PSP: Private and Secure Payment with RFID

Erik-Oliver Blass[1] Anil Kurmus[1] Refik Molva[1] Thorsten Strufe[2]
[1]EURECOM, Sophia Antipolis, France
[2]TU Darmstadt, Darmstadt, Germany

ABSTRACT

RFID can be used for a variety of applications, e.g., to conveniently pay for public transportation. However, achieving security and privacy of payment is challenging due to the extreme resource restrictions of RFID tags. In this paper, we propose PSP – a secure, RFID-based protocol for privacy-preserving payment. Similar to traditional electronic cash, the user of a tag can pay access to a metro using his tag and so called *coins* of a virtual currency. With PSP, tags do not need to store valid coins, but generate them on the fly. Using Bloom filters, readers can verify the validity of generated coins offline. PSP guarantees privacy such that neither the metro nor an adversary can reveal the identity of a user or link subsequent payments. PSP is secure against *invention* and *overspending* of coins, and can reveal the identity of users trying to *doublespend* coins. Still, PSP is lightweight: it requires only a hash function and few bytes of non-volatile memory on the tag.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*Cybercash, Digital Cash, Payment Schemes, Security*

General Terms

Algorithms, Design, Security

Keywords

RFID, Payment, Privacy, Security, Ecash

1. INTRODUCTION

Radio Frequency Identification (RFID) systems that were originally targeting simple object identification are used more and more for sophisticated applications such as access control and payment. The idea of having several small and cheap devices wirelessly communicate with a reader is appealing for various scenarios. The Oyster Card [29] is a prominent example of a large scale deployment of contactless smartcards, i.e., powerful RFID tags, enabling convenient payment for public transportation services. In such a scenario, people entering a metro just quickly hold their tags close to a reader device in front of a gate to issue the payment for the transport fee. Similar payment and access control schemes are in widespread use for other kinds of public transportation such as buses, tramways, or trains.

Such payment scenarios however raise challenging security and privacy issues. First, an intruder or a malicious user, issuing bogus payments or impersonating legitimate users, should be prevented from having access to public transportation. The second concern is privacy that is often emphasized both as a user requirement and a regulatory matter. Privacy requires that neither an external adversary nor the public transportation system is able to identify or trace users by exploiting the payment system. Furthermore, readers in a metro station or in a bus are often embedded devices that are not permanently connected to a backend system such as a server. So, readers must be able to verify *offline* and within a very short period of time, whether a payment is valid or not.

This payment scenario typically calls for an offline, anonymous electronic payment solution such as the ones targeted by myriads of ecash, payment and micro-payment schemes extensively covered in the literature – e.g., see first seminal papers by Brands [3], Chaum [5], Chaum et al. [6], for micropayment Micali and Rivest [20], Rivest [24], or see van Tilborg [33] for an overview. Yet, due to the inherent requirement for blind signatures, existing solutions for anonymous, offline payment are based on complex asymmetric cryptography. Customizing these solutions for RFID systems therefore implies prohibitive complexity that exceeds the capacity of existing RFID devices. Due to the strong cost and size constraints, RFID tags neither feature complex asymmetric cryptographic primitives nor large amounts of memory, cf., Avoine et al. [2], Choi et al. [7], Dimitrou [10], Pietro and Molva [22], Tsudik [32], Weis et al. [37]. Typically, hash functions are the only cryptographic primitive feasible on the hardware of RFID tags. Alternative approaches based on time-memory trade-offs, i.e., storing a number of precomputed values on the tag like MilliCent [12] or MicroMint [25], are not suitable either, since tags also have very little non-volatile memory. In conclusion, traditional ecash solutions cannot be applied to RFID.

Related work: The Oyster Card has several drawbacks: its security has been broken [11], it uses a physically large, contactless smartcard being more expensive than today's

WPES'09, November 9, 2009, Chicago, Illinois, USA.

tiny RFID EPC Gen 2 tags [1], and bank and metro are combined, so there cannot be any privacy for users. Similar to Oyster Card, large and expensive contactless smartcards are also used in, e.g., Visa's payWave [36] program. Other RFID payment solutions depend on using an additional mobile phone to confirm payment or read out a barcode on the phone's display [16, 17, 35, 38] – which is inconvenient.

To the best of our knowledge, there is no *privacy-preserving*, *secure*, and *offline* payment solution *distinguishing* between bank and payees solely using RFID tags for payment.

This paper presents PSP, a protocol for secure, private, and offline payment suited for RFID tags. Following notions of ecash, the tag serves as a rechargeable *electronic wallet*, it is responsible for repeated "storage" of pre-paid coins of a virtual currency, and manages all communication with the reader. The **main idea** of PSP is that the tag does not physically store coins, but it receives some information to generate a limited number of valid coins on the fly. Using Bloom filters, readers can verify the validity of coins received from tags.

PSP meets the security and privacy requirements raised by RFID based payment as follows:

- An adversary cannot arbitrarily *invent* new coins, i.e., introduce coins into the system for which he did not legitimately pay for through the bank.
- *Overspending* coins is impossible: an adversary cannot replay coins from his own payments or stolen coins from other people's payments he eavesdropped.
- Readers are offline and only synchronize, e.g., once a day, with the bank. Yet, an adversary still cannot *doublespend* coins: he cannot pay with the same coin twice at *different* readers. If he does, his identity will be revealed at the bank. Revealing the identity of adversaries trying to doublespend money has $O(1)$ complexity for the bank, but is impossible for readers.
- Users of RFID tags remain *anonymous* and are *untraceable*: the true identity of a user is hidden to the public transport system and the adversary. Also, neither the public transportation system nor an adversary can trace users/tags on subsequent payments, i.e., link different payments to the same tag.
- PSP is lightweight: besides being able to execute a hash function, tags feature only a few bytes of non-volatile memory.
- PSP can cope with resource-limited readers. Readers are often embedded devices, so their storage and computational resources are, although orders of magnitude higher compared to tags, still restricted. With respect to the number of tags and coins, complexity for verifying a coin is in $O(1)$.

