templates.

4.2 Operational Protocol and Token Architecture

The communication between the token $\mathcal{G}$ and the evaluator $\mathcal{E}$ is shown in Figure 5. $\mathcal{E}$ can, for example, be a local untrusted work station or an untrusted but more powerful chip within a mobile phone; in most cases it will not be the authentication partner. We assume a physical connection between the parties, and hence focus on optimising their workload rather than the number of communication rounds.

Initially, in step 0, $\mathcal{E}$ requests a functionality f (e.g., HMAC or AES_{128}) and both parties need to have (or generate) corresponding inputs $m_{\mathcal{G}}$ and $m_{\mathcal{E}}$.

Step 1, from a theoretic perspective, is the same as the monolithic communication in Figure 1 despite now supporting the modularised approach. Specifically, $\mathcal{G}$ generates the Yao circuit Y^f based on the circuit templates B^{f_i} and sends it step-by-step to be evaluated. Note that modularisation forces three important checks:

1. step 1d checks if all input values (from both $m_{\mathcal{G}}$ and $m_{\mathcal{E}}$) have been used,
2. step 1e checks if all gates have evaluated, and
3. step 4 checks if all output values are valid, i.e., if $w_{o_j} \in \{w_{o_j}^0, w_{o_j}^1\} \; \forall \; o_j$.

When a check condition can not be satisfied the token aborts immediately, meaning in particular that it does not reach step 5 where the result $r = f(m_{\mathcal{G}}, m_{\mathcal{E}}) = Y^f_{[m_{\mathcal{G}}, m_{\mathcal{E}}]}$ is revealed, and that *seed* values of the PRNG are not accidentally reused.

To support the protocol outlined above, Figure 4 outlines a proposed token architecture. The main components and their roles are as follows:

- A general-purpose micro-controller manages the communication protocol in Figure 5, controlling assembly of Y^f from the B^{f_i} circuit templates and performing other tasks (such as message padding for HMAC).
- A general-purpose memory holds the B^{f_i}, micro-controller program and other public run-time variables values which only require correctness.
- Storage of and computation using secret values is limited to the Yao core, which consists of:
  - A SHA-256 core, used to encrypt wire labels and also as a PRNG.
  - Control logic used for auxiliary operations such as wire label generation.
  - A secure memory, split into two parts: a non-volatile part holds $m_{\mathcal{G}}$ and *seed*, while a volatile part holds Δ_t, Δ_{t+1} and, for each wire i, the tuple $\{w_i^0, w_i^1\}$. Note that $\Delta_{t+1} = 0$ unless the token is processing update gates, and that a crucial role of the secure memory as a whole is to prevent read-out or other leaks of values such as $m_{\mathcal{G}}$ and *seed*.

5 Analysis and Results

5.1 Security Analysis

This section attempts to explore security aspects of the proposal outlined in Section 4. After a statement of general assumptions, we deal specifically with potential attack vectors exploited during an SPA or DPA attack, or by a malicious adversary within the operational protocol.

Security Assumptions. To be successful, the adversary has to recover the input $x_{\mathcal{G}}$ held by the token for which he can either attempt to recover $x_{\mathcal{G}}$ directly (e.g., via a DPA attack), or try to "ungarble" the Yao circuit $Y_{x_{\mathcal{G}}, x_{\mathcal{E}}^f}$ (or part thereof). In showing neither strategy is viable, we make some important assumptions:

A-1 An authentication protocol that prevents man-in-the-middle attacks against $m_{\mathcal{E}}$ in step $1d$ of the Yao protocol must be selected (if this threat is relevant).
A-2 The control-flow of the token, managed by the micro-controller, is tamper-proof which implies integrity of the general-purpose memory. Although this is a strong requirement, it is common for embedded systems.
A-3 The hash function used for encryption of wire labels (in our case SHA-256) must be circular-2-correlation robust (see Choi et al. [3]).
A-4 The token cannot be reset, and no randomness reused: this prevents an adversary forcing the token to regenerate the same Yao circuit with the same randomness, then reevaluating it with different inputs.

Power Analysis Adversaries. SPA attacks attempt to recover the target value using one (or at least very few) traces and attack capabilities do not scale over time; examination of data-dependent control-flow is one example. There are two possible attack vectors:

SPA-1 For each i-th input wire, the token must send either w_i^0 or w_i^1 to the evaluator depending on the underlying value of m_i. To succeed, the adversary must be able to determine whether m_i is 0 or 1 (for all $m_i \in x_{\mathcal{G}}$).
SPA-2 Gates such as AND and OR are biased towards 0 or 1 in their output: if the adversary determines during computation of

$$\textsc{Enc}_{[w_i^{c_i}, w_j^{c_j}, k]}\left(w_k^{g(m_i,m_j)\oplus\pi_k}\right)$$

which truth table row contains the minority output, they can reverse the permutations (i.e., π_i and π_j) and recover the underlying values of almost all output wire labels from non-XOR gates.

For hardware implementations, both SPA attack vectors will be implemented using multiplexers whose data dependency of the power consumption is usually already hidden well enough without countermeasures (e.g., [16, Appendix A.3]). Even if this is not the case, traditional countermeasures (e.g., random masking of the select signal with corresponding permutation of the inputs) are efficient.

In contrast, DPA attacks attempt to recover the security-critical target value by applying statistical distinguishers to a large set of traces; issues of signal-to-noise ratio, as well as explicit countermeasures, determine the exact number. More formally, let k be the target value, v be a variable value and r the result of some generic operation $\odot$. A DPA adversary collects traces relating to execution of $r_i \leftarrow k \odot v_i$ for many different $v_1, v_2, \ldots, v_\sigma$. The potency of a DPA attack is then judged by σ, the number of traces required to be reliably recover k. Our approach is to have a design-time constant bound $\tau \ll \sigma$ instead of allowing the adversary to control it. Put another way, we bound the leakage such an adversary can utilise in a DPA attack: if the application of conventional countermeasures can prevent attacks with said leakage level, the token is secure.

DPA-1 The token must compute $w_i^0 \leftarrow w_i^1 \oplus (\Delta_t \parallel 1)$ for every wire. As such we have

$$\tau_{\textbf{DPA-1}} = \max(\delta_1, \delta_2, \ldots)$$

where δ_t denotes the number of wires using Δ_t. If the technique in Section 3.2 is used correctly, $\tau_{\textbf{DPA-1}}$ is a constant determined by the token designer.

DPA-2 For each gate, the four values of $w_{\{i,j\}}^{\{0,1\}}$ are each used twice as input to $\mathsf{SHA\text{-}256}(w_i^{\{0,1\}} \parallel w_j^{\{0,1\}} \parallel k)$. Focusing on one label, wlog. w_i say, and one external value, wlog. 0 say, the attacker gets two traces for each gate where w_i^0 is used as input. Therefore, we have

$$\tau_{\textbf{DPA-2}} = 2 \cdot \max_{\forall k}(G_k)$$

where G_k represents the fan-out of the k-th gate (and input wires are also considered as being driven by imaginary gates). Note that a similar attack vector exists when the token processes an XOR gate. Such a gate must compute w_k^0 and w_k^1, and one possible approach is to compute

$$\begin{aligned} w_k^0 &\leftarrow w_i^0 \oplus w_j^0 \\ w_k^1 &\leftarrow w_i^0 \oplus w_j^1 \end{aligned} \tag{4}$$

in which case $\tau_{\textbf{DPA-2}}$ conveniently covers this attack vector as well.

