Anonymous Split E-Cash – Towards Mobile Anonymous Payments

MARIJN SCHEIR, KU Leuven ESAT/COSIC and iMinds

JOSEP BALASCH, KU Leuven ESAT/COSIC and iMinds

ALFREDO RIAL, IBM Research Zürich

BART PRENEEL, KU Leuven ESAT/COSIC and iMinds

INGRID VERBAUWHEDE, KU Leuven ESAT/COSIC and iMinds

Anonymous E-Cash was first introduced in 1982 as a digital, privacy-preserving alternative to physical cash. A lot of research has been since then devoted to extend and improve its properties, leading to the appearance of multiple schemes. Despite this progress, the practical feasibility of E-Cash systems is still today an open question. Payment tokens are typically portable hardware devices in smart card form, resource constrained due to their size, and therefore not suited to support largely complex protocols such as E-Cash. Migrating to more powerful mobile platforms, for instance smartphones, seems a natural alternative. However, this implies to move computations from trusted and dedicated execution environments to generic multi-application platforms, which may result in security vulnerabilities. In this work we propose a new anonymous E-Cash system to overcome this limitation. Motivated by existing payment schemes based on MTM (Mobile Trusted Module) architectures, we consider at design time a model in which user payment tokens are composed of two modules: an untrusted but powerful execution platform (e.g. smartphone) and a trusted but constrained platform (e.g. secure element). We show how the protocol's computational complexity can be relaxed by a secure split of computations: non-sensitive operations are delegated to the powerful platform, while sensitive computations are kept in a secure environment. We provide a full construction of our proposed Anonymous Split E-Cash scheme and show that it fully complies with the main properties of an ideal E-Cash system. Finally, we test its performance by implementing it on an Android smartphone equipped with a Java Card compatible secure element.

General Terms: Security, Privacy, Design

Additional Key Words and Phrases: E-Cash, privacy-enhancing technologies, payment systems, anonymity, Android, Java Card

1. INTRODUCTION

As the importance of electronic payments in today's ubiquitous society continues to grow, so does users' awareness on the amount of personal information that is involved in financial transactions. These data can be easily collected and analyzed by banks and/or vendors for purposes unrelated to the actual transaction, the most common one being direct marketing techniques. Enforcing a correct use of the information once it has left the users' control seems utopian, as there is no realistic way to prevent data from being distributed, stolen, and essentially further (mis-)used. More troublesome is the fact that sensitive personal information might inherently be stored within payment transaction databases: sexual orientation and political views might be inferred from a user's list of purchases, medical conditions can be exposed by analyzing a set of procured medications, and precise location is often revealed in scenarios where electronic payments are tied to physical locations, e.g. in public transportation systems. In all these scenarios there is a clear need for alternative electronic payments aiming to conceal, or at least minimize, the disclosure of sensitive information.

Privacy-preserving electronic payments have been known to be possible since 1982, when David Chaum introduced the concept of anonymous E-Cash [Chaum 1983].

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish Government, FWO G.00130.13N and FWO G.0876.14N. Josep Balasch, Bart Preneel and Ingrid Verbauwhede are with iMinds and the COSIC group of Departement Elektrotechniek (ESAT), KU Leuven, Belgium, e-mail: firstname.lastname@esat.kuleuven.be. Alfredo Rial is with IBM Research, e-mail: lia@zurich.ibm.com. Part of this work was carried out while Marijn Scheir was a master student at the COSIC research group.

Briefly speaking an anonymous E-Cash system is composed of three players: a central bank, a set of merchants (or vendors), and a set of users (or buyers). Users can *withdraw* a wallet of electronic coins from the bank, and they can later *spend* these coins by purchasing goods from a merchant. At the end of the purchase transaction, the merchants can *deposit* the collected coins into their bank account.

From a functional point of view this procedure is similar to existing off-line (non-anonymous) payment systems such as Proton [PROTON 2014] (in Belgium), Chipknip [Chipknip 2014] (in the Netherlands), or Geldkarte [Geldkarte 2014] (in Germany). From a privacy point of view however, an anonymous E-Cash system ensures some additional properties, namely:

- (1) *User anonymity*: It is not possible for a merchant to learn the identity of a user during or after a purchase transaction.
- (2) *Payment unlinkability*: It is not possible for a merchant to determine whether two or more purchase transactions correspond to the same or different user(s).
- (3) *Coin unlinkability*: It is not possible for a bank to conclude whether a coin withdrawn by a user corresponds to a coin deposited by a merchant.

These strong privacy properties can obviously not conflict with other generic security requirements of off-line payments, for instance prevention of coin *double-spending* and coin *double-depositing*. E-Cash protocols are designed such that system misuse invalidates the privacy guarantees. For instance, users that double-spend a coin are automatically detected and their anonymity is revoked. Similarly, vendors attempting to double-deposit coins can be exposed and identified.

Another similarity between anonymous and non-anonymous payment systems is the necessity of a portable payment token on the user side. The role of this token is not only to store secret cryptographic materials, but also to carry out payment transactions in a secure fashion. A *tamper resistant* element, for instance a smart card with controlled execution environments, is often used to safeguard the system against corruption efforts from malicious participants. The security of these trusted elements is certified by evaluation labs in numerous Common Criteria [Common Criteria 2014] and EMVCo [EMVCo 2014] certifications before being accepted into the banking market.

Unfortunately, the high complexity of anonymous E-Cash systems leads to more stringent implementation costs than non-anonymous schemes. As a consequence, current standalone payment tokens appear insufficiently powerful to support the complex protocols employed in E-Cash. In this work we devise a new anonymous E-Cash scheme that does not require a fully tamper resistant element on the user side to perform all operations. Instead, it takes inspiration from Mobile Trusted Module (MTM) architectures and considers at design time a payment token composed of two independent embedded computing platforms: an untrusted but powerful execution platform (mobile phone) and a trusted but constrained computation module (secure element). Our system is designed such that the computational complexity at the user side is securely split into two platforms.

1.1. Related Work

The introduction of Chaum's anonymous E-Cash led to the subsequent appearance of systems that achieved new and interesting properties. Among them, Camenisch *et al.* [Camenisch et al. 2005] proposed *compact* E-cash, a system that greatly improved the computational and storage requirements for withdrawing and storing electronic coins. Okamoto [Okamoto 1995] investigated the concept of *divisible* E-cash, allowing users to divide the value of a digital coin to spend it in multiple times. Later, Canard and Gouget [Canard and Gouget 2007] proposed the first divisible off-line E-

Cash scheme with both unlinkability and anonymity without trusted third party. Further works building on the ideas of [Camenisch et al. 2005] introduced other properties such as coin endorsement [Camenisch et al. 2007] or coin transferability [Canard et al. 2008; Canard and Gouget 2008].

In parallel to this, there have also been efforts to evaluate the feasibility of anonymous E-Cash systems and their cryptographic building blocks on hardware platforms. An important line of work has focused on implementations of anonymous credential systems such as the Identity Mixer (IDEMIX) [Camenisch and Lysyanskaya 2001; Camenisch and Herreweghen 2002]. A variant of the Direct Anonymous Attestation (DAA) protocol [Brickell et al. 2004] is implemented by Balasch [Balasch 2008] on an 8-bit AVR microcontroller. Similar schemes are presented by Sterckx *et al.* [Sterckx et al. 2009] and by Bichsel *et al.* [Bichsel et al. 2009], both running on standard Java Card smart cards. An implementation of the U-Prove credential system [Brands 2000] on a MULTOS smart card platform is described by Mostowski and Vullers [Mostowski and Vullers 2011]. Finally, implementations of self-blindable credentials due to Verheul [Verheul 2001] on Java Card platforms are presented by Batina *et al.* [Batina et al. 2010].

An implementation of the E-Cash scheme due to Brands [Brands 1994] is presented by Clemente-Cuervo *et al.* [Clemente-Cuervo et al. 2007]. The authors use a PDA equipped with a 400 MHz Xscale processor as target device. The same scheme is implemented by Hinterwalder *et al.* [Hinterwalder et al. 2013a] on a dedicated Moo RFID tag [Hong Zhang 2011] equipped with a Texas Instruments MSP430F2618 16-bit processor. Further works have investigated the use case of anonymous ticketing for NFC mobile phones, for instance, Derler *et al.* [Derler et al. 2011] and Hinterwalder *et al.* [Hinterwalder et al. 2013b].

In contrast to these works, our work considers the mobile phone as untrusted platform and therefore unsuitable to be used as a standalone payment token. We consequently look for another approach, namely assume the existence of a trusted device that collaborates with the phone in order to carry out payment transactions. We stress that this architecture is not novel, i.e. the concept of MTM acting as a trusted anchor in mobile platforms already exists. In fact, it is used by several existing mobile payment applications. Original Equipment Manufacturers (OEMs) such as Google or Apple employ in-built secure elements as trust anchors of their payment applications Google Wallet and Apple Pay, respectively. Similarly, Visa payWave [Visa payWave 2014] is only available to mobile phones certified by Visa which contain, among others, a Visa approved hardware and software secure container. Finally, other companies make use of SIM cards as trusted elements present in the phone. This is the case for instance of the Mobile PayPass application from MasterCard.

1.2. Our contribution

The main contribution of our work is to bring MTM models as currently used by (non-anonymous) payment applications to the world of anonymous E-Cash systems. For this, we propose a new anonymous E-Cash system specially suited for MTM architectures. Our system, which we call Anonymous Split E-Cash, is a variant of the compact E-Cash scheme of Camenisch *et al.* [Camenisch et al. 2005]. The main difference is that calculations on the user side are split into two independent computing platforms that communicate with each other. Non-sensitive operations, i.e. those which do not rely on secrets and/or critical payment information, are carried out in a powerful but untrusted device, e.g. an embedded controller running an operating system. Sensitive operations on the other hand are performed on a less powerful but completely trusted element providing strong security guarantees. The key challenge of our approach lies in determining where and how to efficiently split the computations between these two

elements. For this we follow a similar approach as in the Direct Anonymous Attestation (DAA) protocol [Brickell et al. 2004] employed in Trusted Platform Module (TPM) systems.

Our proposed system complies with the main properties of existing e-cash schemes, namely correctness, user anonymity, payment unlinkability, detection of double spenders, and balance correctness of the bank. Additionally, it fulfills properties to protect the trusted element from a corrupt device: withdraw authorization, withdraw completion, spend authorization and spend completion. They ensure that a corrupt device cannot withdraw and spend coins without authorization from the trusted element. In order to determine to what extent the proposed E-Cash system can lead to realizable implementations, we also evaluate the performance of a proof-of-concept implementation. As target device we select a combination of off-the-shelf devices: an Android based smartphone (playing the role of the powerful, untrusted environment) and a certified Java Card secure module (playing the role of the computational CPU in the trusted element).

2. HIGH-LEVEL DESCRIPTION

Figure 1 illustrates the three parties that interact with each other in our anonymous E-Cash system: a bank $\mathcal{B}$, a merchant $\mathcal{M}$ and a user $\mathcal{U}$. We distinguish operations performed by the user depending on whether they are executed on the host $\mathcal{H}$ or on the secure element $\mathcal{SE}$. The host is a relatively computationally powerful device owned by $\mathcal{U}$. As depicted in the figure, we envision $\mathcal{H}$ to be a smartphone. Mobile phones play nowadays a central role in people's lives, which makes them a suitable choice for our system. Due to their multi-application architecture and the large amount of stakeholders involved in their manufacturing and deployment, we naturally consider them untrusted in our E-Cash construction and only take advantage of their powerful computational capabilities. The security assurances are instead placed on $\mathcal{SE}$, a less powerful platform connected to $\mathcal{H}$ that stores secret parameters and only carries out operations depending on such parameters.

For security reasons, it is necessary that there exists a direct link between the human user and $\mathcal{SE}$. This is a consequence of our model. Because $\mathcal{H}$ is an untrusted entity, communication between the human user and $\mathcal{SE}$ cannot be simply relayed through $\mathcal{H}$. In other words, $\mathcal{SE}$ cannot be fully integrated within $\mathcal{H}$. If that were the case, the information exchanged between the human user and $\mathcal{SE}$ could be altered or simply blocked. Moreover, $\mathcal{H}$ could try to execute withdraw and spend transactions without involving the human user, leading to undesired corruption scenarios.

In order to solve these issues, we envision $\mathcal{SE}$ to be a relatively small custom payment token that *externally* plugs to $\mathcal{H}$. This payment token would be provided by $\mathcal{B}$ to $\mathcal{U}$ upon engagement in the E-Cash application, similarly to today's smart card based electronic payments. Communication between $\mathcal{H}$ and $\mathcal{SE}$ takes place via the employed external phone interface, for instance, a micro USB port. Additionally, $\mathcal{SE}$ comes equipped with means to directly interface to $\mathcal{U}$. We envision a simple yet sufficient interface composed of two main components: a small LCD module and a few push buttons. The LCD module is used to visually inform the user of the requests and amounts involved in each transaction. The push buttons on the other hand, allow the user to initiate the main E-Cash operations (withdraw and spend) as well as to confirm/reject transactions. Additionally, other push buttons such as increase/decrease could be used to modify the number of coins involved in the transaction.