Note that in contrast to ecash with blind signatures, PSP will not prevent a malicious bank from tracing users: by eavesdropping wireless communication between tags and readers, the bank can identify which user issued a payment at which reader. Hence, PSP does *not* protect privacy against the bank. However, revealing privacy to the bank can only be prevented by using computationally expensive blind signature techniques.

The sequel of the paper is structured as follows: first, an overview of the main idea behind PSP is given in Section 2; Section 3 presents the adversary model assumed in this paper; Section 4 introduces the general setup of the system, the bank, tags, readers, notion of time etc; Section 5 then presents PSP in detail: the process of preparing the whole system by the bank, the act of (re-)charging a tag, and the actual payment protocol; Section 6 discusses, why PSP is secure, i.e., analyzes its security and privacy properties.

2. OVERVIEW

This paper uses the term "metro" for all kinds of public transportation systems: buses, trams, (urban) railways etc. As with the Oyster Card, access to the public transportation system is only granted, if Alice, the user of a tag, can pay her trip using her RFID tag and a reader. Otherwise, a gate or barrier to access public transport will not open, or the bus driver will not let her use public transportation. Note that PSP is quite general and its usage is not restricted to payment for public transport, but can be extended to all kinds of ecash scenarios. However, details on how to adapt some aspects of PSP for different payment scenarios are out of scope of this paper and will be addressed in future.

Before starting with the more precise description of the system setup and all protocol details in sections 4 and 5, the following paragraphs give an *informal overview* about the basic concepts of PSP.

Payment in PSP. The idea of payment with PSP is that each metro user Alice carries an RFID tag T_{Alice}. Before Alice passes a barrier, she wipes T_{Alice} close to a reader attached to the barrier. Reader and T_{Alice} exchange data, using the PSP protocol. In general, the data exchanged is payment information – in PSP, this is some kind of "digital coins" of money. PSP basically achieves reader authentication and the actual payment. First, T_{Alice} sends a coin as a commitment to the reader. Then, T_{Alice} performs a challenge-response protocol for reader authentication. As tags and readers do not share keys in PSP, T_{Alice} knows a set of precomputed challenges and matching response pairs provided by the bank. Once the reader is successfully authenticated by T_{Alice}, T_{Alice} completes the payment by revealing the preimage of the committed coin to the reader. Before opening the barrier, the reader verifies whether the coin received from Alice is valid and whether it has been spent before. To protect against an adversary trying to eavesdrop and steal a coin, T_{Alice} sends not only a valid coin during payment, but an additional "fake" coin. Only readers, but not an adversary, can distinguish valid from fake coins.

Charging: Storing Money on Tags. Before any payment, Alice has to "charge" her tag with digital coins. Therefore, Alice goes to a bank and gives "real" money to the bank. In return, the bank sends back some information to the tag which is stored on the tag's non-volatile memory. Using this information, the tag can later generate digital coins used for payments. Eventually, a tag is "exhausted", i.e., all the coins Alice has paid for are spent. Alice can go to the bank and (re-)charge her tag or discard the old tag and buy a new one. During charging, the bank also gives the tag a set of so called verification bits that will be used during above mentioned challenge-response authentication.

Reader Preparation. The bank also prepares readers. During an initial phase, before any charging of tags, before any payment, and only *once*, the bank prepares a Bloom fil-

ter for each reader. This Bloom filter represents all possible valid coins in the system. Consequently, later during payment, a reader can therewith verify, whether a coin received from a tag is valid or not.

Maintenance. Readers are off-line most of the time, i.e., during normal, daily operation. But periodically, e.g., once a day during the night, readers conduct maintenance: readers connect to the bank, send in the coins collected over the day, and receive from the bank information to update their Bloom filters.

Time. To cope with new tags entering the payment system over time, old ones being discarded, and new coins, time is divided into "epochs" with PSP. The typical duration of an epoch is in the order of several days or *one month*. At the beginning of a new epoch, the bank carries out the abovementioned initial preparation of readers once. Also, the bank and all readers discard all information predating the last epoch, e.g., the Bloom filters. As a result, Alice might not be able to pay with coins which are older than 2 epochs anymore – coins eventually expire.

PSP's Security and Privacy Properties. The information received from the bank allows Alice's tag (or an adversary) to only generate as many coins as she has paid for. Any other generated coin is, with high probability, rejected by readers. If Alice tries to spend one valid coin, i.e., a coin she really paid for, twice at the same reader, this is immediately detected and rejected by this reader. If Alice tries to spend a valid coin on the same day at two different readers, this is detected by the bank during maintenance, and the bank can identify Alice as being the origin of such malicious behavior. An adversary cannot impersonate a reader and thereby steal coins from tags, as he would have to authenticate himself to a tag. Finally, no one can establish any correlation among coins even when they originate from the same tag. Therefore, neither the metro nor an adversary can track Alice.