Concrete, non-optimised examples for these bounds are given in Section 5.2. If our design is used to protect against DPA attacks, functionalities that were inherently secure against SPA clearly inherit any SPA vulnerabilities of the underlying Yao circuit approach. We suggest that preventing SPA attacks on our design using conventional countermeasures is, broadly speaking, easier than preventing DPA attacks on the functionality in question: the cost of preventing the former is easily justified by the improvement offered wrt. the latter.

Catering for Timing Analysis Adversaries. The execution time associated with generating of a Yao circuit is inherently independent of the inputs to that Yao circuit: it depends *only* on the circuit size. As far as the architecture is concerned, we do not use a cache for the micro-processor in order to avoid cache-based timing attacks. Working without a cache is a common decision for cryptographic tokens and therefore not an exceptional burden of our design.

Catering for Malicious Adversaries. One advantage of Yao circuits is the availability of related security proofs. For semi-honest adversaries, Figure 5 preserves proofs already given by [11,21,7,8,3]. This is a direct result of steps 1*a* to 1*e* being equivalent to the single generate-evaluate step from previous protocols.

However, we also need to consider malicious adversaries. Lindell et al. [13] show how a two-party computation protocol using Yao circuits can cover the case of malicious $\mathcal{G}$ using a cut-and-choose approach. Our scenario is far less complex, since Y^f is generated by the token which is implicitly trusted: we disallow a malicious $\mathcal{G}$. Therefore we only have to consider a malicious $\mathcal{E}$, and show it cannot learn anything about $x_{\mathcal{G}}$ not implied by the result $r = f(x_{\mathcal{G}}, x_{\mathcal{E}})$ by deviating from the protocol. $\mathcal{E}$ has two options (which we expand on below): it can either a) provide faulty input or b) perform variants on early termination.

Faulty data. The only steps where $\mathcal{E}$ can provide faulty data are 1a and 3. As $m_{\mathcal{E}}$ in step 1a is the input of the functionality, sending a faulty $m_{\mathcal{E}}$ has no impact on the security of the Yao protocol: it can only influence the output of the functionality f. In contrast, sending faulty w_{o_j} in step 3 is potentially a problem if the adversary sends labels from intermediate wires instead of the output labels. However, this is prevented by the check in step 4 which ensures that for each output wire o_j, exactly one wire label $w_{o_j}^{c_{o_j}} \in \{w_{o_j}^0, w_{o_j}^1\}$ has been sent before the result is revealed.

Early termination. Since $\mathcal{E}$ can not learn anything from one of the partial circuits (i.e., a given $Y_{[m_{\mathcal{G}}, m_{\mathcal{E}}]}^{f_i}$) until the protocol is finished (i.e., until $\mathcal{G}$ reveals the result), $\mathcal{E}$ cannot profit from straight early termination. However, if the functionality f requires s iterations of a loop to form $f = f_{s-1} \circ \cdots \circ f_1 \circ f_0$, per the description of AES_{128} in Section 3.1 for instance, the adversary could potentially gain information from terminating the loop early, i.e., to get $f' = f_{s'-1} \circ \cdots \circ f_1 \circ f_0$ for $0 < s' < s$: this would be analogous to a reduced-round attack. To prevent this, we require the token to check (in steps 1d and 1c) whether the Yao circuit for f has been completely generated or whether some $Y_{[m_{\mathcal{G}}, m_{\mathcal{E}}]}^{f_i}$ is missing.

5.2 Experimental Results and Analysis

Our goal is to study gross, indicative metrics and trade-offs rather than focus on absolute figures that could be improved via incremental optimisation. For the evaluation of our proposed design, we implemented a VHDL compiler (per Section 4.1), a token simulator $\mathcal{G}$, an evaluator $\mathcal{E}$ as well as two payload functionalities, namely AES_{128} and $\mathsf{HMAC}_{\mathsf{SHA\text{-}256}}$. For each payload, we considered variants that differ in their frequency of Δ update: for AES_{128} three variants are used, for $\mathsf{HMAC}_{\mathsf{SHA\text{-}256}}$ two variants. The variants are as follows:

`AES` Baseline AES_{128} implementation without updating of Δ.
`AES_U1` AES_{128} with a Δ update after the fifth iteration of the round function.
`AES_U9` AES_{128} with a Δ update after every iteration of the round function.
`HMAC` Baseline HMAC implementation without updating of Δ.
`HMAC_U` HMAC with Δ updates after every iteration of the compression function.

Table 1. Efficiency metrics and leakage bounds for our token design and a range of payload implementations. The block size for AES_{128} is 128 bits, for HMAC it is 512 bits (including the padding in the last block).

	#blocks	#Δ	#XOR	#non-XOR	#SHA-256	RAM	$\lvert B^{f_i} \rvert$	$\lvert m_{\mathcal{G},sec} \rvert$	$\lvert m_{\mathcal{G},pub} \rvert$	$\tau_{\mathbf{DPA\text{-}1}}$	$\tau_{\mathbf{DPA\text{-}2}}$
`AES`	1	1	19088	5760	24578	245.9kB	12318B	176B	–	7296	11
`AES_U1`	1	2	19088	5888	25091	263.4kB	13628B	176B	–	3776	11
`AES_U9`	1	10	19088	6912	29195	262.3kB	12845B	176B	–	960	11
`HMAC`	1	1	148080	129680	556866	1883.6kB	121942B	64B	32B	167824	19
	2	1	222120	194520	835170	2489.8kB	121942B	64B	32B	251608	19
	3	1	296160	260384	1113474	3069.0kB	121942B	64B	32B	335392	19
	4	1	370200	324200	1391778	3671.4kB	121942B	64B	32B	419176	19
`HMAC_U`	1	3	148080	130192	558916	1911.0kB	122981B	64B	32B	84040	19
	2	4	222120	195288	835170	2500.8kB	122981B	64B	32B	84040	19
	3	5	296160	260384	1117574	3113.4kB	122981B	64B	32B	84040	19
	4	6	370200	325480	1396903	3724.6kB	122981B	64B	32B	84040	19

Table 1 details efficiency metrics for implementation of these variants on the platform described and shows the two associated bounds $\tau_{\textbf{DPA-1}}$ and $\tau_{\textbf{DPA-2}}$. The first three columns specify the payload, the number of input blocks from $\mathcal{E}$ and the number of Δ values being used at run-time.

Efficiency. The columns #XOR and #non-XOR in Table 1 give the number of gates in the resultant Yao circuit. Compared to [21,8] we have considerably smaller AES_{128} circuits, which is mainly due to omission of key scheduling and, to a less extent, use of more optimised S-box formulas of Boyar et al. [2]. The omission of key scheduling implies a small penalty of having to store all round keys $m_{\mathcal{G},sec}$ in secure ROM.

The column #SHA-256 shows the number of distinct uses of the SHA-256 core, each a one-block hash. Ignoring the absolute simulation time, we feel this metric best represents the execution time of a concrete token since the SHA-256 core will most likely be the throughput bottleneck. [7] use a SHA-256 core which requires 67 cycles per 512 bit block at 66 MHz. Based on these numbers, a crude time estimation (based only on calls to the SHA-256 core) is 24ms for `AES` and 1418ms for `HMAC_U` with 4 message blocks.