Besides these changes in the user domain, interactions between anonymous E-Cash parties are similar to those of existing off-line payment schemes. An initialization setup phase allows to generate and distribute cryptographic keys and parameters to all participating parties. In the first step $\mathcal{U}$ and $\mathcal{B}$ execute together a withdraw protocol.

Fig. 1. Anonymous E-Cash for resource constrained devices.

As a result, $\mathcal{U}$ obtains a wallet composed of n coins. The withdrawal step is executed in such a way that $\mathcal{H}$ obtains only partial information about the wallet, thus preventing $\mathcal{H}$ from spending coins without $\mathcal{SE}$'s approval. In the second step, $\mathcal{U}$ and $\mathcal{M}$ run the spend protocol in which $\mathcal{U}$ transfers a coin of a certain wallet to $\mathcal{M}$. Finally in the third step, $\mathcal{M}$ sends a coin to $\mathcal{B}$ via the deposit protocol to store it in its bank account. Before $\mathcal{B}$ accepts the coin, it checks that neither the coin has already been deposited nor that $\mathcal{U}$ has committed double spending. If both checks turn negative, $\mathcal{B}$ deposits the coin in $\mathcal{M}$'s bank account. Otherwise $\mathcal{B}$ does not accept the coin and runs the accuse protocol. In case of a double-deposit, $\mathcal{B}$ marks $\mathcal{M}$ as double depositor; in case of double-spending, $\mathcal{B}$ is able to revoke $\mathcal{U}$'s anonymity and to issue a publicly-verifiable proof of double-spending.

2.1. Security Framework

The security of a protocol φ is analyzed by comparing the view of an environment $\mathcal{Z}$ in a real execution of φ against that of $\mathcal{Z}$ when interacting with an ideal functionality $\mathcal{F}$ that carries out the desired task. The environment $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. In the real world, $\mathcal{Z}$ can communicate freely with an adversary $\mathcal{A}$ who controls the network as well as any corrupt parties. In the ideal world, $\mathcal{Z}$ interacts with dummy parties, who simply relay inputs and outputs between $\mathcal{Z}$ and $\mathcal{F}$, and a simulator $\mathcal{S}$. We say that a protocol φ securely realizes $\mathcal{F}$ if $\mathcal{Z}$ cannot distinguish the real world from the ideal world, i.e., $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and parties running protocol φ or with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}_\varphi$.

The identity of a party consists of a party identifier pid and a session identifier sid . A set of parties are a protocol instance if they have the same session identifier sid .

A protocol $\varphi^{\mathcal{G}}$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$ -hybrid model when φ is allowed to invoke the ideal functionality $\mathcal{G}$. Therefore, for any protocol ψ that securely realizes functionality $\mathcal{G}$, the composed protocol φ^ψ , which is obtained by replacing each invocation of an instance of $\mathcal{G}$ with an invocation of an instance of ψ , securely realizes $\mathcal{F}$.

Our protocol makes use of the functionalities $\mathcal{F}_{\text{REG}}$ for key registration, $\mathcal{F}_{\text{SMT}}$ for secure message transmission, and $\mathcal{F}_{\text{ASMT}}$ for anonymous secure message transmission. We refer to [Camenisch et al. 2014] for the description of those ideal functionalities.

2.2. Ideal Functionality

To describe how our anonymous split E-Cash system should act we describe an ideal functionality $\mathcal{F}_{\text{ASEC}}$. $\mathcal{F}_{\text{ASEC}}$ interacts with the following dummy parties: the bank $\mathcal{B}$, the secure elements $\mathcal{SE}_1, \dots, \mathcal{SE}_I$, the hosts $\mathcal{H}_1, \dots, \mathcal{H}_I$ and the vendors $\mathcal{M}_1, \dots, \mathcal{M}_J$. The parties are dummy in the sense that they just forward their inputs to $\mathcal{F}_{\text{ASEC}}$. Based on these inputs, $\mathcal{F}_{\text{ASEC}}$ calculates the desired output and forwards this output to each of the parties it is interacting with.

The identifier $\mathcal{U}_i$ denotes a pair $(\mathcal{SE}_i, \mathcal{H}_i)$. Our functionality assigns a unique identifier to wallets and coins. Each wallet is denoted by $(\mathcal{U}_i, w)$ such that $w \in [1, w_i]$, where w_i is a counter of the number of wallets of $\mathcal{U}_i$. Each coin is given an identifier $(\mathcal{U}_i, w, cn)$, where cn specifies a coin counter that is used to distinguish between coins in the wallet w .

Our functionality stores all the relevant information into a table Table I. Table I assigns to each coin $(\mathcal{U}_i, w, cn)$ a unique random identifier P . P is a coin pseudonym given to the vendors $\mathcal{M}_j$ and to the bank $\mathcal{B}$, and cannot be linked to the user that spent that coin. Additionally, Table I contains a list of transaction identifiers, one for each time the coin was spent. Each transaction identifier is associated to the identity of the merchant that received the coin at that transaction, and a bit that indicates whether the coin obtained with that transaction identifier was deposited at the bank or not.

Table I. An entry in the table of $\mathcal{F}_{\text{ASEC}}$.

$(\mathcal{U}_i, w, cn)$	P	$[id_1, \mathcal{M}_{j,1}, b_1], \dots, [id_M, \mathcal{M}_{j,M}, b_M]$
--------------------------	-----	---

We depict the ideal functionality $\mathcal{F}_{\text{ASEC}}$ and discuss the interfaces of $\mathcal{F}_{\text{ASEC}}$:

Functionality $\mathcal{F}_{\text{ASEC}}$

Running with a bank $\mathcal{B}$, vendors $\mathcal{M}_1, \dots, \mathcal{M}_J$, hosts $\mathcal{H}_1, \dots, \mathcal{H}_I$ and secure elements $\mathcal{SE}_1, \dots, \mathcal{SE}_I$, and parameterized with a number of coins per wallet N , $\mathcal{F}_{\text{ASEC}}$ works as follows:

- (1) On input (setup, sid) from $\mathcal{B}$, $\mathcal{F}_{\text{ASEC}}$ executes the following program:
 - Continue only if $sid = (\mathcal{B}, sid')$ and if Table I is not initialized.
 - Initialize an empty table Table I. For $i = 1$ to I , initialize to zero a wallet counter w_i . Send a public delayed output (setupend, sid) to $\mathcal{B}$.
- (2) On input (withdraw, sid) from $\mathcal{SE}_i$, $\mathcal{F}_{\text{ASEC}}$ executes the following program:
 - Continue only if $sid = (\mathcal{B}, sid')$.
 - Send (withdraw, sid) to $\mathcal{S}$ and wait for (withdraw, sid) from $\mathcal{S}$.
 - If $\mathcal{H}_i$ or $\mathcal{B}$ is corrupt, send (withdraw, $sid, \mathcal{SE}_i$) to $\mathcal{S}$ and wait for (withdraw, sid, b) from $\mathcal{S}$. If $b = 0$, send (withdrawend, $sid, 0$) to $\mathcal{SE}_i$ and exit the program.
 - Increment the counter w_i and initialize to zero a coin counter cn for wallet w_i of user $\mathcal{U}_i$.
 - Send (withdrawend, sid, w_i) to $\mathcal{SE}_i$ and (withdrawend, $sid, \mathcal{SE}_i$) to $\mathcal{B}$.
- (3) On input (spend, $sid, w, \mathcal{M}_j$) from $\mathcal{SE}_i$, $\mathcal{F}_{\text{ASEC}}$ runs the following program:
 - Continue only if $sid = (\mathcal{B}, sid')$ and $w \in [1, w_i]$.
 - Increment the counter cn for wallet w of user $\mathcal{U}_i$. Continue only if $cn \leq N$ or if $\mathcal{SE}_i$ is corrupt.
 - If $\mathcal{SE}_i$ is corrupt, send (spend, sid) to $\mathcal{SE}_i$ and wait for (spend, sid, cn) from $\mathcal{SE}_i$.
 - Send (spend, sid) to $\mathcal{S}$ and wait for (spend, sid) from $\mathcal{S}$.

- If $\mathcal{H}_i$ is corrupt, send $(\text{spend}, \text{sid}, \mathcal{SE}_i)$ to $\mathcal{S}$ and wait for $(\text{spend}, \text{sid}, b, c)$ from $\mathcal{S}$. If $b = 0$, send $(\text{spend}, \text{sid}, 0)$ to $\mathcal{SE}_i$ and exit the program.
 - If $\mathcal{M}_j$ is corrupt, send $(\text{spend}, \text{sid})$ to $\mathcal{S}$ and wait for $(\text{spend}, \text{sid}, id)$ from $\mathcal{S}$, else create a unique transaction identifier id .
 - If there is not an entry $[(\mathcal{U}'_i, w', cn'), \perp, \perp]$ in Table I such that $\mathcal{U}'_i = (\mathcal{SE}_i, \mathcal{H}_i)$, $w' = w$ and $cn = cn'$, create a random unique P and append a new row $[(\mathcal{U}_i, w, cn), P, [id, \mathcal{M}_j, 0]]$ to Table I, else add a new tuple $[id, \mathcal{M}_j, 0]$ to the entry $[(\mathcal{U}_i, w, cn), \perp, \perp]$.
 - If $c = 0$ and $\mathcal{H}_i$ is corrupt, send $(\text{spendend}, \text{sid}, P, id, \mathcal{SE}_i)$ to $\mathcal{M}_j$, else send $(\text{spendend}, \text{sid}, P, id)$ to $\mathcal{M}_j$. Send $(\text{spendend}, \text{sid}, P, id)$ to $\mathcal{SE}_i$.
- (4) On input $(\text{deposit}, \text{sid}, P, id)$ from $\mathcal{M}_j$, $\mathcal{F}_{\text{ASEC}}$ executes the following program:
- Continue only if $\text{sid} = (\mathcal{B}, \text{sid}')$ and there is a row $[(\mathcal{U}_i, w, cn), P', [id'_1, \mathcal{M}'_{j,1}, b_1], \dots, [id'_M, \mathcal{M}'_{j,M}, b_M]]$ in Table I such that both $P = P'$ and there is tuple $[id'_m, \mathcal{M}'_{j,m}, b_m]$ such that $id'_m = id$ and $\mathcal{M}'_{j,m} = \mathcal{M}_j$.
 - Send $(\text{deposit}, \text{sid})$ to $\mathcal{S}$ and wait for $(\text{deposit}, \text{sid})$ from $\mathcal{S}$.
 - If in the tuple $[id'_m, \mathcal{M}'_{j,m}, b_m]$ such that $id'_m = id$ and $\mathcal{M}'_{j,m} = \mathcal{M}_j$ mentioned above, $b_m = 1$, send $(\text{depositend}, \text{sid}, \mathcal{M}_j, P, id, d\text{depositalert})$ to $\mathcal{B}$.
 - Else, for all the tuples $[id'_1, \mathcal{M}'_{j,1}, b_1], \dots, [id'_M, \mathcal{M}'_{j,M}, b_M]$ in the same row in Table I, if $b_m = 1$ for any $m \in [1, M]$, send $(\text{depositend}, \text{sid}, \mathcal{M}_j, \mathcal{U}_i, P, id, d\text{spentalert})$ to $\mathcal{B}$ and set $b_m = 1$ in the tuple $[id'_m, \mathcal{M}'_{j,m}, b_m]$ such that $id'_m = id$ and $\mathcal{M}'_{j,m} = \mathcal{M}_j$.
 - Else, set $b_m = 1$ in the tuple $[id'_m, \mathcal{M}'_{j,m}, b_m]$ such that $id'_m = id$ and $\mathcal{M}'_{j,m} = \mathcal{M}_j$ and send $(\text{depositend}, \text{sid}, \mathcal{M}_j, P, id)$ to $\mathcal{B}$.
- (5) On input $(\text{accuse}, \text{sid}, \mathcal{U}_i, P, id, \hat{id})$ from any party $\mathcal{P}$, $\mathcal{F}_{\text{ASEC}}$ runs as follows:
- Continue only if $\text{sid} = (\mathcal{B}, \text{sid}')$ and there is a row $[(\mathcal{U}'_i, w, cn), P', [id'_1, \mathcal{M}'_{j,1}, b_1], \dots, [id'_M, \mathcal{M}'_{j,M}, b_M]]$ in Table I such that $\mathcal{U}'_i = \mathcal{U}_i$, $P = P'$ and there are two tuples $[id'_k, \mathcal{M}'_{j,k}, b_k]$ and $[id'_m, \mathcal{M}'_{j,m}, b_m]$ such that $id'_k = id$ and $id'_m = \hat{id}$.
 - In those tuples $[id'_k, \mathcal{M}'_{j,k}, b_k]$ and $[id'_m, \mathcal{M}'_{j,m}, b_m]$, if $b_k = b_m = 1$, send $(\text{accuseend}, \text{sid}, \text{guilty})$ to $\mathcal{P}$, else send $(\text{accuseend}, \text{sid}, \text{notguilty})$ to $\mathcal{P}$.