Bloom Filters. The following is a quick introduction to Bloom filters limited to what is necessary for understanding this paper. For more information, refer to Broder and Mitzenmacher [4]. We use Bloom filters, as they are a *space efficient* data structure representing a set of elements. The following operations are supported. 1.) addBF(x, y) adds a new element y to Bloom filter x. 2.) isElement(x, y) outputs $true$, if y has been added to Bloom filter x, $false$ otherwise. Here, isElement(x, y) is prone to *false positives*: it might output $true$ with probability P, even if y has not been added to x before. False negatives are impossible. For convenience, isNotElement(x, y) outputs the opposite of isElement(x, y). Finally, 3.) GenerateEmptyBloomFilter(s) returns a new empty Bloom filter of a specified size s.

3. ADVERSARY MODEL

Following definitions from Cramer and Damgård [8], we assume an *active* adversary: The adversary can not only listen to wireless communication between tags and readers, but also initiate communication with arbitrary tags and readers. The adversary might act like a man-in-the-middle and *rush*, i.e., he can intercept messages, modify or even selectively block them before forwarding them to their destination[1]. The adversary also sees the "outcome" of a payment, i.e., if the barrier opens or not after tag and reader have exchanged some messages. The adversary is computationally, timewise, and memory-wise bounded to typical "security margins". For example, he cannot invert a hash function.

An adversary might also compromise tags. He can readout all the memory of the tag and tamper with the data and logic stored on a tag. As as result, the tag's behavior might not comply with the protocol anymore. If he compromises a tag, there is, of course, no way to prevent him from spending all coins of the tag the original owner paid for. In conclusion, the above adversary is equivalent to the one of Juels and Weis [18] or the STRONG adversary of Vaudenay [34].

Typically users care about being traced by curious payees, in this case the metro system. Consequently, PSP should not only be private against an "outsider" adversary as described above, but also against the payees. In the sequel, we assume that, in addition to the outsider adversary, a group of *honest-but-curious* (HBC) readers might collaborate. Their goal is to reveal users' privacy and to trace users. Note that with Oyster Card, there is traceability of users [27].

In contrast to ecash, we do not consider the bank as an adversary and interested in violating users' privacy. Instead in this paper, the bank is trusted by tags (and the metro). We conjecture that users intuitively trust their bank much more than the metro or any payee. Not trusting the bank would require properties equivalent to blind signatures, cf., Chaum [5], which cannot be afforded on tags due to their computational overhead. Furthermore, we assume all communication between readers and bank as trusted using traditional security mechanisms. Also, the communication between Alice and the bank, e.g., for charging her tag, takes place through a secure channel.

4. SYSTEM ASSUMPTIONS AND SETUP

Due to their limited capacity, the most complex operation tags can afford is a cryptographic hash function [2, 10, 18, 19, 22, 26, 32, 37]. For convenience, we assume communication between tags and readers to be error-free. As wireless communication is typically prone to static noise, we assume appropriate mechanisms like ARQ techniques to be implemented on lower communication layers. As mentioned in Section 3, the adversary might, however, selectively drop messages. In this case, underlying ARQ mechanisms will give a timeout to PSP.

In sequel of this section, we introduce the components of PSP.

4.1 Money

Tags, the bank, and readers use a virtual currency called *coins*. Although we use, by a stretch of language, the notion of *storing* coins on a tag, tags do not directly "store" coins in their non-volatile memory: instead, the bank will provide tags with information that allows them to *create valid coins* in real-time.

Alice can *charge* a tag with money, i.e., trade in real money (\$, £, €, ...) into coins. Alice can buy and (re-)charge her tag only at a bank or at special cash machines. For convenience, the term "bank" used throughout this paper encompasses all places where Alice can charge her tag, and

[1]Note that, without special physical assumptions such as time or distance bounding, Mafia Fraud [9] is possible in ecash systems. Preventing Mafia Fraud is out of scope of this paper, but PSP can be extended, e.g., using RFID time bounding protocols, cf., Hanke [14], Hanke and Kuhn [15].

these places are synchronized and connected online to the same kind of backend banking system.

For convenience, we make the following simplifications regarding handling coins, charging, and payment in PSP:

1.) The exchange rate is \$1 for one coin. Transportation will always cost integer multiples of coins, there is no notion of fractions of a coin.

2.) A tag can only be (re-)charged, if all its coins have been spent. Also, a tag can only be charged with $\gamma_{\max}$ coins at a time. Every time Alice wants to charge her tag with $\gamma_{\max}$ coins, the bank provides Alice's tag with a so called "ID". This ID will be used to generate coins during payments.

As PSP with the above simplifications for better understanding is rather limited, we will extend it and add more flexibility in Section 5.5.

4.2 Per Epoch System Parameters

In PSP, time is divided into consecutive *epochs* $\epsilon_1, \ldots, \epsilon_{256}$. For example, one epoch is one month. Using statistics available over the last recent years with traditional payments, bank and metro know the following averages or expectation values. On average *and per epoch* ϵ_i:

There are τ different tags $T_1, \ldots, T_\tau$ in the system.

A tag will spend γ_{avg} coins on average in public transport during one epoch.

Unused coins will expire after 2 epochs, e.g., after two months[2].

In total, there are $\eta = \tau \cdot \gamma_{\text{avg}}$ coins in the system. Consequently, the number of IDs to generate η coins on average is $\#\text{ID} = \frac{\eta}{\gamma_{\max}}$.

Generally, the metro should choose all above system parameters using appropriate safety margins.

4.3 Readers

The system consists of ρ readers. Each reader has a unique *Reader ID*, $\text{RID} = 1, \ldots, \rho$.