A significant issue is the amount of RAM required at run-time. To assess this, we measured the simulator heap and stack usage using the Valgrind `massif` tool [27]. We note that the tool itself is not perfect, and that the result includes overhead of up to 20% relating to performance and security counters. Even so, the indicative RAM requirement is large: it remains within the capability of devices in our remit, but clearly beyond smart-cards or RFID tokens for example. The requirement stems in the most part from storing *all* wire labels $\{w_i^0, w_i^1\}$ in RAM. One possible trade-off would be to store only w_i^0 and recompute w_i^1 when needed, reducing the RAM usage by a factor of two, but increasing the number of traces available by a factor similar to the maximum fan-out. [7] chose a keyed PRNG which allows recomputation of w_i^0 when needed, thus reducing the RAM requirements drastically. However, any keyed PRNG is vulnerable to DPA attacks with unlimited τ which negates our aim of bounding the leakage.

An interesting observation can be made about the RAM usage of `AES_U1` and `AES_U9`. Intuitively, one would expect the RAM usage to always grow in line with the number of Δ_t used. In this case, the opposite happens because `AES_U1` applies the Δ updating within the top-level entity (which also accounts for the larger $|B^{f_i}|$), requiring more wires for which RAM is allocated during the entire run-time. `AES_U9` performs the updating at the end of the round function entity instead, and the RAM for additional wires can be deallocated as soon as each round function instance of has been completed.

The size of the templates, $|B^{f_i}|$ (stored in unsecured ROM), profits directly from modularisation. As predicted, the size of $|B^{f_i}|$ for HMAC does not depend on the message size as it would have for the traditional approach.

Security. Having explained $\tau_{\textbf{DPA-1}}$ and $\tau_{\textbf{DPA-2}}$ in Section 5.1, we note that our Δ updating technique limits $\tau_{\textbf{DPA-1}}$ as predicted; note esp. the `HMAC_U` payload, where updating Δ fixes previously unlimited leakage to a constant chosen by the token designer.

The result for $\tau_{\textbf{DPA-2}}$ is an absolute upper bound, i.e., for all output wires we counted how often it gets used while processing the follow-up gates. As explained in Section 5.1, if a wire is used as input to a non-XOR gate each label gets used twice; for XOR gates Equation 4 gives the numbers relevant to our implementation. For an attacker it will be very difficult to combine traces from two different operations like this but we prefer to err on the side of security by overestimating the attacker. With numbers this low, **DPA-2** is almost irrelevant as an attack vector. But having a low $\tau_{\textbf{DPA-2}}$ was an explicit aim of our work: $\tau_{\textbf{DPA-2}}$ is the only possible attack vector on the SHA-256 core, and therefore $\tau_{\textbf{DPA-2}}$ is crucial to determine the level of conventional countermeasures needed to protect the SHA-256 core. Compared to the SHA-256 core, protecting the XOR from **DPA-1** to match a much higher $\tau_{\textbf{DPA-1}}$ is inexpensive.

As a reference one may look at the Power-Trust micro-processor of Tillich et al. [26], which has parts of the ALU implemented within a secure zone. For evaluation purposes they implemented the secure zone in three different logic styles (namely CMOS, iMDPL [22] and DWDDL [30]) and performed a DPA attack against an AES_{128} software implementation using the secure zone. While it is difficult to directly extrapolate from a design as different from ours, this at least gives an estimate: there is no reason why secure logic styles such as iMDPL and DWDDL should fare worse for our token. For the DPA attack on the secure zone to be successful, Tillich et al. required 130,000 traces against the (unprotected) CMOS implementation, 260,000 traces against the iMDPL implementation and 675,000 traces against the DWDDL implementation. With $\tau_{\textbf{DPA-1}} = 7296$ in the worst case for AES_{128} and $\tau_{\textbf{DPA-1}} = 84040$ for `HMAC_U` we surmise that both iMDPL and DWDDL would have successfully thwarted the DPA attack from Tillich et al. against an implementation of our token. It is important to note, that for both bounds we did not yet try to find the absolute minimum. For example it is possible to add additional gates to achieve fan-out = 2 and thus $\tau_{\textbf{DPA-2}} \leq 4$ while $\tau_{\textbf{DPA-1}}$ can be easily reduced by updating Δ more often within the round resp. compression functions, not just at their end.

6 Conclusions

In essence, this paper has demonstrated that an embedded token can be designed which gives strong bounds on the number of useful traces a power analysis adversary can collect. Our design methodology

1. is generic in that it works for all payloads and use-cases (cf. PIN block),
2. does not impose limits on the token lifetime,
3. does not require synchronization (cf. key update schemes),
4. is easily verifiable, and
5. successfully limits attack capabilities for side-channel adversaries.

In relation to the former point, we have already extended previous work through support for a Yao circuit for HMAC. Exploration of further primitives based on modularisation (including methods and trade-offs to further reduce the leakage bound), plus incremental optimisation of both the token design and operational

protocol (especially the RAM requirement) are interesting avenues for further work. In relation to the latter point, the clear next step is to produce experimental results from a concrete implementation of the token. This would, for example, allow investigation of the concrete leakage and whether implementation specific high order moments occur which increase τ for higher order attacks.

Acknowledgements. The work described in this paper has been supported in part by EPSRC grant EP/H001689/1 and by Academy of Finland, project #138358. We would like to thank Elisabeth Oswald for her valuable comments.

References

1. Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP - A Secure Multi-Party Computation System. In: CCS, pp. 257–266 (2008)
2. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 178–189. Springer, Heidelberg (2010)
3. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.-S.: On the Security of the "Free-XOR" Technique. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 39–53. Springer, Heidelberg (2012)
4. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010)
5. Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-partY computations. In: CCS, pp. 451–462 (2010)
6. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster Secure Two-Party Computation Using Garbled Circuits. In: USENIX Security Symposium (2011)
7. Järvinen, K., Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Embedded SFE: Offloading Server and Network Using Hardware Tokens. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 207–221. Springer, Heidelberg (2010)
8. Järvinen, K., Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 383–397. Springer, Heidelberg (2010)
9. Kocher, P., Lee, R., McGraw, G., Raghunathan, A., Ravi, S.: Security as a New Dimension in Embedded System Design. In: DAC, pp. 753–760 (2004)
10. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
11. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
12. Kreuter, B., Shelat, A., Shen, C.: Billion-gate secure computation with malicious adversaries. In: USENIX Security Symposium (2012)
13. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
14. Malka, L., Katz, J.: VMCrypt – Modular Software Architecture for Scalable Secure Computation. In: CCS, pp. 715–724 (2011)

15. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - A Secure Two-Party Computation System. In: USENIX Security Symposium, pp. 287–302 (2004)
16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (2007)
17. Medwed, M., Petit, C., Regazzoni, F., Renauld, M., Standaert, F.-X.: Fresh rekeying II: Securing multiple parties against side-channel and fault attacks. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 115–132. Springer, Heidelberg (2011)
18. Medwed, M., Standaert, F.-X., Großschädl, J., Regazzoni, F.: Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 279–296. Springer, Heidelberg (2010)
19. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Electronic Commerce, pp. 129–139 (1999)
20. National Institute of Standards and Technology (NIST). The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication 198-1 (July 2008)
21. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
22. Popp, T., Kirschbaum, M., Zefferer, T., Mangard, S.: Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 81–94. Springer, Heidelberg (2007)
23. Ravi, S., Raghunathan, A., Kocher, P.C., Hattangady, S.: Security in Embedded Systems: Design Challenges. TECS 3(3), 461–491 (2004)
24. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
25. Standaert, F.-X., Pereira, O., Yu, Y., Quisquater, J.-J., Yung, M., Oswald, E.: Leakage Resilient Cryptography in Practice. In: Towards Hardware-Intrinsic Security, pp. 99–134 (2010)
26. Tillich, S., Kirschbaum, M., Szekely, A.: Implementation and Evaluation of an SCA-Resistant Embedded Processor. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 151–165. Springer, Heidelberg (2011)
27. Valgrind Project. Massif User Manual, `http://valgrind.org/docs/manual/ms-manual.html`
28. Yao, A.C.: Protocols for secure computations. In: Foundations of Computer Science, pp. 160–164 (1982)
29. Yao, A.C.: How to generate and exchange secrets. In: Foundations of Computer Science, pp. 162–167 (1986)
30. Yu, P., Schaumont, P.: Secure FPGA circuits using controlled placement and routing. In: CODES+ISSS, pp. 45–50 (2007)