- (1) The setup interface is called by the bank $\mathcal{B}$. First, $\mathcal{F}_{\text{ASEC}}$ checks if $\text{sid} = (\mathcal{B}, \text{sid}')$. This check allows each bank to have its own instance of the e-cash protocol. $\mathcal{F}_{\text{ASEC}}$ initializes an empty table Table I and a wallet counter w_i for each secure element $\mathcal{SE}_i$.
- (2) The withdraw interface is called by a secure element $\mathcal{SE}_i$. First $\mathcal{F}_{\text{ASEC}}$ asks the simulator $\mathcal{S}$ permission to continue through the message $(\text{withdraw}, \text{sid})$. This models the fact that, in the real protocol, the adversary controls the network and therefore is able to stop or delay withdraw requests. Second, when the host or the bank are corrupt, $\mathcal{F}_{\text{ASEC}}$ also asks permission to continue through the message $(\text{withdraw}, \text{sid}, \mathcal{SE}_i)$, this time including the identity of the secure element. Finally, $\mathcal{F}_{\text{ASEC}}$ creates a new wallet for $\mathcal{U}_i$, initializes the coin counter for that wallet and informs the secure element and the bank that the wallet is created. We note that, for simplicity, all the wallets have the same number N of coins.
- (3) The spend interface is called by a secure element $\mathcal{SE}_i$ on input a wallet w and a merchant $\mathcal{M}_j$. First, $\mathcal{F}_{\text{ASEC}}$ checks that the wallet exists and that it has not run out of coins. If the secure element $\mathcal{SE}_i$ is corrupt, $\mathcal{F}_{\text{ASEC}}$ allows $\mathcal{SE}_i$ to choose which coin should be spent or double-spent. Second, $\mathcal{F}_{\text{ASEC}}$ asks the simulator $\mathcal{S}$ permis-

sion to continue. Third, if the host $\mathcal{H}_i$ is corrupt, $\mathcal{F}_{\text{ASEC}}$ sends the identity $\mathcal{SE}_i$ to the simulator $\mathcal{S}$ and $\mathcal{S}$ tells $\mathcal{F}_{\text{ASEC}}$ whether it allows the spend request to proceed ($b \neq 0$) and whether the identity $\mathcal{SE}_i$ should not be revealed to the merchant ($c \neq 0$). Then, $\mathcal{F}_{\text{ASEC}}$ creates a new transaction identifier id or receives id from a corrupt merchant. If the coin $(\mathcal{U}_i, w, cn)$ was never spent, $\mathcal{F}_{\text{ASEC}}$ creates a new coin pseudonym P and stores a new row in Table I, else $\mathcal{F}_{\text{ASEC}}$ adds a new tuple $[id, \mathcal{M}, 0]$ to the existing row for coin $(\mathcal{U}_i, w, cn)$ in Table I. Finally, $\mathcal{F}_{\text{ASEC}}$ sends the pseudonym P and the transaction identifier id to the merchant, along with the identity $\mathcal{SE}_i$ if the corrupt host requested that. $\mathcal{F}_{\text{ASEC}}$ sends P and id to $\mathcal{SE}_i$.

- (4) The deposit interface is called by a merchant $\mathcal{M}_j$ on input a coin pseudonym P and a transaction identifier id . First, $\mathcal{F}_{\text{ASEC}}$ checks that the merchant $\mathcal{M}_j$ has received a coin with a pseudonym P and a transaction identifier id . Second, $\mathcal{F}_{\text{ASEC}}$ asks the simulator $\mathcal{S}$ permission to continue. Then $\mathcal{F}_{\text{ASEC}}$ checks if the coin with pseudonym P and a transaction identifier id has already been deposited, and in that case, deposits the coin with the bank and alerts the bank of a double-depositing. Else, $\mathcal{F}_{\text{ASEC}}$ checks if the coin with pseudonym P was already deposited under other transaction identifiers, and if it is the case, sends it to the bank, alerts the bank of a double spending, and marks the coin as deposited. Else, if none of the conditions applies, $\mathcal{F}_{\text{ASEC}}$ marks the coin as deposited and sends it to the bank.
- (5) The accuse interface is called by any party $\mathcal{P}$ on input a user $\mathcal{U}_i$, a coin pseudonym P and transaction identifiers id and $\hat{id}$. $\mathcal{F}_{\text{ASEC}}$ checks whether the coin with pseudonym P belongs to $\mathcal{U}_i$ and was deposited with transaction identifiers id and $\hat{id}$. In that case, $\mathcal{F}_{\text{ASEC}}$ tells $\mathcal{P}$ that $\mathcal{U}_i$ is guilty, and otherwise that $\mathcal{U}_i$ is not guilty of double-spending.

$\mathcal{F}_{\text{ASEC}}$ guarantees the same security properties as traditional E-Cash schemes in which $\mathcal{U}$ is not divided into two entities. It guarantees balance correctness, anonymity, unlinkability, exculpability and identification of double-spenders. Additionally, the ideal protocol provides extra security properties to protect $\mathcal{SE}_i$ from a malicious $\mathcal{H}_i$: withdraw authorization, withdraw completion, spend authorization and spend completion. A short description of all these properties follows:

- *Balance correctness.* No coalition of malicious merchants $\mathcal{M}$ and users $\mathcal{U}$ can deposit more coins than the ones they previously withdrew. When $\mathcal{M}$ deposits a coin through $(\text{deposit}, sid, P, id)$, $\mathcal{F}_{\text{ASEC}}$ checks whether the coin P was already deposited. If it was deposited with the same transaction identifier id , then $\mathcal{F}_{\text{ASEC}}$ informs the bank that the merchant is double-depositing the coin. If it was deposited with a different transaction identifier, $\mathcal{F}_{\text{ASEC}}$ informs the bank that the user double-spent the coin.
- *Anonymity and Unlinkability.* No coalition of malicious $\mathcal{B}$ and $\mathcal{M}$ can revoke $\mathcal{U}$'s anonymity, link the spending of a coin to its withdrawal, or link two spendings to the same entity. $\mathcal{F}_{\text{ASEC}}$ sends $\mathcal{M}$ and $\mathcal{B}$ a random value P and transaction identifier id to identify the coin when it is spent and deposited. P and id reveal no information about the wallet the coin belongs to or the user that spends the coin. $\mathcal{U}$'s anonymity can be revoked if $\mathcal{U}$ double-spends a coin. A malicious $\mathcal{H}$ can also reveal $\mathcal{U}$'s identity.
- *Exculpability.* No coalition of malicious $\mathcal{B}$ and $\mathcal{M}$ can falsely accuse $\mathcal{U}$ of double spending. When a party $\mathcal{P}$ sends $(\text{accuse}, sid, \mathcal{U}_i, P, id, \hat{id})$, $\mathcal{F}_{\text{ASEC}}$ checks whether the coin P belongs to $\mathcal{U}_i$ and has been deposited twice with transaction identifiers id and $\hat{id}$ and only in that case tells $\mathcal{P}$ that the user is guilty.
- *Identification of double-spenders.* If a coin P is deposited at the bank with a transaction identifier id and the coin P was deposited before with a different transaction identifier, $\mathcal{F}_{\text{ASEC}}$ sends $\mathcal{B}$ the identity of the user that double-spent the coin.

- *Withdraw Authorization.* $\mathcal{H}_i$ cannot withdraw wallets without $\mathcal{SE}_i$'s authorization. $\mathcal{F}_{\text{ASEC}}$ only accepts withdraw requests from the secure element $\mathcal{SE}_i$.
- *Withdraw Completion.* $\mathcal{H}_i$ cannot prevent $\mathcal{SE}_i$ from obtaining a wallet after it was withdrawn. $\mathcal{F}_{\text{ASEC}}$ informs both $\mathcal{SE}_i$ and $\mathcal{B}$ that a wallet is withdrawn by sending the messages $(\text{withdrawend}, \text{sid}, w_i)$ and $(\text{withdrawend}, \text{sid}, \mathcal{SE}_i)$ respectively. Therefore, $\mathcal{F}_{\text{ASEC}}$ ensures that, if an honest $\mathcal{B}$ receives confirmation that a wallet was withdrawn by a secure element $\mathcal{SE}_i$, then $\mathcal{SE}_i$ receives that wallet. Nevertheless, a corrupt $\mathcal{H}_i$ can prevent $\mathcal{SE}_i$ from withdrawing wallets, but in that case the bank does not receive any confirmation that a wallet was withdrawn.
- *Spend Authorization.* $\mathcal{H}_i$ cannot spend a coin without $\mathcal{SE}_i$'s authorization. $\mathcal{F}_{\text{ASEC}}$ only accepts spend requests from the secure element $\mathcal{SE}_i$.
- *Spend Completion.* $\mathcal{H}_i$ cannot prevent a coin spent by $\mathcal{SE}_i$ from being received by the merchant $\mathcal{M}_j$. $\mathcal{F}_{\text{ASEC}}$ informs both $\mathcal{SE}_i$ and $\mathcal{M}_j$ that the spend operation is completed. Nevertheless, $\mathcal{H}_i$ can prevent $\mathcal{SE}_i$ from spending coins. In this case, $\mathcal{M}_j$ does not receive a coin and $\mathcal{SE}_i$ does not receive any confirmation that the coin was spent to $\mathcal{M}_j$.

3. CONSTRUCTION OF ANONYMOUS SPLIT E-CASH

In this section we present a construction of our system based on the compact E-Cash scheme by Camenisch *et al.* [Camenisch et al. 2005]. The reason to choose this protocol as basis is five-fold. First of all, only a limited number of interactions with the bank are required. Second, it fulfills the first four properties of the ideal functionality presented in Sect. 2.2. In other words, the system is designed such that even after corruption of a user, the security properties of bank and merchant remain the same. Third, the compact E-Cash scheme is both memory and time efficient. Fourth, it employs commonly known cryptographic primitives, which is of importance when pursuing an implementation on commercially available devices. And last of all, the original scheme has a potential to be split up into a host and a secure element part.

We give in the following a detailed description of our system. Recall that the system contains three types of players: a bank $\mathcal{B}$, merchants $\mathcal{M}$ and users $\mathcal{U}$. We distinguish operations performed by a user $\mathcal{U}$ depending on whether they are executed on the (untrusted) host platform $\mathcal{H}$ or on the (trusted) secure element $\mathcal{SE}$. The protocol consists of five phases: setup, withdraw, spend, deposit and accuse. For brevity, when a party communicates with another party, we indicate whether this communication is handled through $\mathcal{F}_{\text{SMT}}$ or $\mathcal{F}_{\text{ASMT}}$ without showing how these functionalities are invoked.

3.1. Setup Phase

On input $(\text{setup}, \text{sid})$, $\mathcal{B}$ executes the following steps:

- (1) Continue only if $\text{sid} = (\mathcal{B}, \text{sid}')$ and if a key pair (sk_B, pk_B) is not stored.
- (2) Generate a special RSA modulus $n = ab$ with safe primes $a = 2a' + 1$ and $b = 2b' + 1$, such that a, a', b, b' are all prime numbers and the resulting n has $2k$ bits, where k is the security parameter.
- (3) Pick generators $Z, Q, V_1, \dots, V_5$ of QR_n and set $(n, Z, Q, V_1, \dots, V_5)$ as a public key of the Camenisch-Lysyanskaya signature scheme.
- (4) Generate a safe prime $p = 2q + 1$, where q is also a prime, and pick a generator g of the subgroup $\mathbb{G}_q$ of order q of $\mathbb{Z}_p^*$.
- (5) Set the key pair of the bank as $(sk_B, pk_B) = (a, (q, g, n, Z, Q, V_1, \dots, V_5))$.
- (6) Send $(\text{register}, \text{sid}, pk_B)$ to $\mathcal{F}_{\text{REG}}$ and output $(\text{setupend}, \text{sid})$.

3.2. Withdraw Phase

The (withdraw, sid) command is issued by the human user through the dedicated interface to $\mathcal{SE}$, e.g. by selecting the desired amount of coins to be retrieved using the increase/decrease buttons and confirming the transaction by pushing the withdraw button. On input (withdraw, sid), $\mathcal{SE}$ executes the following steps:

Step 1. $\mathcal{H}$ and $\mathcal{SE}$ jointly compute a commitment A to the wallet secrets (u, s, t) .