Readers are not assumed to be permanently online connected to the bank and also cannot exchange data with each other. Instead, readers are offline most of the time and connect to the bank using a certain schedule, e.g., once a day during the night. In conclusion, readers are not synchronized most of the time.

Finally for cost and reliability reasons, readers are also assumed to be resource restricted embedded devices. We assume their available storage to be similar to what is available on today's embedded memory technologies, e.g., less than 1 GByte.

4.4 Security Parameters

PSP assumes a cryptographic hash function h that can be executed on a tag. Output size of h is 128 bit; we use, e.g., a SHA-1 implementation for RFID tags and truncate the output to 128 bit, cf., Choi et al. [7] or more lightweight hash functions such as SQUASH for RFID, cf., Shamir [26]. Truncation to 128 bit helps to reduce storage amount on the tag, as we will see in Section 5.7.

PSP uses (optimized) Bloom filters [4] to store information about all η valid coins during one epoch. In the following, we present the major security properties and parameters required for PSP.

1.) Parameter κ defines the number of different hash functions used for Bloom filters. As h is a cryptographic hash function, we can, instead of using κ *different* hash functions $h_1, \ldots, h_\kappa$, simply define $h_i(x) = h(i, x)$, where "," denotes an unambiguous pairing of inputs.

2.) Parameter μ defines the storage size of each Bloom filter, i.e., its number of bits. For given κ and η, it is possible to define the size μ of the Bloom filter with $\frac{\mu}{\eta} = \frac{\kappa}{\ln 2}$ such that the probability of any bit in the Bloom filter being set to 1 is $p = \frac{1}{2}$. As a result, the probability of finding a single false positive in the Bloom filter is $P = \frac{1}{2^\kappa}$. A false positive in our context is the case where an adversary computes or guesses by chance one single coin which is accidently accepted by the Bloom filter – as described in the following sections in higher detail.

3.) Parameter ω is the number of *verification bits* used for reader-to-tag authentication. An adversary is able to impersonate a reader with $2^{-\omega}$ probability to receive one single valid coin from a valid tag. *Note:* If an adversary fails to compute the correct verification bits for one coin from a valid tag, the tag will send him "fake" payments – as described in the following sections in higher detail.

[2]The bank can later reimburse expired coins, cf., Section 6.2.

5. PROTOCOL DESCRIPTION

5.1 Preparation of new epoch ϵ_{i+1}

For the first epoch, as well as periodically at the end of each epoch (e.g., once a month), the bank prepares the system for the subsequent epoch ϵ_{i+1} as follows.

1.) The bank creates a new 128 bit symmetric *epoch key* $K_{\epsilon_{i+1}}$ and sends it to all readers. Also, the bank creates two new, empty hash-tables called $\Delta_{\epsilon_{i+1}}$, $\Sigma_{\epsilon_{i+1}}$. The bank discards all stored information of epoch ϵ_{i-1}, that is, hash-tables $\Delta_{\epsilon_{i-1}}, \Sigma_{\epsilon_{i-1}}$.

2.) The so called IDs are prepared. In epoch ϵ_{i+1}, $\forall k : 1 \leq k \leq \#\text{ID}$, $\text{ID}_k^{\epsilon_{i+1}}$ can be computed as: $\text{ID}_k^{\epsilon_{i+1}} = h(K_B, k, \epsilon_{i+1})$, where K_B is a 128 bit key only known to the bank.

3.) The bank generates Bloom filters for all readers as shown in Algorithm 1. Prior to this, each reader discards the information stored about epoch ϵ_{i-1}, i.e., $\text{BF}_{\text{RID}}^{\epsilon_{i-1}}$ and $\text{spentBF}_{\text{RID}}^{\epsilon_{i-1}}$.

```
foreach RID do
    BF^{ε_{i+1}}_RID := GenerateEmptyBloomFilter(μ);
    for k := 1 to #ID do
        for l := 1 to γ_max do
            coin := h(h(RID, ID_k^{ε_{i+1}}, l));
            addBF(BF^{ε_{i+1}}_RID, coin);
        end
    end
    spentBF^{ε_{i+1}}_RID := GenerateEmptyBloomFilter(μ);
    Bank ⟶ Reader RID : {BF^{ε_{i+1}}_RID, spentBF^{ε_{i+1}}_RID};
end
```

Algorithm 1: Preparing Bloom filters

In summary, a valid *coin* in PSP can be created simply by knowing a valid RID, a valid ID, and a simple counter. The $\text{BF}_{\text{RID}}^{\epsilon_{i+1}}$ Bloom filter is filled with all possible *coins* that will exist during epoch ϵ_{i+1}. Based on $\text{BF}_{\text{RID}}^{\epsilon_{i+1}}$, reader RID will be in a position to verify whether coins presented to it are valid. The $\text{spentBF}_{\text{RID}}^{\epsilon_{i+1}}$ Bloom filter, while empty at the beginning, will later store all coins spent during epoch ϵ_{i+1}. It will enable reader RID to check whether a coin has already been spent, cf., Section 5.3. Note that with one valid ID, up to $\gamma_{\max}$ valid coins for reader RID can be created.

5.2 (Re-)Charging a Tag

In epoch ϵ, Alice wants to buy a new tag or recharge an old one. Alice's tag is T_{Alice}. The idea behind charging T_{Alice} is that the bank gives IDs to T_{Alice} which will enable T_{Alice} to later generate valid coins. To that effect, the bank maintains counters ξ^ϵ to store the information about which IDs have already been sold to tags. So, $1 \leq \xi^\epsilon \leq \#\text{ID}$.