Lightweight Authentication Protocol for Low-Cost RFID Tags

Pierre Dusart[1] and Sinaly Traoré[2]

[1] XLIM - UMR CNRS $n°7252$, Faculté des Sciences et Techniques,
123 Avenue Albert THOMAS, 87060 LIMOGES CEDEX, France
pierre.dusart@xlim.fr
[2] Faculté des Sciences et Techniques, USTTB, BP E 32 06 BAMAKO, MALI
sinaic2002@yahoo.fr

Abstract. Providing security in low-cost RFID (Radio Frequency Identification) tag systems is a challenging task because low-cost tags cannot support strong cryptography which needs costly resources. Special lightweight algorithms and protocols need to be designed to take into account the tag constraints. In this paper, we propose a function and a protocol to ensure pre-shared key authentication.

Keywords: RFID, Authentication.

1 Introduction

In the future, optical bar codes based systems will be replaced by Radio Frequency Identification systems. These systems are composed of two parts:

- a RFID tag which replaces the bar code;
- a RFID reader which handles information send from the tag.

The tag consists of a microchip which communicates with a reader through a small integrated antenna. Various external form factors can be used: the tag can look like a sheet of paper, like a plastic card or can be integrated below bar code for backward device's compatibility.

RFID tags offer many advantages over optical bar codes [1]:

- the use of microchip enables a range of functionalities like computing capability or readable/writable storage. The stored data, depending on the capacity of the tag, can be static identification number up to rewritable user data.
- the use of RF antenna enables communication between the reader and the tag without line of sight from a distance of several decimeters [2]. A reader can communicate sequentially with up to hundred tags per second.

To provide further functionalities than bar codes, the tag may require data storage. For example, the price of a product can be stored into the tag [3]. To know the price of a product, the customer can ask directly the tag instead of asking

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 129–144, 2013.

the database server connected with the cash register. With these new features, the adoption of RFID technology is growing: inventory without unpacking [4], prevention of counterfeiting [5], quality chain with environmental sensing [6] are deployed applications. The tag systems can be easily adapted for universal deployment by various industries with low prices.

But a new technology must also take into account problems inherited from legacy systems. For example in a shop, security problems to deal with are:

- an item is changed to another (it means for RFID to substitute a tag for a fake one);
- a price is changed without authorization by a malicious user (it means for RFID, to write a tag), ...

In addition, the privacy problem must be considered in some context i.e. an user must not reveal unintentionally information about himself. It means for RFID, the ability of a tag to reveal its identity only to authenticated partners.

To cope with security and privacy problems, the first idea is to use asymmetric cryptography (e.g. RSA [7]) like in public key infrastructures. Unfortunately tags with strong cryptography [8] and tamper resistant hardware [9] are too expensive for a wide deployment.

Hence a constraint class of cryptography [10], named Lightweight Cryptography, appears.

The aim of this paper is to propose a protocol and its related computational function. Section 2 introduces the system model and the underlying assumptions for our protocol. Then related work is presented in section 3. The protocol environment is described in section 4. Section 5 presents the protocol details and the computational functions. Section 6 provides an analysis of some security constraints and shows that the protocol satisfies the lightweight class. Section 7 illustrates how our protocol behaves against cryptographic attacks.

2 System Model and Assumptions

We consider a system with one RFID tag reading system and several low cost RFID tags. We assume that each tag shares a secret K with the reader, which is shared in a secure manner before the beginning of the communication (e.g. in manufacturing stage). The aim of the communication is to authenticate the tag i.e. find its identity and prove that it belongs to the system (by knowing the same secret).

The tag is passively powered by the reader, thus:

- the communication needs to be short (speed and simplicity of an algorithm are usually qualifying factors);
- the communication can be interrupted at any time if the reader does not supply enough energy to the tag.

For cost reasons, the standard cryptographic primitives (hash function, digital signature, encryption) are not implemented (no enough computation power is

available or too much memory is required). Hence, we need a protocol using primitives with a low complexity. This property which is named "Lightweight property" [10] consists to use basic boolean operations like XOR, AND, ...

The security of protocols needs also a good random number generator [11]. This part can be assumed by the reader environment where the features can be higher and costly (e.g. a computer connected with a tag reading system).

3 Related Work

The RFID technology needs security mechanisms to ensure the tag identity. Hence a tag spoofing, where an attacker replaces the genuine tag by its own creation, is defeated if good authentication mechanisms are used. But classical authentication solutions use cryptographic primitives like AES [12] or hash functions (SHA1 [13] or MD5 [14]) which are not adapted to low cost RFID tags. It is thus necessary to look for new suitable primitives for this specific constraint resources environment. In [15–18], authors suggest some protocol families based on elementary arithmetic (e.g. binary bit addition or modular addition by a power of 2). However in [19], B. Defend *et al.* put in defect XOR and SUBSET protocols given in [15] by learning key sequence. They proved that with few resources, an attacker can recover the session keys of these two protocols. The LMAP, M^2AP and EMAP protocols proposed respectively in [16–18] allow a mutual authentication between the reader and the tag but are also completely broken [20] by key recovery attacks. In [21], the authors proposed a family of protocols, called S-protocols, based on a family of generic random number generators that they introduced in the same paper. They presented a formal proof which guarantees the resistance of the S-protocol against the attacks of desynchronization [22, 23] and impersonation [24]. With a small modification, they proposed the family of S^*-protocols, which not only has the properties of S-protocols but also allows a mutual authentication between the reader and the tag. However authors do not show that their generic functions are compatible with lightweight RFID tags. In [25], Yeh proposes a protocol corrected by Habibi [26], but attacks [27] appear using $O(2^{17})$ off-line evaluations of the main function. Recently, some protocols are also defined in ISO/IEC WD 26167-6. Since they use AES engine [28], they are out of the scope of this paper.

4 Protocol Requirements and Specifications

We want to use a very simple dedicated protocol which uses a non-invertible function h. We provide a protocol in which the tag identity is sent in a secure manner and the tag is authenticated according to a challenge given by the reader. Then the reader shows that it knows a secret key by calculating an answer to the tag challenge.

We present the authentication protocol: the reader needs to verify the identity of the tag. For the verification of the tag identity iD, the RFID reader $\mathcal{R}$ sends

to the tag $\mathcal{T}$ a challenge C. Next, the tag proves its identity iD by computing a response using the common secret K, shared with the reader. We avoid taking $K = 0$ for a maximum security. Denoting by $Auth$ this response, the authentication phase is presented in the following scheme:

- $\mathcal{R} \longrightarrow \mathcal{T} : C = (C_0, C_1, \ldots, C_{15})$ where C_i are bytes randomly chosen.
- $\mathcal{T} \longrightarrow \mathcal{R} : Auth = [iD \oplus h_K(C), h_{iD}(C)]$

To verify, the reader computes $h_K(C)$ using its challenge C and the key K and then it can retrieve the identity of the tag. Next the authentication of the tag can be verified by computing $h_{iD}(C)$ using the result of previous computation and the first challenge. The protocol allows card authentication by the reader. It can be adapted to allow mutual authentication with a slightly modification: a challenge C' (which can be a counter) is sent with the tag response *Auth*. Next the reader should respond with the computation of $h_{K \oplus C'}(C \oplus iD)$.