- (1) Continue only if $sid = (\mathcal{B}, sid')$.
- (2) If pk_B is not stored, $\mathcal{SE}$ sends (retrieve, sid) to $\mathcal{F}_{REG}$ and waits for a message (retrieve, sid, pk_B) from $\mathcal{F}_{REG}$.
- (3) If (sk_U, pk_U) is not stored, $\mathcal{SE}$ picks a secret key $sk_U = u \in_R \mathbb{Z}_q$, computes a public key $pk_U = g^u$ and sends (register, $\mathcal{U}, pk_U$) to $\mathcal{F}_{REG}$.
- (4) $\mathcal{SE}$ randomly picks $v' \in_R [1 \dots \lfloor \frac{n}{4} \rfloor]$ and $s' \in_R \mathbb{Z}_q^*$, computes $A'' = V_1^u V_2^{s'} Q^{v'} \mod n$, and sends A'' to $\mathcal{H}$ through $\mathcal{F}_{SMT}$.
- (5) $\mathcal{H}$ picks $t \in_R \mathbb{Z}_q^*$, computes $A' = A'' V_3^t \mod n$, and sends A' to $\mathcal{B}$ through $\mathcal{F}_{SMT}$.
- (6) If pk_U is not stored, $\mathcal{B}$ sends (retrieve, $\mathcal{U}$) to $\mathcal{F}_{REG}$ and waits for a message (retrieve, $\mathcal{U}, pk_U$) from $\mathcal{F}_{REG}$.
- (7) $\mathcal{B}$ picks $r' \in_R \mathbb{Z}_q^*$, computes $A = A' V_2^{r'} \mod n$, and sends r' and a nonce $n_i \in_R \{0, 1\}^{l_H}$ to $\mathcal{H}$ through $\mathcal{F}_{SMT}$.
- (8) $\mathcal{H}$ computes $A = A' V_2^{r'} = V_1^u V_2^s V_3^t Q^{v'} \mod n$, and sends (A, r') to $\mathcal{SE}$ through $\mathcal{F}_{SMT}$.
- (9) $\mathcal{SE}$ computes $s = s' + r'$ and stores (s, A) .

Step 2. $\mathcal{H}$, $\mathcal{SE}$, and $\mathcal{B}$ engage in a protocol to obtain $\mathcal{B}$'s blind signature on (u, s, t) . From points 1 to 4, $\mathcal{H}$ and $\mathcal{SE}$ jointly compute the following zero-knowledge proof of knowledge:

$$\text{ZKPK} \{ (u, s, t, v') : pk_u = g^u \wedge A = V_1^u V_2^s V_3^t Q^{v'} \wedge u, s, t \in \{0, 1\}^{l_x + l_\phi + l_H} \wedge v' \in \{0, 1\}^{l_y + l_\phi + l_H} \},$$

which is verified by $\mathcal{B}$ in point 5 below.

- (1) $\mathcal{SE}$ randomly picks $r_u, r_s \in_R \{0, 1\}^{l_x + l_\phi + l_H}$ and $r_{v'} \in_R \{0, 1\}^{l_y + l_\phi + l_H}$, computes $T'_A = V_1^{r_u} V_2^{r_s} Q^{r_{v'}} \mod n$ and $T_U = g^{r_u} \mod p$, and sends (T'_A, T_U) to $\mathcal{H}$ through $\mathcal{F}_{SMT}$.
- (2) $\mathcal{H}$ picks $r_t \in_R \{0, 1\}^{l_x + l_\phi + l_H}$, computes $T_A = T'_A V_3^{r_t} \mod n$ and sequentially computes $S_x = n \| V_1 \| V_2 \| V_3 \| Q \| A$ and $c' = H(S_x \| T_A \| T_U \| n_i)$, and sends c' to $\mathcal{SE}$ through $\mathcal{F}_{SMT}$.
- (3) $\mathcal{SE}$ picks $n_t \in \{0, 1\}^{l_\phi}$, computes $c = H(c' \| n_t)$, calculates $s_u = r_u - cu$, $s_{v'} = r_{v'} - cv'$ and $s_s = r_s - cs$, and sends $(c, n_t, s_u, s_s, s_{v'})$ to $\mathcal{H}$ through $\mathcal{F}_{SMT}$.
- (4) $\mathcal{H}$ computes $s_t = r_t - ct$ and sends $(c, n_t, s_u, s_{v'}, s_s, s_t)$ to $\mathcal{B}$ through $\mathcal{F}_{SMT}$.
- (5) $\mathcal{B}$ checks whether $s_u, s_s, s_t \in \{0, 1\}^{l_x + l_H + l_\phi}$ and $s_{v'} \in \{0, 1\}^{l_y + l_H + l_\phi}$, computes the values $\tilde{T}_A = A^c V_1^{s_u} V_2^{s_s} V_3^{s_t} Q^{s_{v'}} \mod n$ and $\tilde{T}_U = pk_u^c g^{s_u} \mod p$, and checks if $c = H(H(S_x \| \tilde{T}_A \| \tilde{T}_U \| n_i) \| n_t)$.
- (6) $\mathcal{B}$ computes $e \in_R [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$, $v'' \in \{0, 1\}^{l_v}$ and $d = (\frac{Z}{AQ^{v''}})^{\frac{1}{e}} \mod n$, and sends (d, e, v'') to $\mathcal{H}$ through $\mathcal{F}_{SMT}$.
- (7) $\mathcal{H}$ checks whether $e \in [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$ and $Z = AQ^{v''} d^e \mod n$, stores $(J = 2^L, d, e)$, and sends (d, e, v'', t, J) to $\mathcal{SE}$ through $\mathcal{F}_{SMT}$.
- (8) $\mathcal{SE}$ calculates $v = v' + v''$, checks the signature (d, e, v) , and stores the wallet $W = (u, s, t, \sigma_B(u, s, t), J)$.

Step 3. $\mathcal{SE}$ sends a confirmation message directly to $\mathcal{B}$ through $\mathcal{F}_{\text{SMT}}$, increments a counter w of the number of wallets it stores, initializes to 0 a counter J of the coins spent in wallet w , and outputs (withdrawnd, sid, w). $\mathcal{B}$ withdraws 2^L coins from $\mathcal{U}$'s account and outputs (withdrawnd, $sid, \mathcal{SE}$). If this confirmation message is not received, $\mathcal{B}$ blocks $\mathcal{U}$'s account. This prevents a malicious $\mathcal{H}$ from withdrawing more wallets that later on are not handed to $\mathcal{SE}$.

3.3. Spend Phase

On input (spend, $sid, w, \mathcal{M}$), $\mathcal{SE}$ starts the following protocol which we subdivide into 3 steps.

Step 1. $\mathcal{SE}$ and $\mathcal{H}$ compute a coin's serial number S and double-spending tag T , along with commitments needed to check the correctness of both S and T .

- (1) $\mathcal{SE}$ continues only if $sid = (\mathcal{B}, sid')$ and if it stores a wallet w .
- (2) $\mathcal{SE}$ increments the counter J for wallet w and continues only if $J \leq N$, where N is the number of coins in a wallet.
- (3) $\mathcal{SE}$ picks $\rho_6 \in_R [1 \dots \lfloor \frac{n}{4} \rfloor]$, computes $B = V_4^u V_5^{\rho_6} \mod n$ and $S = g^{1/(J+s)} = g^{(J+s)^{-1}} \mod p$, and sends (B, S, J) to $\mathcal{H}$ through $\mathcal{F}_{\text{SMT}}$.
- (4) $\mathcal{H}$ sends a spend request to $\mathcal{M}$ through $\mathcal{F}_{\text{ASMT}}$.
- (5) If (sk_M, pk_M) are not stored, $\mathcal{M}$ picks a secret key $sk_M = v \in_R \mathbb{Z}_q$, computes a public key $pk_M = g^v$, stores (sk_M, pk_M) , and sends (register, $\mathcal{M}, pk_M$) to $\mathcal{F}_{\text{REG}}$.
- (6) $\mathcal{M}$ picks a unique transaction identifier $id_{trans} \in_R \{0, 1\}^{l_H}$ and a nonce $n_i \in_R \{0, 1\}^{l_H}$ and sends it to $\mathcal{H}$ through $\mathcal{F}_{\text{ASMT}}$.
- (7) If pk_M is not stored, $\mathcal{H}$ sends (retrieve, $\mathcal{M}$) to $\mathcal{F}_{\text{REG}}$ and waits for a message (retrieve, $\mathcal{M}, pk_M$) from $\mathcal{F}_{\text{REG}}$.
- (8) $\mathcal{H}$ computes $R = H(pk_M || id_{trans})$ and w_i for $i = 1 \dots 4$ such that $\sum_{i=1}^4 w_i = J_1^2 - (J - J_0)^2$, picks $\rho_i \in_R [1 \dots \lfloor \frac{n}{4} \rfloor]$ for $i = 0 \dots 4$, computes $T = pk_u g^{R/(J+t)} \mod p$ and $W_i = V_4^{w_i} V_5^{\rho_i} \mod n$ for $i = 1 \dots 4$, computes $A_J = V_4^J V_5^{\rho_0} \mod n$, and stores $(S, T, A_J, B, W_1, \dots, W_4)$.

Step 2. $\mathcal{SE}$ and $\mathcal{H}$ compute jointly the following zero-knowledge proof of knowledge ϕ and send it to $\mathcal{M}$.

$$\text{ZKPK} \{ \begin{array}{l} (u, s, t, J, v_B, e, \alpha, w_1, \dots, w_4, \rho_0, \dots, \rho_7) : \\ Z = Q^{v_B} V_1^u V_2^s V_3^t d_B^e \wedge A_J = V_4^J V_5^{\rho_0} \wedge g = S^{(J+s)} \wedge \\ W_i = V_4^{w_i} V_5^{\rho_i}; i = 1 \dots 4 \wedge B = V_4^u V_5^{\rho_6} \wedge \\ V_4^{(J_1^2 - J_0^2)} = (A_J V_4^{-2J_0})^J (\prod_{i=1}^4 W_i^{w_i}) V_5^{\rho_5} \wedge \\ 1 = B^J B^t V_4^{-\alpha} V_5^{\rho_7} \wedge u, s, t \in \{0, 1\}^{l_x + l_H + l_\phi} \wedge \\ g^R = T^{(J+t)} g^{-\alpha} \wedge e \in [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}] \end{array} \},$$

with $\alpha = u(J + t)$.

(1) $\mathcal{SE}$ executes the following:

- (a) Pick randomly $r_u, r_s \in \{0, 1\}^{l_x + l_H + l_\phi}$, $r_{\rho_6} \in \{0, 1\}^{l_y + l_H + l_\phi}$, and $r_{v_B} \in \{0, 1\}^{l_e + l_n + l_\phi + 1}$.
- (b) Pick $r \in \{0, 1\}^{l_n + l_\phi}$.
- (c) Compute $v_B = v + er$ and $d_B = dQ^{-r} \mod n$ such that (d_B, e, v_B) is a randomized signature.
- (d) Compute $T'_Z = V_1^{r_u} V_2^{r_s} Q^{r_{v_B}} \mod n$.
- (e) Compute $T_B = V_4^{r_u} V_5^{r_{\rho_6}} \mod n$.
- (f) Compute $T'_g = S^{r_s} \mod p$.