1.) Alice gives the equivalent amount of money to buy $\gamma_{\max}$ coins.

2.) The bank computes the yet unused $\text{ID}^\epsilon_{\xi^\epsilon} = h(K_B, \xi^\epsilon, \epsilon)$ and sends the tuple $(\text{ID}^\epsilon_{\xi^\epsilon}, \epsilon)$ to T_{Alice} which stores it in non-volatile memory.

3.) For each coin sold to Alice, the bank computes so called *verification bits* ν_j, $|\nu_j| = \omega$ bit, and sends them to Alice using Algorithm 2. T_{Alice} stores ν_j. K_ϵ is the current epoch key (as described in Section 5.1).

for $j := 1$ **to** $\gamma_{\max}$ **do**
  $chall := h(\text{ID}^\epsilon_{\xi^\epsilon}, j)$; `// Challenge`
  $\nu_j := \lfloor h(K_\epsilon, chall) \rfloor_\omega$; `// Response`
  Bank $\longrightarrow T_{\text{Alice}}$: $\{\nu_j\}$;
end

Algorithm 2: Computation of verification bits

Here, $\lfloor h(x) \rfloor_\omega$ is a *truncated* hash value, the first ω bits of output of $h(x)$. As only the readers and the bank know K_ϵ, Alice can later use *chall* as a challenge to any reader and verify a reader's response using the ν_j – therewith providing reader authentication. Together with tuple $(\text{ID}^\epsilon_{\xi^\epsilon}, \epsilon)$, T_{Alice} stores a local counter c in its non-volatile memory. c keeps track of how many coins have already been spent by T_{Alice}. Also, a maximum value max is stored, representing the maximum number of coins that can be spent with $\text{ID}^\epsilon_{\xi^\epsilon}$, $\max = \gamma_{\max}$.

4.) The bank adds information to its hash-tables Δ_ϵ and Σ_ϵ to protect against double spending as described in Algorithm 3.

for RID $:= 1$ **to** ρ **do**
  for $j := 1$ **to** $\gamma_{\max}$ **do**
    $coin := h(h(\text{RID}, \text{ID}^\epsilon_{\xi^\epsilon}, j))$;
    $prechall := \{\xi^\epsilon, \epsilon, j\}$;
    addHash$(\Delta_\epsilon, coin, prechall)$;
    $spent := 0$;
    $Identity := \{Name, spent\}$;
    addHash$(\Sigma_\epsilon, prechall, Identity)$;
  end
end
$\xi^\epsilon := \xi^\epsilon + 1$;

Algorithm 3: Bank prepares against double spending

addHash(x, y, z) stores value z in hash-table x at key y.

In $Name$, the bank stores an (unique) identifier of Alice, e.g., her name or her account number. With $Name$, the bank should be able to identify Alice at a later point in time. *Spent* is one bit storing the information, whether the user identified by $Name$ has already spent a coin based on $(\text{ID}^\epsilon_{\xi^\epsilon}, j)$ at any reader.

5.3 Payment

For the sake of simplicity, the price for using metro lines is fixed at 1 coin. This section describes the payment protocol (Figure 1) through which T_{Alice} pays 1 coin to reader RID. Let the current epoch be ϵ, the current ID used by T_{Alice} to generate coins be ID^ϵ_ξ, the verification bits be ν_j.

Payment Overview: Reader RID initiates the protocol by sending RID to tag T_{Alice}. T_{Alice} responds by sending two *coins*, a valid coin and a fake one. Sending these coins serves both the purpose of confusing a potential adversary and achieving T_{Alice}'s commitment for the payment. T_{Alice} also sends a challenge *chall* and the epoch ϵ of the valid coin. Upon receipt of message 2 and successful verification of the valid coin, the reader replies by sending verification bits v. Finally, if v matches *chall*, the reader is authenticated and T_{Alice} reveals the preimage of the committed valid coin, otherwise the authentication of the reader has failed, and T_{Alice} replies with the preimage of the committed fake coin.