5 Proposal Description

Our protocol uses a function h that is composed of two sub-functions S and f taking respectively one and two bytes as input. The function h used in the protocol must be lightweight (for low-cost devices) and satisfy some properties:

- must be a like a one-way function (from output, input cannot be retrieved);
- its output must seem to be random;
- its output length must be sufficient to have enough intrinsically security (to avoid replay and exhaustive authentication search).

We define an input size and an output size of 16 bytes for h and the same size for the secret key K. Output size is chosen to be presented in the 16-byte form to iterate an algorithm defined on byte. Function f which processes byte data blocks and a substitution function S are described in the following subsections.

5.1 Function Design

***f* Function.** Here we define the function f which needs two input bytes to produce an output result of one byte.

$$\begin{array}{rcl} f : \mathbb{F}_{256} \times \mathbb{F}_{256} & \longrightarrow & \mathbb{F}_{256} \\ (x, y) & \longmapsto & z \end{array}$$

with

$$z := \left[[x \oplus ((255 - y) \ll 1)] + 16 \cdot [((255 - x) \oplus (y \gg 1)) \bmod 16] \right] \bmod 256, \quad (1)$$

where $\oplus$ is the bitwise exclusive or, $+$ represents the classical integer addition, $n \gg 1$ divides n by 2, $n \ll 1$ multiplies n by 2 and keeps the result modulo 256 by not taking into account a possible overflow and "16·" is the classical multiplication by 16. In the subsection 6.2, we explain how to keep lightweight these various operations by using 8-bit registers.

We have the following properties:

- f is non-symmetric, i.e., for all (x, y) pair in $\mathbb{F}_{256} \times \mathbb{F}_{256}$, the function verifies $f(x, y) \neq f(y, x)$;
- f has a uniform distribution of values, i.e., for all z in $\mathbb{F}_{256}$, the function verifies
$$\sharp\{(x, y) \in \mathbb{F}_{256} \times \mathbb{F}_{256} : f(x, y) = z\} = 256.$$

These properties can be easily verified. Hence we consider that the f function is one-way: one cannot retrieve the good (x, y)-entry with the z value. The function h inherits of this property.

Let $i \in \{0, \cdots, 15\}$ a vector index and $j \in \{1, 2, 3, 4\}$ a round index. Let $M = (M_0, \cdots, M_{15})$ a vector of 16 bytes. The function f does not use the same entries depending on a vector index i and a round index j. We define:

$$F_i^j(M) = f(M_i, M_{(i+2^{j-1}) \bmod 16}).$$

and

$$F^j(M) = (F_0^j(M), F_1^j(M), \cdots, F_{15}^j(M)).$$

A working example of these indexes can be found in the table 2.

***S* Function.** Our S function is not a new one. We choose the AES [12, 29] SubBytes function for the quality of its properties.

The SubBytes transformation is a non-linear byte substitution. For example, the eight-bits data "00000000" is transformed into B = "01100011".

To avoid attacks based on simple algebraic properties, the definition of SubBytes Transformation is the composition of the following two transformations in the finite field $\mathbb{F}_{2^8}$ with a chosen structure representation $\mathbb{F}_{2^8} \approx \mathbb{F}_2(X)/(X^8 + X^4 + X^3 + X + 1)$.

The first transformation is the multiplicative inverse in Galois Field $GF(2^8)$, known to have good non-linearity properties. Then the multiplicative inverse of each element is taken (the 8bit-element "00000000", or {00} in hexadecimal format, is mapped to itself). Next, the previous result is combined with an invertible affine transformation:

$$x \mapsto Ax \oplus B,$$

where A is a 8×8 fixed matrix over $GF(2)$ and B is the number defined above and $\oplus$ operates "Exclusive Or" on the individual bits in a byte.

The SubBytes Transformation is also chosen to avoid any fixed point ($S(a) \neq a$), any opposite fixed point ($S(a) \neq \bar{a}$) and also any self invertible point ($S(a) \neq S^{-1}(a)$).

Because it is based on many mathematical objects, the SubBytes function could seem difficult to implement but the transformation could be reduce in an 8-bit substitution box. Hence for any element the result can be found by looking up in a table (see the Figure 7 of [12]: substitution values for the byte $\{xy\}$ (in hexadecimal format)).

We define by S the following transformation: let $M = (M_0, \cdots, M_{15})$ a 16-byte vector. Let S the function which associates M with the vector

$$S(M) = (\text{SubBytes}(M_0), \cdots, \text{SubBytes}(M_{15})).$$

5.2 Description of the Authentication Function $h : (C, K) \longrightarrow H$

Formally, we will follow the tag computation. First, we add the challenge to key by Xor operation, i.e. we calculate $D = C \oplus K = (C_0 \oplus K_0, \ldots, C_{15} \oplus K_{15})$. Then we apply the substitution S to D. The first state M^0 is initialized by $M^0 = S(D)$. Then, we calculate the following values:

$$\begin{aligned} M^1 &= S(F^1(M^0)) \oplus K, \\ M^2 &= S(F^2(M^1))) \oplus K, \\ M^3 &= S(F^3(M^2))) \oplus K, \\ M^4 &= S(F^4(M^3))) \oplus K. \end{aligned}$$

Finally, the function returns $H = M^4 = (M_0^4, \ldots, M_{15}^4)$. We denote the result H by $h_K(C)$.

The figure 1 summarizes this description and a more classical definition can be found through the algorithm 1.

Input : C, K
Output : H
  $M^0 = S(C \oplus K)$
for $j = 1$ to 4 do
  $M^j = S(F^j(M^{j-1})) \oplus K$
end for
  $H = M^4$
return H

Fig. 1. Authentication Function

6 Analysis

6.1 Protocol Security

The identity of the tag is not revealed directly: the tag's identity iD is masked by $h_K(C)$, output of h function which appears random. But the reader can still determine the iD identity using the shared secret key K. The reader verifies that this identity has been used to compute the second part of authentication. At this state, the reader is sure that the tag with iD identity knows the secret key K.

But as aforementioned section 4, a mutual authentication can be set by adding the following steps. The reader shows that it knows K and iD by computing $h_{K \oplus C'}(C \oplus iD)$ where C' is the challenge given by the tag. The tag authenticates the reader by computing in the same way and comparing the proposed result with the computed one. If they are equal, the mutual authentication is achieved.

Algorithm 1. Tag computations

Input: $C = (C_0, \ldots, C_{15})$, $K = (K_0, \ldots, K_{15})$
Output: $H = (H_0, \ldots, H_{15})$

```
{Comment: Computation of M^0 = S(C ⊕ K)}
for i = 0 to 15 do
   M_i ← S(C_i ⊕ K_i)
end for
{Comment: Computation of S(F^j(M^{j-1}))}
for j = 1 to 4 do
   for i = 0 to 15 do
      k ← M_i ⊕ ((M_{i+2^{j-1} mod 16}) ≪ 1)
      l ← (255 − M_i) ⊕ (M_{i+2^{j-1} mod 16} ≫ 1) mod 16
      t ← (k + 16 l) mod 256
      Temp_i ← S(t)
   end for
   {Comment: Computation of M^{j+1} = M^j ⊕ K}
   for i = 0 to 15 do
      M_i ← Temp_i ⊕ K_i
   end for
end for
for i = 0 to 15 do
   H_i ← M_i
end for
return H
```

Now we consider two cases:

- Fake Tag: the tag receives the challenge C. It can choose arbitrarily a number iD to enter into the system. But it does not know K to compute the first part of authentication response.
- Fake reader: the reader chooses and sends C. Next it receives a proper tag authentication. It cannot find iD thanks to $h_{iD}(C)$ (because h is a one-way function) nor K.