- and sends (T'_Z, T'_B, T'_g, d_B) to $\mathcal{H}$ through $\mathcal{F}_{\text{SMT}}$.
- (2) $\mathcal{H}$ executes the following:
- (a) Pick $r_t \in \{0, 1\}^{l_x + l_H + l_\phi}$, $r_e \in \{0, 1\}^{l_e + l_H + l_\phi}$, and $r_J \in \{0, 1\}^{l + l_H + l_\phi}$.
 - (b) Pick $r_{\rho_i} \in \{0, 1\}^{l_y + l_H + l_\phi}$, for $i = [0, \dots, 5, 7]$.
 - (c) Pick $r_{w_i} \in \{0, 1\}^{l + l_\phi}$, for $i = [1, \dots, 4]$.
 - (d) Compute $T_Z = T'_Z V_3^{r_t} d_B^{r_e} \bmod n$ and $T_{A_J} = V_4^{r_J} V_5^{r_{\rho_0}} \bmod n$.
 - (e) Compute $T_g = T'_g S^{r_J} \bmod p$.
 - (f) Compute $T_{W_i} = V_4^{r_{w_i}} V_5^{r_{\rho_i}} \bmod n$, for $i = [1, \dots, 4]$.
 - (g) Compute $T_{V_4^{J_1^2 - J_0^2}} = (A_J V_4^{-2J_0})^{r_J} W_i^{r_{w_i}} V_5^{r_{\rho_5}} \bmod n$, for $i = [1, \dots, 4]$.
 - (h) Compute $T_1 = B^{r_J + r_t} V_4^{-r_\alpha} V_5^{r_{\rho_7}} \bmod n$.
 - (i) Compute $T_{g^R} = T^{r_J + r_t} g^{-r_\alpha} \bmod p$.
 - (j) Compute $S_C = Q \| V_i \| d_B \| g \| W_k \| A_J \| B \| T$, for $i = [1, \dots, 4]$ and $k = [1, \dots, 5]$.
 - (k) Compute $T_C = T_Z \| T_{A_J} \| T_g \| T_{W_i} \| T_{V_4^{J_1^2 - J_0^2}} \| T_1 \| T_B \| T_{g^R}$, for $i = [1, \dots, 4]$.
- $\mathcal{H}$ sends $c' = H(S_C \| T_C \| n_i)$ to $\mathcal{SE}$ through $\mathcal{F}_{\text{SMT}}$.
- (3) $\mathcal{SE}$ executes the following:
- (a) Pick $n_t \in \{0, 1\}^{l_H}$.
 - (b) Compute $c = H(c' \| S \| n_t)$.
 - (c) Compute $s_u = r_u - cu$, and $s_{v_B} = r_{v_B} - cv_B$.
 - (d) Compute $s_s = r_s - cs$, and $s_{\rho_6} = r_{\rho_6} - c\rho_6$.
- and sends $(c, n_t, s_u, s_s, s_{v_B}, s_{\rho_6})$ to $\mathcal{H}$ through $\mathcal{F}_{\text{SMT}}$.
- (4) $\mathcal{H}$ executes the following:
- (a) Compute $s_t = r_t - ct$, $s_e = r_e - ce$, and $s_J = r_J - cJ$.
 - (b) Compute $s_{\rho_i} = r_{\rho_i} - c\rho_i$ for $i = [0, \dots, 5, 7]$.
 - (c) Compute $s_{w_j} = r_{w_j} - cw_j$, for $i = [1, \dots, 4]$.
 - (d) Compute $s_\alpha = r_\alpha - c\alpha$.
 - (e) Set $s_{all} = (s_u, s_s, s_t, s_{v_B}, s_e, s_J, s_{\rho_i}, s_\alpha, s_{w_k})$, for $i = [0, \dots, 7]$ and $k = [1, \dots, 4]$.
- and sends (d_B, s_{all}, c, n_t) and $(S, T, A_J, B, W_1, \dots, W_4)$ (computed in Step 1) to $\mathcal{M}$ through $\mathcal{F}_{\text{ASMT}}$.
- (5) $\mathcal{M}$ verifies the proof as follows:
- (a) Compute $\tilde{T}_Z = V_1^{s_u} V_2^{s_s} V_3^{s_t} d_B^{s_e} Z^c Q^{s_{v_B}} \bmod n$.
 - (b) Compute $\tilde{T}_{A_J} = V_4^{s_J} V_5^{s_{\rho_0}} A_J^c \bmod n$.
 - (c) Compute $\tilde{T}_g = S^{s_J + s_s} g^c \bmod p$.
 - (d) Compute $\tilde{T}_{W_i} = V_4^{s_{w_i}} V_5^{s_{\rho_i}} W_i^c \bmod n$, for $i = [1, \dots, 4]$.
 - (e) Compute $\tilde{T}_{V_4^{J_1^2 - J_0^2}} = (A_J V_4^{-2J_0})^{s_J} W_i^{s_{w_i}} V_5^{s_{\rho_5}} V_4^{(J_1^2 - J_0^2)c} \bmod n$, for $i = [1, \dots, 4]$.
 - (f) Compute $\tilde{T}_B = V_4^{s_u} V_5^{s_{\rho_6}} B^c \bmod n$ and $\tilde{T}_1 = B^{s_J + s_t} V_4^{-s_\alpha} V_5^{s_{\rho_7}} \bmod n$.
 - (g) Compute $\tilde{T}_{g^R} = T^{(s_J + s_t)} g^{-s_\alpha} g^{Rc} \bmod p$.
 - (h) Compute $\tilde{S}_C = Q \| V_i \| d_B \| g \| W_k \| A_J \| B \| T$, for $i = [1, \dots, 5]$ and $k = [1, \dots, 4]$.
 - (i) Compute $\tilde{T}_c = \tilde{T}_Z \| \tilde{T}_A \| \tilde{T}_g \| \tilde{T}_{W_i} \| \tilde{T}_{V_4^{J_1^2 - J_0^2}} \| \tilde{T}_B \| \tilde{T}_1 \| \tilde{T}_{g^R}$, for $i = [1, \dots, 4]$.
 - (j) Check $c = H(H(\tilde{S}_c \| \tilde{T}_c \| n_i) \| S \| n_t)$.
 - (k) Check $s_u, s_s, s_t \in \{0, 1\}^{l_x + l_H + l_\phi}$ and $s_e \in \{0, 1\}^{l_e + l_H + l_\phi}$.

$\mathcal{M}$ stores the coin (S, R, T, ϕ) .

Step 3. $\mathcal{M}$ signs (S, R) using sk_M , sends the signature to $\mathcal{H}$ through $\mathcal{F}_{ASMT}$ and outputs $(spendend, sid, S, R)$. $\mathcal{H}$ verifies the signature, sends the signature, id_{trans} and R to $\mathcal{SE}$ through $\mathcal{F}_{SMT}$ and outputs $(spendend, sid, S, R)$. If pk_M is not stored, $\mathcal{SE}$ retrieves pk_M through $\mathcal{F}_{REG}$, verifies the signature, and checks that R contains pk_M . If the checks verify, $\mathcal{SE}$ outputs $(spendend, sid, S, R)$. If $\mathcal{SE}$ does not output anything, the human user knows that either the host prevented the merchant from retrieving the coin, or the merchant does not acknowledge reception of the coin.

3.4. Deposit Phase

On input $(deposit, sid, S, R)$, $\mathcal{M}$ sends the coin (S, R, T, ϕ) to $\mathcal{B}$ through $\mathcal{F}_{SMT}$. $\mathcal{B}$ verifies ϕ . If S and R are not fresh, $\mathcal{B}$ outputs $(depositend, sid, \mathcal{M}, S, R, ddepositalert)$. If S is not fresh, let (S, R, T, ϕ) and (S, R', T', ϕ') be two coins with the same S . Identify the double-spender as follows

$$\left(\frac{T_2^{R_1}}{T_1^{R_2}}\right)^{(R_1-R_2)^{-1}} = \left(\frac{g^{uR_1+R_1R_2(J_2+S)}}{g^{uR_2+R_1R_2(J_1+S)}}\right)^{(R_1-R_2)^{-1}} = g^u$$

$\mathcal{B}$ retrieves the identity $\mathcal{U}$ associated with public key g^u and outputs $(depositend, sid, \mathcal{M}, \mathcal{U}, S, R, dspentalert)$. If S is fresh, $\mathcal{B}$ stores (S, R, T, ϕ) and outputs the message $(depositend, sid, \mathcal{M}, S, id)$ to $\mathcal{B}$.

3.5. Accuse Phase

On input $(accuse, sid, \mathcal{U}, S, R, R')$, $\mathcal{P}$ retrieves two coins (S, R, T, ϕ) and (S, R', T', ϕ') with the same S . $\mathcal{P}$ verifies ϕ and ϕ' and checks that $R \neq R'$. If it is the case, the value g^u can be computed as in the deposit phase. If the user $\mathcal{U}$ is associated with the public key g^u , $\mathcal{P}$ outputs $(accuseend, sid, guilty)$. Otherwise, it outputs $(accuseend, sid, notguilty)$.

4. SECURITY ANALYSIS OF OUR CONSTRUCTION

To prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{ASEC}$, we have to show that for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ there exists a simulator S , such that $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and the protocol in the real world or with S and $\mathcal{F}_{ASEC}$. The simulator thereby plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{ASEC}$ for all corrupt parties in the ideal world.

The security analysis where both the secure element $\mathcal{SE}$ and the host $\mathcal{H}$ are corrupt follows the security analysis given in [Camenisch et al. 2005] for the case where the user $\mathcal{U}$ is corrupt. Similarly, the security analysis where both the secure element $\mathcal{SE}$ and the host $\mathcal{H}$ are honest, but some merchants $\mathcal{M}$ and/or the bank $\mathcal{B}$ are corrupt, follows the security analysis given in [Camenisch et al. 2005] when $\mathcal{U}$ is honest. We note that our protocol modifies the way $\mathcal{U}$ does computations internally by splitting $\mathcal{U}$ into the secure element $\mathcal{SE}$ and the host $\mathcal{H}$, but the view of the protocol given to the merchants and the bank is equivalent to the situation where $\mathcal{U}$ is not split up.

Moreover, the security analysis when $\mathcal{SE}$ is corrupt and $\mathcal{H}$ is honest is not of practical interest. In our model, we assume tamper-resistant secure elements $\mathcal{SE}$ and therefore we assume that an adversary able to corrupt the secure element also corrupts $\mathcal{H}$.

We must therefore analyse the security of our protocol when the secure elements $\mathcal{SE}_1, \dots, \mathcal{SE}_I$ are honest but (a subset of) the hosts $\mathcal{H}_1, \dots, \mathcal{H}_I$ are corrupt, and the bank $\mathcal{B}$ and (a subset of) the merchants $\mathcal{M}_1, \dots, \mathcal{M}_J$ are either honest or corrupt. As described in Section 2, when $\mathcal{H}$ is corrupt, the anonymity property does not hold anymore for the respective secure element. Therefore, we can simplify our analysis by considering only one secure element $\mathcal{SE}$ and one host $\mathcal{H}$. We note that considering the

collusion of several corrupted hosts against the bank and the merchant is not needed since this analysis, as mentioned above, is described in [Camenisch et al. 2005] for a collusion of fully corrupt users.

Because of space limitations, we give a detailed security proof for the case where the host is corrupt but the secure element, the bank and the merchants are honest. The case where the host and some merchants are corrupt is similar, i.e., the proof that a corrupt host and merchant cannot withdraw and spend coins without the secure element's authorization follows the proof where only the host is corrupt. The difference is that a corrupt merchant may not acknowledge reception of the coin in Step 3 of the spend protocol, in which case the human user detects that no confirmation was received. The case where the host and the bank are corrupt is also similar. The proof that a corrupt bank cannot accuse an honest user of coin double-spending is also based on the inability of the adversary to spend coins without the secure element's authorization.

THEOREM 4.1. *When the host is corrupt, our e-cash scheme realizes $\mathcal{F}_{\text{ASEC}}$ in the $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$ and $\mathcal{F}_{\text{ASMT-hybrid}}$ model and in the random oracle model under the discrete logarithm assumption, the hiding property of the Damgaard-Fujisaki [Damgård and Fujisaki 2002] commitment scheme, the unforgeability of the Camenisch-Lysyanskaya [Camenisch and Lysyanskaya 2001] signature scheme and the pseudorandomness of the Dodis-Yampolskiy [Dodis and Yampolskiy 2005] pseudorandom function.*

In our proof, we need to rewind the adversary. Therefore, we do not prove security in the universal composability framework.

We will show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between his view of the real world protocol and his view of the ideal world protocol with non-negligible probability. The first game corresponds to the real world protocol, while the last game corresponds to the ideal world protocol. We denote by $\Pr[\mathbf{Game} \ i]$ the probability that the environment distinguishes **Game** i from the real world protocol, and for each game **Game** i , we show that $|\Pr[\mathbf{Game} \ i] - \Pr[\mathbf{Game} \ i-1]|$ is negligible.

— **Game** 0: This game corresponds to the execution of the real-world protocol. In **Game** 0, the random oracle queries sent by the adversary are replied with consistent random values, i.e., equal queries are answered with the same value. Therefore, $\Pr[\mathbf{Game} \ 0] = 0$.

— **Game** 1: **Game** 1 follows **Game** 0, except that **Game** 1 replaces Step 2.1, Step 2.2 and Step 2.3 of the withdraw phase by the following:

- (1) **Game** 1 picks a random challenge c , picks random $s_u, s_s \in_R \{0, 1\}^{l_x + l_\phi + l_H}$ and $s_{v'} \in_R \{0, 1\}^{l_y + l_\phi + l_H}$, and computes $T'_A = (A/V_3^t)^c V_1^{s_u} V_2^{s_s} Q^{s_{v'}} \bmod n$ and $T_U = (pk_U)^c g^{s_u} \bmod p$. (We note that **Game** 1 can calculate $V_3^t = A'/A''$ in Step 1.7 of the withdraw phase.)
- (2) A random oracle query $(S_x \| T_A \| T_U \| n_i)$ is stored and answered with a consistent random value c' .
- (3) **Game** 1 picks $n_t \in \{0, 1\}^{l_\phi}$, assigns c to the random oracle query $(c' \| n_t)$, and sends $(c, n_t, s_u, s_s, s_{v'})$.