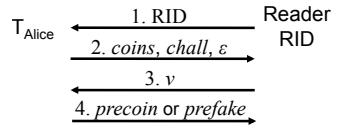

Figure 1: Message flow in PSP

Payment Details: Algorithm 4 presents a more detailed sketch of the payment protocol. In ①, reader RID periodically broadcasts its ID. Using $\text{ID}^\epsilon_{\xi^\epsilon}$, c, and RID, T_{Alice} computes a challenge *chall*, a valid *coin* out of *precoin*, and a (pseudo-)random invalid $fake$ coin out of $prefake$. In ②, T_{Alice} sends *coin*, $fake$, *chall*, and ϵ to the reader. Here, the order of sending *coin* and $fake$ swaps depending on a random bit b: flip(b, y, z) is $\{y, z\}$ **iff** $b = 0$, and $\{z, y\}$ otherwise. So, T_{Alice} randomly chooses the order of sending valid *coin* and $fake$ coin to mislead the adversary. The adversary cannot distinguish between the valid and the fake coin. Sending *coin* and $fake$ serves as a commitment, where the preimages will be revealed later. The reader verifies, whether one of the two received coins, $coin_1$, $coin_2$, is valid, i.e., is in its Bloom filter $\text{BF}^\epsilon_{\text{RID}}$. Thereby, the reader identifies which coin is the valid coin (called *vcoin* thereafter), see ③. Also, the reader verifies whether this coin has not been spent on any reader before the last scheduled maintenance (check $\text{spentBF}^\epsilon_{\text{RID}}$), and whether the coin has not been spent on this reader since the last maintenance. For the latter, the reader maintains a simple list, $\text{spentList}^\epsilon_{\text{RID}}$. If *vcoin* passes the above tests, see ④, *vcoin* is added to $\text{spentList}^\epsilon_{\text{RID}}$, the reader computes the truncated hash-value of *chall* using K_ϵ, and sends the result back to T_{Alice}. T_{Alice} verifies received verification bits to authenticate the reader. In ⑤, if the verification bits match, T_{Alice} sends *precoin*, the preimage of the *coin* to the reader. Otherwise, T_{Alice} assumes malicious behavior and sends $prefake$, the preimage of $fake$. This exchange of *precoin* or $prefake$ after the two coins achieves the commitment of T_{Alice} without allowing an adversary (impersonating a legitimate reader) to determine if the coin it receives is a fake. In ⑥, the reader verifies if the hash of the received preimage (*precoin* or $prefake$) matches coin. In case the hash does not match, *vcoin* is included in another simple list, $\text{reimburseList}^\epsilon_{\text{RID}}$. Also, if the protocol gets somehow interrupted, and the reader does not receive the last message, the reader will include *vcoin* in $\text{reimburseList}^\epsilon_{\text{RID}}$. Although the barrier does not open in the last two cases, and the coin is "spent", coins on $\text{reimburseList}^\epsilon_{\text{RID}}$ can be later reimbursed to Alice by the bank as described in Section 6.2. In general, if the reader suspects any misbehavior or cheating during PSP, it sleeps ("sleep") for a reasonable amount of time, e.g., 10 seconds, and exits protocol execution (**exit**). Also, an alarm might go of, security personnel arrives etc.

T_{Alice}	Reader RID
`// Receive` RID	$T_{\text{Alice}} \longleftarrow$ RID : {RID} ①
$chall := h(\text{ID}^{\epsilon}_{\xi^\epsilon}, c)$;	
$precoin := h(\text{RID}, \text{ID}^{\epsilon}_{\xi^\epsilon}, c)$;	
$coin := h(precoin)$;	
$prefake := h(\text{RID}, c, \text{ID}^{\epsilon}_{\xi^\epsilon})$;	
$fake := h(prefake)$;	
$b := \lfloor h(\text{ID}^{\epsilon}_{\xi^\epsilon}, c, \text{ID}^{\epsilon}_{\xi^\epsilon}) \rfloor_1$;	
$coins := \text{flip}(b, \{coin, fake\})$;	
$c := c + 1$;	
② $T_{\text{Alice}} \longrightarrow$ RID : $\{coins, chall, \epsilon\}$	`// Receive:` $\{coin_1, coin_2\}$
	if isElement($\text{BF}^{\epsilon}_{\text{RID}}, coin_1$) **then** ③
	$vcoin := coin_1$;
	elseif isElement($\text{BF}^{\epsilon}_{\text{RID}}, coin_2$) **then**
	$vcoin := coin_2$;
	else sleep; **exit**;
	if isNotElement($\text{spentBF}^{\epsilon}_{\text{RID}}, vcoin$)
	and isNotElement($\text{spentList}^{\epsilon}_{\text{RID}}, vcoin$)
	then ④
	addList($\text{spentList}^{\epsilon}_{\text{RID}}, vcoin$);
	$v := \lfloor h(K_\epsilon, chall) \rfloor_\omega$;
	$T_{\text{Alice}} \longleftarrow$ RID : $\{v\}$
`// Authenticate reader`	
if $v = \nu_c$ **then**	
`// Finish payment` ⑤	
$T_{\text{Alice}} \longrightarrow$ RID : $\{precoin\}$	`//`*received* `is` *precoin* `or` *prefake*
else $T_{\text{Alice}} \longrightarrow$ RID : $\{prefake\}$	**if** $h(received) = vcoin$ ⑥ **then**
	openBarrier;
	else
	addList($\text{reimburseList}^{\epsilon}_{\text{RID}}, vcoin$);
	sleep; **exit**;
	end
	end
	else sleep; **exit**;

Algorithm 4: Payment procedure

5.4 Periodic Maintenance

Once a day all readers connect to the bank, either simultaneously or using some kind of load-balancing schedule mechanism. The current epoch is ϵ_i.

1.) All readers send their $\{\text{spentList}^{e_{i-1}}_{\text{RID}}, \text{spentList}^{e_i}_{\text{RID}}, \text{reimburseList}^{\epsilon_{i-1}}_{\text{RID}}, \text{reimburseList}^{\epsilon_i}_{\text{RID}}\}$, with $1 \leq \text{RID} \leq \rho$, to the bank. All readers remove all entries from their spentLists and reimburseLists.

2.) The bank now checks all these lists, whether the included entries, *coin*s of the form $h(h(\text{RID}, \text{ID}^{\epsilon}_{\xi^\epsilon}, c))$, have already been spent, cf., Algorithm 5.

With getValue(x, y), hash-table x is queried with key y the corresponding value is returned. ModifyHashValue(x, y, z) sets the value belonging to key y in hash-table x to z.