6.2 Lightweight

We have to establish that function could be programmed using usual assembler instructions. We refer to ASM51 Assembler [30]. First we use 8-bit registers. To represent an entry of 128 bits, eight registers or space blocks must be reserved.

Next we can implement the f function defined by (1) using very simple instructions using a register named A and a carry C:

- The computation of $A \ll 1$ can be translated by `CLR` C (Clear Carry) followed by `RLC` A (Rotate Left through Carry). The computation of $A \gg 1$ can be translated by `RRC` A (Rotate Right through Carry).
- The computation of $255 - A$ can be translated by `CPL` A, the complemented value.

- The bitwise-xor is classically translated by `XRL`.
- The modular reduction by 16 can by translated by `AND 0x0F`.
- The multiplication by 16 can be translated by four left shift or by `AND 0x0F` followed by `SWAP` which swaps nibbles.
- The modular addition (mod 256) can be translated simply by `ADD` without taking care of possible carries of an 8-bit register.

The SubBytes function can be implemented by looking up in a table as explain in the Figure 7 of [12]. This part of AES algorithm can be computed with a few gates compared to the whole AES (The most penalizing part being the key expansion according to the table 3 of [31]).

Now we claim that properties of h function presented in section 5 are satisfied:

- the overflows of f are intended and contribute to the non-reversibility of the h function,
- the output seems random (subsection 6.4),
- the avalanche criterion (subsection 6.3) shows that the outputs distribution of f is well reported to h outputs.

6.3 Strict Avalanche Criterion

The strict avalanche criterion was originally presented in [32], as a generalization of the avalanche effect [33]. It was introduced for measuring the amount of nonlinearity in substitution boxes (S-boxes), like in the Advanced Encryption Standard (AES).

The avalanche effect tries to reflect the intuitive idea of high-nonlinearity: a very small difference in the input producing a high change in the output, thus an avalanche of changes.

Denote by HW the Hamming weight and $DH(x, y) = HW(x \oplus y)$ the Hamming distance.

Mathematically, the avalanche effect can be formalized by

$$\forall x, y | DH(x, y) = 1, \quad \text{average}(DH(F(x), F(y))) = \frac{n}{2},$$

where F is candidate to have the avalanche effect.

So the output of a n-bit random input number and one generated by randomly flipping one of its bits should be, on average, $n/2$. That is, a minimum input change (one single bit) is amplified and produces a maximum output change (half of the bits) on average.

First we show that if an input bit is changed then the modification will change an average of one half of the following byte. The input byte x will be changed to x' with a difference Δx of one bit. After the first SubBytes transformation, the difference will be

$$S(x \oplus k) \oplus S(x' \oplus k) = S(y) \oplus S(y + \Delta x),$$

with $y = x \oplus k$. We have in average

$$\frac{1}{256 \cdot 8} \sum_{y} \sum_{\Delta x, HW(\Delta x)=1} HW(S(y) \oplus S(y + \Delta x)) \approx 4,$$

where HW is the Hamming weight. Hence an average of four bits will change if the difference is of one bit. Furthermore, for any difference Δx,

$$\frac{1}{256 \cdot 256} \sum_{y} \sum_{\Delta x} HW(S(y) \oplus S(y + \Delta x)) = 4.$$

Our function satisfies the avalanche effect as

$$\frac{1}{256^2} \sum_{x} \sum_{y} HW(x \oplus S(f(x, y))) \approx 4.$$

Next we show that if an input bit is changed then the modification will be spread over all the bytes of the output. Suppose that a bit of the k^{th} byte M_k^0 is changed ($1 \leq k \leq 16$). Then M^1 is also changed as the SubBytes substitution is not a constant function. At the first round, the bytes k and $k + 1$ will be modified. At the second round, the bytes k, $k + 2$, $k + 1$ and $k + 3$ will be modified. Furthermore, eight bytes will be modified and at the end, the whole 16 bytes will be modified.

For example, if the first input byte is changed (M_0^0 is changed). Then M_0^0 is used for compute M_0^1 and M_{15}^1, hence a difference appears in M_0^1 and M_{15}^1, and so on. We trace the difference diffusion in the following table:

Table 1. Diffusion table

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
First Xor	x															
$j = 1$	x															x
$j = 2$	x	x													x	x
$j = 3$	x	x									x	x	x	x	x	x
$j = 4$	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Last Xor	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

If another byte is changed, the same remark works by looking in the dependence table 2.

Hence for any input difference, the modification will change an average of one half of the output.

Table 2. Dependency table

	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14	k=15
j=1	0,1	1,2	2,3	3,4	4,5	5,6	6,7	7,8	8,9	9,1	10,11	11,12	12,13	13,14	14,15	15,0
j=2	0,2	1,3	2,4	3,5	4,6	5,7	6,8	7,9	8,1	9,11	10,12	11,13	12,14	13,15	14,0	15,1
j=3	0,4	1,5	2,6	3,7	4,8	5,9	6,1	7,11	8,12	9,13	10,14	11,15	12,0	13,1	14,2	15,3
j=4	0,8	1,9	2,1	3,11	4,12	5,13	6,14	7,15	8,0	9,1	10,2	11,3	12,4	13,5	14,6	15,7

6.4 Security Quality

To evaluate the security quality, we take $Y = 1$ et $X = 0$. We consider the iterated outputs of the authentication function. Hence we test the series $h_Y(X)$, $h_Y(h_Y(X))$, ... like a random bitstream with the NIST test suite [34]. The bitstream satisfies all the tests (parameters of NIST software: 10^6 input bits, 100 bitstreams).

Table 3. NIST Statistical Test Results

Test Name	Percentage of passing sequences with Significance level $\alpha = 0.01$
1. Frequency Test (Monobit)	99/100
2. Frequency Test (Block)	100/100
3. Runs Test	100/100
4. Longest Run of Ones	99/100
5. Binary Matrix Rank Test	98/100
6. Discrete Fourier Transform Test	98/100
7. Non-Overlapping Template	98/100
8. Overlapping Template	98/100
9. Maurers Universal Statistical	100/100
10 Linear Complexity Test	100/100
11. Serial Test	99/100
12. Approximate Entropy Test	100/100
13. Cumulative Sums (Cusum) Test	98/100
14. Random Excursions Test	90/93
15. Random Excursion Variant Test	91/93

6.5 Hardware Complexity: Implementation and Computational Cost

We choose a 8bit-CPU Tag for cost reasons. We implement the authentication function on a MULTOS Card [35] without difficulties. This card is not a low-cost card but we only test the implementation with basic instructions. The code size of the authentication function (with S-box table) without manual optimization is 798 bytes.

We can optimize the memory usage:

- the S-box table can be placed in Read-Only memory area: 256 bytes needed for AES SubBytes Table.
- the variables placed in the Random Access Memory Memory can be optimized. For internal state computation, one have to represent M with 16 bytes and we need two supplementary temporary bytes: at each round, a state byte value M_i is used twice to compute the next state. In fact M_i^j is used for compute M_i^{j+1} and $M_{i+2^{j-1} \bmod 16}^{j+1}$. After computation of these two variables, the space allocation for the variable M_i^j can be reused. Next we compute the value $M_{i+2^{j-1} \bmod 16}^{j+1}$ depending on $M_{i+2^{j-1} \bmod 16}^{j}$ and another

byte. Now we can delete the memory space for $M^j_{i+2^{j-1} \bmod 16}$ and compute another byte of M_{j+1}, step by step. Hence we use only two additional bytes to compute the next state of M.