Game 1 shows that, in the random oracle model, the secure element $\mathcal{SE}$ is able to simulate the proof computed in Step 2 of the withdrawal protocol, i.e., the secure element can compute the proof without knowledge of the secret values (u, s, v') . We show that $|\Pr[\mathbf{Game} \ 1] - \Pr[\mathbf{Game} \ 0]| = 0$.

PROOF. The values (T'_A, T_U) , as well as $(n_t, s_u, s_s, s_{v'})$, are equally distributed in **Game** 1 and **Game** 0. Additionally, the zero-knowledge proof of knowl-

edge sent to the bank in Step 2 of **Game 1** still verifies correctly. For T_U , it is clear that the value computed in **Game 1** equals the verification value. For T_A , we have that, if the corrupt host followed the protocol specification, $T_A = T'_A V_3^{r_t} = (A/V_3^t)^c V_1^{s_u} V_2^{s_s} Q^{s_{v'}} V_3^{r_t}$, and the verifier of the proof computes $\tilde{T}_A = A^c V_1^{s_u} V_2^{s_s} V_3^{r_t} Q^{s_{v'}} \bmod n = (A)^c V_1^{s_u} V_2^{s_s} V_3^{r_t - ct} Q^{s_{v'}} \bmod n = (A)^c V_1^{s_u} V_2^{s_s} Q^{s_{v'}} V_3^{-ct} V_3^{r_t} = (A/V_3^t)^c V_1^{s_u} V_2^{s_s} Q^{s_{v'}} V_3^{r_t}$. $\square$

— **Game 2:** **Game 2** follows **Game 1**, except that **Game 2** replaces Step 2.1 of the spend phase by the following:

- (1) Pick randomly $s_u, s_s \in \{0, 1\}^{l_x + l_H + l_\phi}$, $s_{\rho_6} \in \{0, 1\}^{l_y + l_H + l_\phi}$, and $s_{v_B} \in \{0, 1\}^{l_e + l_n + l_\phi + 1}$ and a random oracle response c .
- (2) Pick $r \in \{0, 1\}^{l_n + l_\phi}$.
- (3) Compute $d_B = Q^r \bmod n$.
- (4) Compute $T'_Z = (AQ^{v''}/V_3^t)^c V_1^{s_u} V_2^{s_s} Q^{s_{v_B}} \bmod n$. (We note that **Game 1** can calculate $V_3^t = A'/A''$ in Step 1.7 of the withdraw phase.)
- (5) Compute $T_B = (B)^c V_4^{s_u} V_5^{s_{\rho_6}} \bmod n$.
- (6) Compute $T'_g = (g)^c S^{s_s} \bmod p$.

In Step 2.2, the random oracle queries $(S_C \| T_C \| n_i)$ are replied with a consistent random value. **Game 2** also replaces Step 2.3 of the spend phase by the following:

- (1) Pick $n_t \in \{0, 1\}^{l_H}$.
- (2) Set c to be the answer of the random oracle query $(c' \| S \| n_t)$.
- (3) Send $(c, n_t, s_u, s_s, s_{v_B}, s_{\rho_6})$ to the corrupt host.

Game 2 shows that, in the random oracle model, the secure element $\mathcal{SE}$ is able to simulate the proof computed in Step 2 of the spend protocol, i.e., the secure element can compute the proof without knowledge of the secret values (u, s, v, ρ_6) . For brevity, we omit a proof that $|\Pr[\mathbf{Game 2}] - \Pr[\mathbf{Game 1}]| = 0$ since it is based on the same idea as the proof of indistinguishability between **Game 1** and **Game 0**.

— **Game 3:** **Game 3** follows **Game 2**, except that **Game 3** does the following changes in Step 1 of the withdraw phase:

- *Step 1.3:* If (sk_U, pk_U) is not stored, **Game 3** computes a public key pk_U by picking a random element of $\mathbb{G}_q$.
- *Step 1.4:* **Game 3** picks a random element A'' in QR_n and sends A'' to the corrupt host $\mathcal{H}$ through $\mathcal{F}_{\text{SMT}}$.
- *Step 1.7:* **Game 3** sets $V_3^t = A'/A''$, picks $r' \in_R \mathbb{Z}_q^*$, and sends r' and a nonce $n_i \in_R \{0, 1\}^{l_H}$ to the corrupt $\mathcal{H}$.
- *Step 1.9:* **Game 3** stores $A = A' V_2^{r'}$.

Additionally, **Game 3** does the following changes in Step 1 of the spend phase:

- *Step 1.3:* **Game 3** picks a random element B from QR_n and a random element S of $\mathbb{G}_q$ and sends (B, S, J) to $\mathcal{H}$.

We note that, at this point, the secure element does not know any of the secret values (u, s, v, ρ_6) . The proofs computed in Step 2 of the withdraw phase and in Step 3 of the spend phase are now simulated proofs of a false statement. We show that $|\Pr[\mathbf{Game 3}] - \Pr[\mathbf{Game 2}]|$ is negligible.

PROOF. The public key pk_U is equally distributed in **Game 2** and in **Game 3**. The value A'' corresponds to a Damgaard-Fujisaki commitment [Damgård and Fujisaki 2002] to u with randomness s' and v' . Therefore, under the hiding property of this commitment scheme, it is not possible to distinguish A'' from a random value. The value B is also a Damgaard-Fujisaki commitment to u with randomness ρ_6 . The value S is an evaluation of the Dodis-Yampolskiy pseudorandom function [Dodis and Yampolskiy 2005] for key s and input J . Therefore, under the pseudorandomness of

this function, it is indistinguishable from a random value. Therefore, **Game 2** and **Game 3** are indistinguishable with overwhelming probability. $\square$

- **Game 4:** **Game 4** follows **Game 3**, except that, if in Step 2.4 of the withdraw phase, the corrupt $\mathcal{H}$ sends a proof $(c, n_t, s_u, s_{v'}, s_s, s_t)$ such that the proof verifies correctly but any of the values $(s_u, s_{v'}, s_s)$ was not sent before to the corrupt $\mathcal{H}$ for the proof with challenge c and nonce n_t , **Game 4** extracts any of the respective secrets u , v' and s . We show that extraction works in the random oracle model and, therefore, $|\Pr[\mathbf{Game\ 4}] - \Pr[\mathbf{Game\ 3}]|$ is negligible.

PROOF. First, we note that there must be an oracle query $(c' \| n_t)$ whose output was c . Otherwise, in the random oracle model, if no query $(c' \| n_t)$ is recorded, then a new random value would be assigned to that query which would equal c with negligible probability, and thus the proof would not verify correctly.

Given $(c' \| n_t)$, the extractor identifies the random oracle query $(S_x \| T_A \| T_U \| n_i)$ from the corrupt host whose output was c' . This oracle query must also exist, since otherwise when the proof is verified another value different from c' would be assigned with overwhelming probability, and thus the proof would not be correct.

After identifying the oracle query $(S_x \| T_A \| T_U \| n_i)$, the extractor rewinds the corrupt host to the point where this oracle query was made and replies it with a different random value c'_1 . Then the extractor sets a value $c_1 \neq c$ for the oracle query $(c'_1 \| n_t)$. Finally, the corrupt host sends a proof $(c_1, n_t, s_u, s_{v'}, s_s, s_t)$. Let, for instance, s_u be the value that was not sent before the corrupt $\mathcal{H}$. Then, given the proofs $(c, n_t, s_u, s_{v'}, s_s, s_t)$ and $(c_1, n_t, s_{u1}, s_{v'}, s_s, s_t)$, the extractor can extract the secret value u by computing $u = (s_u - s_{u1}) / (c_1 - c)$. The other secret values can be extracted in the same way. $\square$

- **Game 5:** **Game 5** follows **Game 4**, except that, if in Step 2.4 of the withdraw phase, the corrupt $\mathcal{H}$ sends a proof $(d_B, c, n_t, s_u, s_s, s_t, s_{vB}, s_e, s_J, s_{\rho_i}, s_\alpha, s_{w_k})$ for $i = [0, \dots, 7]$ and $k = [1, \dots, 4]$ such that the proof verifies correctly but any of the values s_* was not sent before to the corrupt $\mathcal{H}$ for the proof with challenge c and nonce n_t , **Game 5** extracts any of the secrets $*$. Similarly to **Game 4**, extraction works in the random oracle model and, therefore, $|\Pr[\mathbf{Game\ 5}] - \Pr[\mathbf{Game\ 4}]|$ is negligible. The proof follows the proof in **Game 4**.
- **Game 6:** **Game 6** follows **Game 5**, except that, if in Step 2.4 of the withdraw phase, the corrupt $\mathcal{H}$ sends a proof $(c, n_t, s_u, s_{v'}, s_s, s_t)$ such that the proof verifies correctly but any of the values $(s_u, s_{v'}, s_s)$ was not sent before to the corrupt $\mathcal{H}$ for the proof with challenge c and nonce n_t , **Game 6** aborts. We show that **Game 6** aborts with negligible probability, and thus $|\Pr[\mathbf{Game\ 6}] - \Pr[\mathbf{Game\ 5}]|$ is negligible. This proves that the host cannot withdraw coins without the help of the secure element.

PROOF. We show that, if there is a corrupt host that makes **Game 6** abort with non-negligible probability, we can make a reduction to break the discrete logarithm problem or the hiding property of the Damgaard-Fujisaki commitment scheme.

In the first case, the challenger sends a pair (g, pk_U) , which the reduction uses to set the g value in the bank's public key and the user's public key. If the corrupt host sends a proof $(c, n_t, s_u, s_{v'}, s_s, s_t)$ where s_u was not sent before to the corrupt host, then the reduction runs the extractor to obtain u and sends u to the challenger as the discrete logarithm of pk_U with respect to g .

In the second case, the challenger sends commitment parameters (V_1, V_2, Q) , which the reduction uses to set those values in the bank's public key, and the commitment value A'' , which the reduction uses to set A'' in Step 1.4 of the withdraw protocol. If the corrupt host sends a proof $(c, n_t, s_u, s_{v'}, s_s, s_t)$ where s_u , $s_{v'}$ or s_s was not sent

before to the corrupt host, the reduction uses the extractor to get u , v' , or s , which violates the hiding property of the commitment scheme. $\square$

- **Game 7:** **Game 7** follows **Game 6**, except that, if in Step 2.4 of the withdraw phase, the corrupt $\mathcal{H}$ sends a proof $(d_B, c, n_t, s_u, s_s, s_t, s_{v_B}, s_e, s_J, s_{\rho_i}, s_\alpha, s_{w_k})$ for $i = [0, \dots, 7]$ and $k = [1, \dots, 4]$ such that the proof verifies correctly but any of the values s_* was not sent before to the corrupt $\mathcal{H}$ for the proof with challenge c and nonce n_t , **Game 7** aborts. The proof that $|\Pr[\mathbf{Game 7}] - \Pr[\mathbf{Game 6}]|$ is negligible follows the proof in **Game 6**. Basically, the reduction employs the extractor to get the secret values and then break the hiding property of the commitment scheme, the pseudorandomness of the pseudorandom function or the unforgeability of the banks signature scheme. This proves that the host cannot spend coins without the help of the secure element.

Game 7 produces the same distribution as our simulator $\mathcal{S}$, which we describe now:

- On input $(\text{setupend}, \text{sid})$ from $\mathcal{F}_{\text{ASEC}}$, $\mathcal{S}$ runs a copy of an honest bank on input $(\text{setup}, \text{sid})$.
- On input $(\text{withdraw}, \text{sid})$ from $\mathcal{F}_{\text{ASEC}}$, $\mathcal{S}$ sends $(\text{withdraw}, \text{sid})$ to $\mathcal{F}_{\text{ASEC}}$. If $\mathcal{F}_{\text{ASEC}}$ does not send a subsequent message $(\text{withdraw}, \text{sid}, \mathcal{SE}_i)$, $\mathcal{F}_{\text{ASEC}}$ runs the withdraw phase between a copy of an honest bank and a copy of an honest user (secure element and host) to simulate the withdraw phase towards the adversary. If $\mathcal{F}_{\text{ASEC}}$ sends $(\text{withdraw}, \text{sid}, \mathcal{SE}_i)$, then $\mathcal{S}$ interacts with the adversary by simulating the secure element and the bank sides of the withdraw phase. This simulation employs the changes described until **Game 7**. If the withdraw phase was completed successfully, $\mathcal{S}$ sends $(\text{withdraw}, \text{sid}, 1)$ to $\mathcal{F}_{\text{ASEC}}$, else $\mathcal{S}$ sends $(\text{withdraw}, \text{sid}, 0)$ to $\mathcal{F}_{\text{ASEC}}$.
- On input $(\text{spend}, \text{sid})$ from $\mathcal{F}_{\text{ASEC}}$, $\mathcal{S}$ sends $(\text{spend}, \text{sid})$ to $\mathcal{F}_{\text{ASEC}}$. If $\mathcal{F}_{\text{ASEC}}$ does not send a subsequent message $(\text{spend}, \text{sid}, \mathcal{SE}_i)$, $\mathcal{S}$ runs a copy of an honest user (secure element and host) and an honest merchant to simulate a spend phase towards the adversary. If $\mathcal{F}_{\text{ASEC}}$ sends $(\text{spend}, \text{sid}, \mathcal{SE}_i)$, $\mathcal{S}$ interacts with the adversary by simulating the secure element and the merchant sides of the spend phase. This simulation employs the changes described until **Game 7**. If the spend phase was completed successfully, $\mathcal{S}$ sets $b = 1$ else $b = 0$. If the adversary reveals the identity of the secure element in a message to the merchant, then $\mathcal{S}$ sets $c = 0$ else $c = 1$. $\mathcal{S}$ sends $(\text{spend}, \text{sid}, b, c)$ to $\mathcal{F}_{\text{ASEC}}$.
- On input $(\text{deposit}, \text{sid})$ from $\mathcal{F}_{\text{ASEC}}$, $\mathcal{S}$ runs a copy of an honest merchant and an honest bank to simulate a deposit phase towards the adversary. If the deposit phase is completed successfully, $\mathcal{S}$ sends $(\text{deposit}, \text{sid})$ to $\mathcal{F}_{\text{ASEC}}$.