What happens in Algorithm 5 is basically that the bank does a reverse lookup for each spent coin to get the information to compute the corresponding ID and counter pair – using hash-table Δ_ϵ. Therewith, the bank can find the name of the user this ID was issued and the information whether this particular coin has been spent already. If this coin has been spent, the bank can take appropriate countermeasures against $Name$. If a coin is on a reimburseList of reader RID and not on *any* other reader's spentList, the bank will reimburse this coin to $Name$. Finally, the bank computes for each reader RID a coin and sends it to

```
for ε ∈ {ε_{i−1}, ε_i} do
    for RID := 1 to ρ do
        foreach entry ∈ spentList^ε_RID do
            {ξ^ε, ε, c} := getValue(Δ_ε, entry);
            ID^ε_{ξ^ε} := h(K_B, ξ^ε, ε);
            {Name, spent} := getValue(Σ_ε, {ξ^ε, ε, c});
            if spent=1 then
                reportIdentity(Name);// Cheating
                    detected
            else
                modifyHashValue(Σ_ε, {ξ^ε, ε, c}, {Name, 1});
                if entry ∈ reimburseList^ε_RID and
                entry ∉ spentList^{ε_{i−1}, ε_i}_{≠RID} then
                    reimburseCoin(Name);
                for j := 1 to ρ do
                    coin := h(h(j, ID^ε_{ξ^ε}, c));
                    Bank⟶ Reader j: {coin}
                end
            end
        end
    end
end
```

Algorithm 5: Periodic maintenance

RID. Readers add this coin to their spentBF Bloom filters: addBF(spentBF$^{\epsilon}_{\text{RID}}$, *coin*). Also, the bank pays the metro with real money (1$, £, €) for one coin.

5.5 Adding Flexibility

In this section, we extend PSP with respect to more flexibility of charging tags and payment as well as distributing the bank's workload to "proxies".

Money. Instead of charging tags only with $\gamma_{\max}$ coins all at once, tags can be charged with sets of coins called *packs* π. Possible packs are, for example, $\pi \in \{10, 20, 50\}$ coins. So, Alice can buy packs of 10 coins, 20 coins, 50 coins, as well as combinations thereof. For every pack of coins that Alice buys at the bank, the bank will handout one ID to Alice. Still, a tag can never be charged with more than a total of $\gamma_{\max}$ coins on a tag simultaneously. Also, Alice cannot buy more than, say, up to 3 packs per charge. If a tag stores coins from more than one pack, it will always completely deplete one pack for payment before using coins generated out of another pack. (Re-)Charging is only allowed, as soon as the total number of coins on the tag is less or equal than 9 coins. If a tag is recharged, it will first deplete the remaining old (≤ 9) coins before using the new ones. So in conclusion, simultaneously, a tag has up to $\gamma_{\max}$ coins stored in a total of ≤ 4 packs.

Again utilizing statistics, the metro knows on average how many packs $\pi_i \in \{10, 20, 50\}$ of coins will be bought by users (including a safety margin) during one epoch. The expected number of packs of size 10, $\pi_i = 10$, is η_{10}, η_{20} for packs $\pi_i = 20$, and η_{50} for $\pi_i = 50$. In conclusion, $10 \cdot \eta_{10} + 20 \cdot \eta_{20} + 50 \cdot \eta_{50} = \eta$.

Now, IDs can be computed as: $1 \leq i \leq 3$, $\pi_i \in \{10, 20, 50\}$, $\forall k : 1 \leq k \leq \eta_{\pi_i}$, $\text{ID}_k^{\pi_i,\epsilon} = h(K_B, i, k, \epsilon)$. The bank maintains 3 *counters* $(\xi^{\epsilon}_{10}, \xi^{\epsilon}_{20}, \xi^{\epsilon}_{50})$ to store the information about which IDs have already been sold to tags. So, $1 \leq \xi^{\epsilon}_{10} \leq \eta_{10}$, $1 \leq \xi^{\epsilon}_{20} \leq \eta_{20}$, $1 \leq \xi^{\epsilon}_{50} \leq \eta_{50}$. During charging, Alice gives the equivalent amount of money to buy $i \leq 3$ packs of coins, $\pi_i \in \{10, 20, 50\}$. In return, Alice's tag T_{Alice} receives and stores i tuples consisting of $\text{ID}^{\pi_i,\epsilon}_{\xi^{\epsilon}_{\pi_i}}$, counter $c_i := 1$, $max_i := \pi_i$, and verification bits $\nu_{i,j}$, $1 \leq j \leq \pi_i$.

If a metro trip costs a total of α coins instead of 1 coin, reader RID broadcasts α together with RID. As soon as T_{Alice} receives this broadcast, it executes the payment protocol of Algorithm 4 α times. T_{Alice} completely depletes coins of one pack before it starts using coins from another.

Proxies. To decrease the workload of the bank, we introduce *proxies*. Readers are not directly, i.e., physically, connected to the bank, but multiple readers are grouped together and physically connected to a bank's *proxy* device, a more powerful computer. For example, readers of metro stations in close physical distance are physically connected to one proxy, buses returning to the same bus garage in the evening are connected to one proxy in the evening. A proxy is connected physically to the bank and will carry out all communication between bank and readers. Readers, even the ones connected to the same proxy, are not assumed to be permanently connected to their proxies. This would be impossible, e.g., for readers in buses. Readers cannot exchange data with each other and are offline most of the time. Readers connect to their proxies once a day during maintenance. Proxies could collect all readers' spentLists and reimburseLists and relay them to the bank. Proxies do not need to be permanently online-connected to the bank, but also only once a day. As with the bank, we assume the proxies, but not the readers, to be trusted by using tamper-proof hardware. Proxies are under full control of the bank (and not the metro), and thus trusted by users. Secret key K_B is not only known to the bank, but also to the proxies. During maintenance, the bank does not compute and send spent coins to all readers, but only sends $\{\text{ID}^{\epsilon}_{\xi^{\epsilon}}, c\}$, as in Algorithm 5, to all proxies. Proxies then compute and send coins for all their attached readers, $coin := h(h(\text{RID}, \text{ID}^{\epsilon}_{\xi^{\epsilon}}, c))$. There are σ proxies in the system.