We evaluate the computational time with a PC computer (Intel CoreDuo T9600 2.8Ghz): 30 s for 10^7 authentications for a program in a C language, i.e. $3\mu s$ per authentication.

6.6 Privacy

Even if RFID technology is used for identify something in tracing system, in many cases this technology would merely cause infringements of private rights. We do not prevent the tracing system from recording informations but we need to protect the tag iD from external recording. Hence if an attacker records all transactions between tag and a reader, he cannot retrieve if the same tag has been read one or many times. Contrarily, a fake reader can determine if it has previously ask a tag by sending always the same challenge and recording responses, but it cannot know the real iD of the tag.

7 Attacks

The attacker's aim is to validate its tag identity. He can do this by producing a response to a challenge. If he can exploit the attack in a feasible way, then we say that the protocol is broken. Such a success of the attacker might be achieved with or without recovering the secret key shared by the reader and the tag. Hence a large key size is not enough to prove that the protocol cannot be broken with brute force attack. We might also take into account other attacks where the attacker can record, measure and study the tag responses. The necessary data could be obtained in a passive or in an active manner. In case of a passive attack, the attacker collects messages from one or more runs without interfering with the communication between the parties. In case of an active attack, the attacker impersonates the reader and/or the tag, and typically replays purposefully modified messages observed in previous runs of the protocol.

7.1 Recording Attacks

Replay Attack by Recording: An attacker tries to extract the secret of a tag. He uses a reader and knows the commands to perform exchanges with the tag. He asks the tag many times. By listening to different requests, one can record n complete answers. A complete record is composed of a challenge C and the associated response $Auth$. Next if a recording challenge C is used or reused, then the attacker knows the correct response $Auth$. This attack works but

- The attacker must have time to record all the possibilities;
- To create a fake tag, the tag must have $2^{128} \cdot (2 \cdot 2^{128})$ bits (e.g. 10^{60} To) of memory to store the previous records and have the good answer. If this type of tag exists, it is not a commercial one.

– The challenge C, generated by the reader environment, is supposed to be random. So for a fixed C, the probability to have the good answer is very low.

Relay Attack [36]: the attacker makes a link between the reader and tag; it's a kind of Man-in-the-Middle attack. He creates independent connections with reader and tag and relays messages between them. Hence a tag can be identified without being in the reader area. The problem can be treated by security environment protections. A partial solution to protect tag against this attack [37] is to limit its communication distance, but this countermeasure limits the potential of RFID tags. A better way is to activate a distance-bounding protocol [38].

Man-In-The-Middle Attack: A man-in-the-middle attack is not possible because our proposal is based on a mutual authentication, in which two random numbers (C, C'), refreshed at each iteration of the protocol, are used. One cannot forge new responses using challenge differences because $h_{iD}(C+\Delta) \neq h_{iD}(C)+\Delta$ and $h_K(C+\Delta) \neq h_K(C)+\Delta$. In the same way, $h_{K\oplus C'\oplus\Delta}(C\oplus iD) \neq h_{K\oplus C'}(C\oplus iD)\oplus\Delta$.

7.2 Side Channels Attacks

Timing Attack: a timing attack [39] is a side channel attack in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithm. The attack exploits the fact that every operation in a computer takes a dedicated time to execute. If the time cost of operation depends on key value or input values, on can retrieve these secret values by timing attack. Hence, during the implementation, we must be aware of the timing attack. For the computation of tag authentication, the time cost of the operations is the same whatever the value of the key. Next for the reader authentication, the tag must compare the reader response with its own computation. With poor security implementation but unfortunately "classical", if a difference between two bytes is found, the algorithm stops and return the information "Authentication failed". This kind of program is sensible to timing attack. The execution time is different according if the value is rapidly found or not found. To be immune from this attack, we make always a fixed number of steps; the response is send when all the response is verified. One can also add dummy cycles to equilibrate the parts of an implementation. Hence our function is resistant to Timing attack.

Power Consumption Attack: an attacker studies the power consumption [40] of the tag. He can do it by monitoring the delivery power from the reader to the tag. As the consumption of the chip depends on the executed instructions, the attacker can observe (SPA) the different parts of an algorithm. Here the algorithm does not need to be secret and the operations do not depend on the key values. One can also use random dummy cycles to disrupt the observation of the same part of program execution. Hence our function is SPA-resistant.

7.3 Mathematical Attacks

Lucky Authentication: A attacker tries to have a good authentication with a fake tag. He sends (C_1, C_2) as $Auth$. The first part $C_1 = iD \oplus h_K(C)$ of the response can be decoded as a existing iD if there is enough tags. But the second part $C_2 = h_{iD}(C)$ is fixed by the decoding iD and the challenge C. The size of C_2 is 16 bytes. Hence

$$P(\text{Authentication OK/False Tag}) \leq \frac{1}{2^{128}}.$$

Nowadays, this probability is sufficient for a good security.

Active Attack: Suppose that an attacker queries the tag $\mathcal{T}$ by sending $C = 0$ as challenge. Then, to determine the secret K, it must solve the equation

$$S(F^4(S(F^3(S(F^2(S(F^1(S(K))) \oplus K)) \oplus K)) \oplus K)) \oplus K = H, \tag{2}$$

where H is the response of $\mathcal{T}$ and the unknowns are the bytes of K. Since each round of the algorithm operations are performed modulo 16 or modulo 256 and the results from these transactions are processed by substitution tables, the equation 2 is very difficult to analyze algebraically.

Linear [41] or Differential [42] Attacks: These attacks depend especially on properties of the substitution function. First remember that for a function g from $\mathbb{F}_{2^m}$ to $\mathbb{F}_{2^m}$, a differential pair (α, β) is linked with the equation $g(x \oplus \alpha) \oplus g(x) = \beta$. The differential attack is based on finding pairs where the probability

$$P(\sharp\{x \in \mathbb{F}_{2^m} : g(x \oplus \alpha) \oplus g(x) = \beta\})$$

is high. If such pair exists then the attack is feasible. Our function is well resistant to this attack. Indeed the substitution function S is constructed by composing a power function with an affine map, which avoid from differential attacks. Our h function inherits from these properties: considering the output z of $f(x, y)$ describes in the paragraph 5.1, it is easy to verify (like in the paragraph 6.3) that

$$\text{for all} \quad \alpha, \beta \in \mathbb{F}_{256}, \quad \sharp\{z \in \mathbb{F}_{256} : S(z \oplus \alpha) \oplus S(z) = \beta\} \leq 4.$$

It allows to avoid the existence of differential pair such that the probability

$$P(\sharp\{x \in \mathbb{F}_{256} : S(x \oplus \alpha) \oplus S(x) = \beta\})$$

be high.

To achieve a linear attack, it aims at awarding credibilities to the equations of the type

$$\langle \alpha, x \rangle \oplus \langle \beta, S(x) \rangle = 0, \quad \text{with} \quad \alpha, \beta \in \mathbb{F}_{256}.$$

We know that for all α and β not identically equal to zero, the equation has a number of solutions close to 128 which makes expensive the linear attack.

7.4 Desynchronizing Attack

In a desynchronization attack, the adversary aims to disrupt the key update leaving the tag and reader in a desynchronized state in which future authentication would be impossible. Compared to some other protocols, the key does not change in our authentication protocol. It is not a lack of security, the key may change during stocktaking or subscription renewal, by changing tag by another with the new key.