5. PROOF-OF-CONCEPT IMPLEMENTATION

In this section we provide an illustrative implementation of our proposed anonymous split E-cash scheme. The goal of this section is not to develop a $\mathcal{SE}$ that satisfies all conditions of our scheme, but rather to be able to evaluate the complexity of the protocol on the user's side. To this end, we focus on the computation and communication demands of the $\mathcal{SE}$. We assume other characteristics relevant to the security of the system, e.g. enabling an interface between $\mathcal{SE}$ and the human user, would be presented in a real-world deployment.

We begin by enumerating the hardware components used in the implementation, providing a motivation of our choices. We then identify and describe the bottlenecks encountered on the user side and propose solutions to overcome (or minimize) them. In the last parts of the section we present and discuss the performance results.

A first critical choice for the implementation consists in defining a set of secure parameter lengths for the cryptographic primitives. Figure 2 depicts our selection.

n :	1024 bits	l_y :	1022 bits	l :	10 bits
p :	512 bits	l_H :	120 bits	l_e :	513 bits
l_x :	511 bits	l_ϕ :	80 bits		

Fig. 2. Parameter lengths of our implementation.

5.1. Platform Choices

Motivated by the availability of open and powerful development tools, we have decided to implement the host on an Android-based smartphone¹. We have selected the rather modest Motorola Milestone, which comes equipped with a 600 MHz ARM Cortex-A8 processor and runs an Android v2.2 operating system. Our host implementation is written in Java language and therefore can be easily ported to other devices. We have employed three classes directly available in the Java API: the *BigInteger* class for large integer arithmetic, the *MessageDigest* class for hash functions, and the *Random* class for random number generation.

Enabling full hardware-based security in our system requires a $\mathcal{SE}$ as described in Section 2. The $\mathcal{SE}$ is composed of two main components: a trusted execution platform and a physical interface to the human user. Driven by our goal to evaluate the performance of our E-Cash proposal, we have opted not to develop such a complete token. Instead, we have focused on what we believe is the main interest of the proof-of-concept, i.e. determining the computational demands that are placed in the trusted execution platform.

We have selected the Mobile Security Card (MSC) manufactured by Giesecke & Devrient (G&D), a secure element that can be internally plugged into the phone. The MSC is to the best of our knowledge the only option to *openly* support hardware-based security on Android platforms [SEEK 2014], and has certain properties that make it quite suitable to our envisioned anonymous split E-Cash application. First and most important, it embeds a *certified* smart card chip containing 72 kB of EEPROM and 1.4 kB of RAM secured (among others) against DPA/SPA physical attacks [Kocher et al. 1999]. Second, the MSC supports both Java Card 2.2.1 and Global Platform 2.1.1 APIs, making our implementation portable to any other compliant device. Third, it features multiple cryptographic hardware accelerators for hash functions as well as symmetric and asymmetric encryption algorithms. And fourth, the MSC is physically manufactured as a microSD card easy to interface to multiple mobile platforms such as our Motorola Milestone smartphone.

The bank and merchant are less strict in terms of hardware requirements and we implement them on a generic PC platform for the sake of completeness. Note that neither the functionality of the bank nor the merchant are the bottleneck of our scheme. Consequently, we will not detail these implementations in the rest of this section.

Communication. The MSC offers two standardized smart card communication interfaces. Communication between the Android phone ($\mathcal{H}$) and the MSC Java Card ($\mathcal{SE}$) proceeds in a master-slave fashion as described in the ISO 7816-3 standard [ISO 7816-3 2006]. Android phone (master) and Java Card (slave) communicate by exchanging Application Protocol Data Units (APDUs), the format of which is described in ISO 7816-4 [ISO 7816-4 2005]. The phone is always the first to send an APDU command to the Java Card, which performs some computations upon reception of the command,

¹<http://developer.android.com/sdk/index.html>

and ultimately sends an APDU response back to the phone. The communication interface supports data transfer rates of 9 600 bps.

For usability reasons, the communication between $\mathcal{U}$ and external entities ($\mathcal{B}$ and $\mathcal{M}$) should proceed over a contactless channel. Our initial goal was to employ NFC but at the time of the implementation we were unable to find an Android smartphone phone that supported NFC development while also having a microSD card slot compatible with the MSC. Due to this - and despite being the most attractive choice in terms of compatibility with existing payment infrastructure -, we had to discard this option. For the sake of developing a functional proof-of-concept with involvement of all entities, we decided to employ Bluetooth communication channels.

5.2. Optimization Strategies

In this subsection we go over a few of the design steps followed during our implementation of the user side operations. In particular, we identify implementation issues or bottlenecks and then propose how these can be optimized. Given that the host is a relatively powerful device that eases implementation efforts, in the following we only address the issues regarding to the most resource constrained element, i.e. the Java Card. Note that most of the following strategies are not exclusive of our implementation and can be ported to other systems employing similar hardware targets.

Issue 1. RAM restrictions

PROBLEM DEFINITION. Our MSC Java Card has only 1.4 kB of RAM available to our applet for temporary variable storage. Given that most of the operands in our scheme instantiation are 1024 bits or larger, careful memory planning is key to the implementation feasibility.

PROPOSED SOLUTION. A total of three different but related optimizations have been carried out. First, instead of using one array for each stored variable we have reserved a single big memory block. This allows us to reduce memory overheads due to multiple declarations, i.e. as every object in Java is required to have a header, employing a single large array saves multiple memory bytes. Second, we ensured that the link to old objects is never lost. Note that as Java does not have a garbage collector, previous memory space is used when a link is lost. And third, we can extend the RAM space by using the APDU buffer as RAM memory. Although this allows to slightly increase the RAM memory with 256 bytes, it is important to only store non confidential information in the buffer, as it can be read by other untrusted applications or devices interfacing the MSC.

Issue 2. Generation and Storage of Random Numbers

PROBLEM DEFINITION. As most privacy-preserving protocols, our anonymous split E-cash scheme requires the generation of multiple large random numbers during its operations. For zero-knowledge proofs of knowledge these random numbers need to be used twice - once at the beginning (commitment phase) and once at the end (opening phase) - which means they need to be temporarily stored in memory. As our implementation already deals with memory restrictions, the use of RAM to store these numbers is not a realistic option. Employing EEPROM on the other hand would not only be slower, but we face its limitation on the number of allowed write operations. It is thus not only crucial to find an efficient way to generate randomness, but also figure out a way to store it on the device when it needs to be employed more than once.

PROPOSED SOLUTION. The solution we have devised is based on the observation that the reused random numbers follow the same sequence as when they are generated, i.e. if a zero-knowledge proof of knowledge requires to generate ten random numbers during the commitment phase, the same ten random numbers will be used in the same order during the opening phase.

Based on this we have decided to generate these random numbers by means of a Pseudo Random Number Generator (PRNG). The pseudo random sequence output of a PRNG is completely determined by its input initial values, namely the PRNG's *seed*. Our approach consists in generating a random seed - using the random class in the Java API - at the beginning of the withdraw or spend protocols. The value of the seed is stored in RAM for the rest of the protocol execution. From that moment on, all random numbers in the implementation that will be later reused are generated using the PRNG. Once the implementation reaches the point in which reuse starts, e.g. the opening phase of the zero-knowledge proof of knowledge, we reset the PRNG with the seed stored in RAM. This allows us to regenerate the random numbers in the same sequence as they were first created and use them on-the-fly without storing them in memory.

We have implemented the NIST recommended X9.31 PRNG, as depicted in Figure 3. The inputs v and ctn correspond to the PRNG's seed, while i is an intermediate variable and r holds the output value. At each iteration the following operations are carried out:

$$\begin{aligned} i &= E_K(ctn), \\ r &= E_k(i \oplus v), \\ v &= E_K(r \oplus i), \\ ctn &= ctn + 1, \end{aligned}$$

where E_K denotes a block cipher employing a secret key K . In our implementation we instantiate E_K by using AES-256 [197 2001], whose functionality is directly available via a hardware co-processor.

Issue 3. Large Integer Arithmetic

PROBLEM DEFINITION. Our scheme employs large integers with a length in the order of 1024 bits and executes multiple operations such as (modular) additions, (modular) subtractions, (modular) multiplications, or (multi-base) modular exponentiations. The Java Card 2.2.1 standard does not provide direct support for such operations and consequently we are forced to develop our own routines from scratch.

PROPOSED SOLUTIONS. We have created a customized Java class to perform all required large integer arithmetic. Our implementation of (modular) additions and (modular) subtractions follows traditional 'pencil and paper' algorithms, as found for instance in [Menezes et al. 1996]. These routines are developed purely in Java and oper-

Fig. 3. NIST recommended X9.31 pseudo random number generator.

ate on the input operands word by word starting from the least significant position. As will be further shown in Sect. 5.3 software based routines perform extremely poorly on Java Cards. Consequently one needs to look for alternative mechanisms when implementing even more costly operations such as multiplications or exponentiations.

Techniques to speed up arithmetic on Java Cards have been presented in [Sterckx et al. 2009; Bichsel et al. 2009]. Their core idea is to take advantage of the fast cryptographic co-processors to perform arithmetic operations. Modular exponentiations can be straightforwardly carried out for instance by using the RSA co-processor in the RSA_NOPAD mode which computes $m^e \bmod n$. There is but one limitation that needs to be handled: the co-processor requires that e and n are of the same size, which is not always the case in our protocol in which the exponent has often more bits than the modulus.

A workaround to this issue is to split the exponent e in two smaller components e_1, e_2 such that:

$$e = e_2 2^{|n|-1} + e_1,$$

where $|n|$ indicates the bit-size of n . The modular exponentiation can be then computed with three calls to the RSA co-processor as follows:

$$\begin{aligned} m^e \bmod n &= m^{e_2 2^{|n|-1} + e_1} \bmod n \\ &= m^{e_2 2^{|n|-1}} m^{e_1} \bmod n \\ &= (m^{e_2} \bmod n)^{2^{|n|-1}} m^{e_1} \bmod n. \end{aligned}$$

Finally, modular multiplications are also implemented such that they can make use of the RSA co-processor. In this case we take advantage of the fact that multiplication can be performed via exponentiations as follows:

$$ab \bmod n = \frac{(a+b)^2 - a^2 - b^2}{2} \bmod n.$$

Although the previous equation might seem rather complex, computing multiplications in this way is almost 4 times faster than implementing the traditional ‘pencil and paper’ algorithm purely in Java.

Operation and length of operands (in bytes)	Timing
MODULAR MULTI-EXPONENTIATION: $G_1^{e_1} G_2^{e_2} G_3^{e_3} \bmod n$	
$n = G_1 = G_2 = G_3 = 128, e_1 = e_2 = 94, e_3 = 158$	6231 ms
$n = G_1 = G_2 = G_3 = 128, e_1 = e_2 = 64, e_3 = 128$	3856 ms
MODULAR EXPONENTIATION: $g^e \bmod n$	
$n = g = e = 64$	270 ms
$n = g = 64, e = 94$	1670 ms
$n = g = 128, e = 158$	4116 ms
RANDOM NUMBER GENERATION (NIST PRNG): $r \in \{0, 1\}$	
$r = 10$	120 ms
$r = 64$	252 ms
$r = 128$	480 ms
RANDOM NUMBER GENERATION (JAVA CARD PRNG): $r \in \{0, 1\}$	
$r = 10$	120 ms
$r = 64$	130 ms
$r = 128$	160 ms
OPENING OPERATION IN ZKPK: $r_i - ci$	
$r_i = 158, c = 20, i = 128$	2634 ms
$r_i = 94, c = 20, i = 64$	2143 ms
ADDITION: $a + b$	
$a = b = 64$	181 ms
$a = b = 128$	341 ms
HASH FUNCTION: $H(x)$	
$x = 30$	125 ms
$x = 94$	130 ms
MODULAR MULTIPLICATION: $ab \bmod n$	
$a = b = n = 64$	1382 ms
$a = b = n = 128$	1652 ms

Fig. 4. Response time of basic operations performed in the MSC secure element.