8 Conclusion

We have presented a lightweight authentication protocol for low-cost RFID tags. The internal functions are well adapted for 8-bit CPU with few memory and without cryptoprocessor, even if it is true that a precise evaluation of the building cost and performance of a tag supporting our protocol (i.e. very few CPU functions and less than 1Kbytes of memory) should be evaluated with a manufacturer.

We use the security qualities of the AES S-Boxes to build a function, specifically dedicated to the authentication, which keeps them. The notions of privacy and the classic attacks are addressed. The proposed version is light in terms of implementation and in a reduced cost what makes it usable on RFID systems. Even if these systems are intended for simple applications as secure counter of photocopies or stock management in a small shop, the security level reached here allows to envisage more ambitious applications.

Acknowledgements. The authors want to thank the anonymous reviewers for their constructive comments which were helpful to improve this paper and Damien Sauveron for proofreading of preliminary versions.

References

1. Agarwal, A., Mitra, M.: RFID: Promises and Problems (April 2006)
2. Weis, S.A.: Rfid (radio frequency identification): Principles and applications
3. Nath, B., Reynolds, F., Want, R.: Rfid technology and applications. IEEE Pervasive Computing 5(1), 22–24 (2006)
4. Östman, H.: Rfid - 5 most common applications on the shop floor (2012), `http://www.rfidarena.com/2012/12/13/rfid-%E2%80%93-5-most-common-applications-on-the-shop-floor.aspx`
5. James, J.: Fda, companies test rfid tracking to prevent drug counterfeiting. AIDS Treat News (417), 5–8 (2005)
6. Miles, S., Sarma, S., Williams, J.: RFID Technology and Applications. Cambridge University Press (2011)
7. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
8. Feldhofer, M., Wolkerstorfer, J.: Strong crypto for rfid tags - a comparison of low-power hardware implementations. In: ISCAS, pp. 1839–1842. IEEE (2007)

9. Kömmerling, O., Kuhn, M.G.: Design principles for tamper-resistant smartcard processors. In: Proceedings of the USENIX Workshop on Smartcard Technology, p. 2. USENIX Association (1999)
10. Poschmann, A.: Lightweight cryptography - cryptographic engineering for a pervasive world. IACR Cryptology ePrint Archive 2009, 516 (2009)
11. Hellekalek, P.: Good random number generators are (not so) easy to find. Math. Comput. Simul. 46(5-6), 485–505 (1998)
12. NIST: Advanced encryption standard (aes), fips 197 (November 2001), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
13. Eastlake, D.E., Jones, P.E.: US Secure Hash Algorithm 1 (SHA1), http://www.ietf.org/rfc/rfc3174.txt?number=3174
14. Rivest, R.L.: The MD5 Message-Digest Algorithm (RFC 1321), http://www.ietf.org/rfc/rfc1321.txt?number=1321
15. Vajda, I., Buttyán, L.: Lightweight authentication protocols for low-cost rfid tags. In: 2nd Workshop on Security in Ubiquitous Computing, in conjunction with Ubicomp 2003 (October 2003)
16. Peris-Lopez, P., Hern, J.C., Tapiador, J.M.E., Ribagorda, A.: Lmap: A real lightweight mutual authentication protocol for low-cost rfid tags. In: Proc. of 2nd Workshop on RFID Security, Ecrypt, p. 06 (2006)
17. Peris-Lopez, P., Hernandez-Castro, J.C., Estevez-Tapiador, J.M., Ribagorda, A.: M^2AP: A minimalist mutual-authentication protocol for low-cost RFID tags. In: Ma, J., Jin, H., Yang, L.T., Tsai, J.J.-P. (eds.) UIC 2006. LNCS, vol. 4159, pp. 912–923. Springer, Heidelberg (2006)
18. Peris-Lopez, P., Hernandez-Castro, J.C., Estevez-Tapiador, J.M., Ribagorda, A.: EMAP: An efficient mutual-authentication protocol for low-cost RFID tags. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops 2006, Part I. LNCS, vol. 4277, pp. 352–361. Springer, Heidelberg (2006)
19. Defend, B., Fu, K., Juels, A.: Cryptanalysis of two lightweight rfid authentication schemes. In: PerCom Workshops, pp. 211–216. IEEE Computer Society (2007)
20. Li, T., Wang, G.: Security analysis of two ultra-lightweight RFID authentication protocols. In: Venter, H., Eloff, M., Labuschagne, L., Eloff, J., von Solms, R. (eds.) New Approaches for Security, Privacy and Trust in Complex Environments. IFIP, vol. 232, pp. 109–120. Springer, Boston (2007)
21. Lee, J., Yeom, Y.: Efficient rfid authentication protocols based on pseudorandom sequence generators. IACR Cryptology ePrint Archive 2008, 343 (2008)
22. Lo, N.W., Yeh, K.H.: De-synchronization attack on rfid authentication protocols. In: International Symposium on Information Theory and its Applications (ISITA), pp. 566–570 (October 2010)
23. van Deursen, T., Radomirovic, S.: Security of rfid protocols - a case study. Electr. Notes Theor. Comput. Sci. 244, 41–52 (2009)
24. Sixth International Conference on Availability, Reliability and Security, ARES 2011, Vienna, Austria, August 22-26. IEEE (2011)
25. Yeh, T.C., Wang, Y.J., Kuo, T.C., Wang, S.S.: Securing rfid systems conforming to epc class 1 generation 2 standard. Expert Syst. Appl. 37(12), 7678–7683 (2010)
26. Habibi, M.H., Alagheband, M.R., Aref, M.R.: Attacks on a lightweight mutual authentication protocol under EPC C-1 G-2 standard. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 254–263. Springer, Heidelberg (2011)
27. Hernandez-Castro, J.C., Peris-Lopez, P., Safkhani, M., Bagheri, N., Naderi, M.: Another fallen hash-based RFID authentication protocol. In: Askoxylakis, I., Pöhls, H.C., Posegga, J. (eds.) WISTP 2012. LNCS, vol. 7322, pp. 29–37. Springer, Heidelberg (2012)

28. Song, B., Hwang, J.Y., Shim, K.A.: Security improvement of an rfid security protocol of iso/iec wd 29167-6. IEEE Communications Letters 15(12), 1375–1377 (2011)
29. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
30. Intel: Mcs-51 instruction set summary (1979)
31. Hamalainen, P., Alho, T., Hannikainen, M., Hamalainen, T.D.: Design and implementation of low-area and low-power aes encryption hardware core. In: Proceedings of the 9th EUROMICRO Conference on Digital System Design, DSD 2006, pp. 577–583. IEEE Computer Society, Washington, DC (2006)
32. Forré, R.: The strict avalanche criterion: Spectral properties of boolean functions and an extended definition. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 450–468. Springer, Heidelberg (1990)
33. Webster, A.F., Tavares, S.E.: On the design of S-boxes. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 523–534. Springer, Heidelberg (1986)
34. NIST: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. NIST Special Publication 800-22rev1a (April 2010)
35. Multos: Multos developer's guide (2012)
36. Kasper, T., Carluccio, D., Paar, C.: An embedded system for practical security analysis of contactless smartcards. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 150–160. Springer, Heidelberg (2007)
37. Schneier, B.: Rfid cards and man-in-the-middle attacks. Schneier Security Blog (2006)
38. Hancke, G.P., Kuhn, M.G.: An rfid distance bounding protocol. In: SecureComm, pp. 67–73. IEEE (2005)
39. 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)
40. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
41. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
42. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)

Author Index