5.3. Performance Evaluation

We depict in Figure 4 the timings obtained by running some arithmetic and cryptographic operations in the MSC. As expected both modular multiplication and exponentiation operations yield the highest timings, i.e. for 1024 bit operands a modular multiplication requires 1.5 seconds and a modular exponentiation consumes slightly more than 4 seconds. The generation of random numbers using the NIST PRNG is slightly larger than using the Java random routine. Perhaps more interesting is the fact that a modular addition with 1024 bit operands requires 341 ms. In other words, the time to perform five modular additions is the same as one modular multiplication, which exemplifies the poor performance of arithmetic software routines on our Java Card based platform.

The global results for an execution of the main protocols on the user side are depicted in Table II. The withdraw protocol requires around 20 seconds to complete, whereas the spend protocol takes around 25 seconds. These results include the communication overhead between $\mathcal{H}$ and $\mathcal{SE}$. As expected, the dominant component of the overall timings in both cases corresponds to the execution in the constrained $\mathcal{SE}$. Performance on $\mathcal{H}$ is on the other hand rather efficient.

For the sake of completeness, we provide in Table III the amount of data that is transferred between the participating entities on both withdraw and spend operations. Recall that in our implementation, the interface between $\mathcal{H}$ and $\mathcal{SE}$ has a transfer

Table II. Computation timings for operations on the user side (including communication).

	$\mathcal{H}$	$\mathcal{SE}$	Total
Withdraw	1.02 s	17.80 s	20 s
Spend	0.15 s	23.34 s	24.90 s

rate of 9 600 bps. Therefore, the communication overhead amounts to 1.18 seconds and 1.39 seconds, respectively. These timings, which are already included in the last column of Table II, show that the communication overhead is roughly 6% of the total execution time.

The communication between $\mathcal{H}$ and external entities, namely $\mathcal{B}$ and $\mathcal{M}$, is envisioned to take place using NFC, already the interface of choice for several (non-anonymous) payment systems. The NFC standard defines three different possible bit rates (106, 212 or 424 kbit/s), which can be used to estimate the communication cost in our implementation. Using the slowest of the three, we obtain an estimated overhead communication of 75 ms and 207 ms for the withdraw and spend phases, respectively.

Table III. Communication between different entities.

	$\mathcal{H} \leftrightarrow \mathcal{SE}$	$\mathcal{H} \leftrightarrow \mathcal{B}$	$\mathcal{H} \leftrightarrow \mathcal{M}$
Withdraw	11 kbits	8 kbits	—
Spend	13 kbits	—	22 kbits

5.4. Discussion

The analysis of the results leads to the following interpretation. While the timings are promising regarding the feasibility and suitability of our approach, a real world deployment would require a considerable speed up of execution times. Taking into account that calculations on the MSC have been thoroughly minimized and optimized in our implementation, the practicality of our system relies on the availability of a more powerful secure element. In other words, the $\mathcal{SE}$ as envisioned in our system cannot be a direct composition of the MSC and the required LCD/push button interfaces.

We consider two possible options to improve performance results in future work. The first one consists in exploring other commercially available devices other than the MSC. The Java Card standard provides only a functional specification of a product. It is up to the manufacturer to decide how to implement the underlying software or hardware such that it complies with the standard. As observed in prior works [Sterckx et al. 2009], the performance of the hardware co-processors between two different Java Cards can easily vary up to 50%. In other words, a direct porting of our implementation to another Java Card could well result in significant improvements. Note also that the new Java Card 2.2.2 en 3.0 standards ensure support for some operations unavailable to our platform, mainly additions, subtractions, and multiplications. The availability of a native arithmetic in the Java Card would bring large improvements on most of our operations. Unreasonably slow computations such as the openings in zero-knowledge proofs of knowledge $s_i = r_i - ci$ could be performed significantly faster.

The second option we consider is to devise a custom token for the $\mathcal{SE}$. The disadvantage of this solution is that it involves costly engineering work and would be more difficult to prototype than a commercial solution. The main advantage, however, is that the token could be fully personalized to optimize our E-Cash application. For instance, dedicated hardware accelerators could be added to the trusted environment in order to improve current implementation bottlenecks. Moreover, the communication times

between $\mathcal{H}$ and $\mathcal{SE}$ would also benefit from this approach. The MSC offers a communication channel with a limited rate of 9 600 bps. The usage of an external interface as we envision, for instance based on a USB connector, would make the communication overhead between $\mathcal{H}$ and $\mathcal{SE}$ negligible in front of computation time.

6. CONCLUSIONS

In this work we have studied the feasibility of the anonymous E-Cash scheme on mobile scenarios without fully tamper-resistant payment devices. Motivated by existing MTM non-anonymous payment architectures, we have considered at design time a payment token composed of an untrusted yet powerful platform connected to a trusted but rather constrained module. Following this approach, we have devised a protocol variant that fulfills all common properties of E-cash while relaxing the amount of trust placed in the payment token. We have evaluated the cost of our proposal on a proof-of-concept implementation that uses only commercially available platforms, and proposed possible directions for future work.

REFERENCES

- Federal Information Processing Standards Publication 197. 2001. Specification for the Advanced Encryption Standard (AES). (2001). <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- Josep Balasch. 2008. *Smart Card Implementation of Anonymous Credentials*. Master's thesis. Katholieke Universiteit Leuven (KU Leuven).
- Lejla Batina, Jaap-Henk Hoepman, Bart Jacobs, Wojciech Mostowski, and Pim Vullers. 2010. Developing Efficient Blinded Attribute Certificates on Smart Cards via Pairings. In *Smart Card Research and Advanced Application - CARDIS 2010 (Lecture Notes in Computer Science)*, Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny (Eds.), Vol. 6035. Springer, 209–222.
- Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. 2009. Anonymous credentials on a standard java card. In *Conference on Computer and Communications Security - CCS 2009*, Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis (Eds.). ACM, 600–610.
- Stefan Brands. 1994. Untraceable Off-line Cash in Wallets with Observers (Extended Abstract). In *Advances in Cryptology - CRYPTO '93 (LNCS)*, Douglas R. Stinson (Ed.), Vol. 773. Springer, 302–318.
- Stefan Brands. 2000. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. Master's thesis. MIT Press.
- Ernest F. Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct anonymous attestation. In *ACM Conference on Computer and Communications Security*. 132–145.
- Jan Camenisch and Els Van Herreweghen. 2002. Design and implementation of the *idemix* anonymous credential system. In *Conference on Computer and Communications Security - CCS 2002*, Vijayalakshmi Atluri (Ed.). ACM, 21–30.
- Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *Advances in Cryptology - EUROCRYPT 2005 (LNCS)*, Ronald Cramer (Ed.), Vol. 3494. Springer, 302–321.
- Jan Camenisch, Anja Lehmann, Gregory Neven, and Alfredo Rial. 2014. Privacy-preserving auditing for attribute-based credentials. In *Computer Security-ESORICS 2014*. Springer, 109–127.
- Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding (Lecture Notes in Computer Science)*, Birgit Pfitzmann (Ed.), Vol. 2045. Springer, 93–118.
- Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. 2007. Endorsed E-Cash. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 101–115.
- Sébastien Canard and Aline Gouget. 2007. Divisible E-Cash Systems Can Be Truly Anonymous. In *Advances in Cryptology - EUROCRYPT 2007 (LNCS)*, Moni Naor (Ed.), Vol. 4515. Springer, 482–497.
- Sébastien Canard and Aline Gouget. 2008. Anonymity in Transferable E-cash. In *ACNS (Lecture Notes in Computer Science)*, Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung (Eds.), Vol. 5037. 207–223.
- Sébastien Canard, Aline Gouget, and Jacques Traoré. 2008. Improvement of Efficiency in (Unconditional) Anonymous Transferable E-Cash. In *Financial Cryptography (Lecture Notes in Computer Science)*, Gene Tsudik (Ed.), Vol. 5143. Springer, 202–214.

- David Chaum. 1983. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, David Chaum, Ronald L. Rivest, and Alan T. Sherman (Eds.). Plenum Press, 199–203.
- Chipknip. last visited: 01/07/2014. An offline smartcard payment system. (last visited: 01/07/2014). <http://www.chipknip.nl>
- Efrén Clemente-Cuervo, Francisco Rodríguez-Henríquez, Daniel Ortiz Arroyo, and Levent Ertaul. 2007. A PDA Implementation of an Off-line e-Cash Protocol. In *Proceedings of the 2007 International Conference on Security and Management - SAM 2007*, Selim Aissi and Hamid R. Arabnia (Eds.). CSREA Press, 452–458.
- Common Criteria. last visited: 01/07/2014. (last visited: 01/07/2014). <http://www.commoncriteriaportal.org>
- Ivan Damgård and Eiichiro Fujisaki. 2002. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *ASIACRYPT*. 125–142.
- David Derler, Klaus Potzmader, Johannes Winter, and Kurt Dietrich. 2011. Anonymous Ticketing for NFC-Enabled Mobile Phones. In *Trusted Systems - INTRUST 2011 (Lecture Notes in Computer Science)*, Liqun Chen, Moti Yung, and Liehuang Zhu (Eds.), Vol. 7222. Springer, 66–83.
- Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *Public Key Cryptography*. 416–431.
- EMVCo. last visited: 01/07/2014. The EMV standards body. (last visited: 01/07/2014). <http://www.emvco.com>
- Geldkarte. last visited: 01/07/2014. An offline smartcard payment system. (last visited: 01/07/2014). <http://www.geldkarte.de>
- Gesine Hinterwälder, Christof Paar, and Wayne P. Burleson. 2013a. Privacy Preserving Payments on Computational RFID Devices with Application in Intelligent Transportation Systems. In *Radio Frequency Identification. Security and Privacy Issues - RFIDSec 2012 (Lecture Notes in Computer Science)*, Jaap-Henk Hoepman and Ingrid Verbauwhede (Eds.), Vol. 7739. Springer, 109–122.
- Gesine Hinterwälder, Christian T. Zenger, Foteini Baldimtsi, Anna Lysyanskaya, Christof Paar, and Wayne P. Burleson. 2013b. Efficient E-Cash in Practice: NFC-Based Payments for Public Transportation Systems. In *Privacy Enhancing Technologies - PETS 2013 (Lecture Notes in Computer Science)*, Emiliano De Cristofaro and Matthew Wright (Eds.), Vol. 7981. Springer, 40–59.
- Benjamin Ransford Kevin Fu Hong Zhang, Jeremy Gummeson. 2011. *Moo: A Batteryless Computational RFID and Sensing Platform*. Technical Report. University of Massachusetts Computer Science.
- ISO 7816-3. last updated: 2006. Smart card standard: Cards with contacts Electrical interface and transmission protocols. (last updated: 2006).
- ISO 7816-4. last updated: 2005. Smart card standard: Organization, security and commands for interchange. (last updated: 2005).
- Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99)*. Springer-Verlag, 388–397.
- Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. 1996. *Handbook of Applied Cryptography* (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.
- Wojciech Mostowski and Pim Vullers. 2011. Efficient U-Prove Implementation for Anonymous Credentials on Smart Cards. In *Security and Privacy in Communication Networks - SecureComm 2011 (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering)*, Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis (Eds.), Vol. 96. Springer, 243–260.
- Tatsuaki Okamoto. 1995. An Efficient Divisible Electronic Cash Scheme. In *CRYPTO (Lecture Notes in Computer Science)*, Don Coppersmith (Ed.), Vol. 963. Springer, 438–451.
- PROTON. last visited: 01/07/2014. An offline smartcard payment system. (last visited: 01/07/2014). http://mypaymentsolution.be/index/en_US/5118014/5126207/Proton.htm
- SEEK. last visited: 01/07/2014. Secure Element Evaluation Kit for the Android platform. (last visited: 01/07/2014). <http://code.google.com/p/seek-for-android>
- Michal Sterckx, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. 2009. Efficient Implementation of Anonymous Credentials on Java Card Smart Cards. In *1st IEEE International Workshop on Information Forensics and Security (WIFS 2009)*. IEEE, London, UK, 106–110.
- Eric R. Verheul. 2001. Self-Blindable Credential Certificates from the Weil Pairing. In *Advances in Cryptology - ASIACRYPT 2001 (Lecture Notes in Computer Science)*, Colin Boyd (Ed.), Vol. 2248. Springer, 533–551.
- Visa payWave. last visited: 01/07/2014. Mobile payments. (last visited: 01/07/2014). <https://developer.visa.com/paywavemobile>