5.6 Real World Parameters

For evaluation, we assume system parameters similar to Oyster Card. Between 2003 and 2007, 10^7 Oyster Cards have been issued [13] from which $\tau = 5 \cdot 10^6$ are in use at the same time [27] (some Oyster Cards are not rechargeable and dropped after use). In 2007, the total revenue of public transport with buses and metro in London was 2.420 billion GBP [31]. If all transport would have been paid with Oyster Cards (still traditional methods of payment, such as cash, are used), then $\eta \approx 10^8$ per epoch. Adopting the Oyster Card setting, we assume $\gamma_{\max} = 80$.

In London, there are 270 metro stations and 6,800 buses [28, 30]. We assume that there are in the average 50 readers per metro station and one reader per bus. As a total, we assume $\rho = 20,000$ readers. By assuming a total number of $\sigma = 20$ proxies in the system, each proxy is on average associated to 1,000 readers.

We choose $\kappa = 22$, resulting in 2^{-22} probability of a single false positive. With $\omega = 1$, the adversary can impersonate a reader with 50%, but as discussed in Section 6, this is acceptable in the overall scenario. Finally, $|Name| = 32$ bit should be sufficient to uniquely identify a single account or user of the payment system.

5.7 Space Analysis

Based on the more flexible extension of PSP and the real world parameters, we can do the following evaluation:

Tag. T_{Alice} stores 4 IDs, 4 c_i, 4 ϵ_i, 4 $\max_i$ to be able to generate coins. $|\text{ID}| = 128$ bit, $|\max_i| = |c_i| = 6$ bit, $|\epsilon_i| = 8$ bit. This sums up to 592 bit. Also, T_{Alice} stores $(\gamma_{\max} \cdot \omega)$ verification bits ν, i.e., $80 \cdot 1 = 80$ bit. Each tag needs to store 672 bit = 84 byte in its non-volatile memory. This is feasible, e.g., with Alien Technology's prominent Higgs-3 RFID-tag [1], featuring 800 bit of non-volatile storage. Compared to the Oyster Card, featuring 1 KByte (=6144 bit) storage [21], this is much less and thus leads to cheaper tags in terms of production costs.

Generally for tags, computational complexity is important. However, computational complexity is low with PSP: for payment, the tag has to do only hash evaluations, simple arithmetic, and to wirelessly send $3 \cdot 128 + 8 = 392$ bit to the reader. Based on related work [2, 10, 18, 22, 32, 37], we consider that these operations are feasible on tags.

Reader. A standard Bloom filter with false-positive probability P and η elements of a set to represent requires $\mu = \frac{\log_2(\frac{1}{P}) \cdot \eta}{\ln 2}$ bit of storage [4]. So with $\eta = 10^8$, each reader needs $\mu \approx 378$ MByte of storage for the BF Bloom filter and the same for the spentBF Bloom filter. As BF and spentBF are required for the current and second-to-last epoch, this would amount to a total of $4 \cdot 378 \approx 1.5$ GByte per reader. However, standard Bloom filters are not space optimal. Optimizations of Bloom filters can achieve lower storage re-

quirements, such as $\log_2(\frac{1}{P})$ bit per element represented in the filter. For example, Putze et al. [23] suggest to build the Bloom filter with a single hash function instead of $\kappa \gg 1$ hash functions. In that optimized version, the hash outputs $h_1(x)$ of all coins are Golomb encoded and partitioned into blocks. The isElement function that looks for *coin* is implemented by selectively decoding one block and looking for the hash output $h_1(coin)$. The data structure of Putze et al. [23] maintains exactly the same properties as a standard Bloom filter, i.e., false-positive probability P, but requires only $\log_2(\frac{1}{P})$ bit storage per element. In our case, this optimization would results in ≈ 262 MByte total storage for each BF filter. Similar to Counting Bloom Filters, Putze et al. [23] allows furthermore for deleting coins out of the data structure, superseding the spentBF Bloom filters on readers. The latter helps in avoiding false positives while checking whether a coin has been spent. Algorithm 4 can therefore be changed such that a reader accepts a coin if it is in its BF Bloom filter and at the same time not on its current spentList. Instead of $4 \cdot 378 \approx 1.5$ GByte, this would keep storage close to a total of $2 \cdot 262 = 524$ MByte for both epochs. Finally, per day, each reader needs on average to store $\frac{10^8}{30 \cdot 20,000} \approx 170$ coins, i.e., ≈ 3 KByte on both spentLists for the two epochs. Readers at frequently used metro stations will require more memory for spentLists, but this will still be in the order of magnitude of KBytes per day. Also, a small amount of memory is required for the two reimburseLists. In conclusion, the total amount of memory is considerably less than 1 GByte which should be feasible even on restricted reader hardware.

Computational complexity for the reader is also low. The isElement function required for validating a coin is cheap: it consists of performing a low complexity Golomb-decoding of a fixed length block and searching for $h_1(coin)$ therein [23].

Bank. The bank needs to store Δ and Σ for the current and last epoch. In Δ, for all 10^8 coins and all $20,000$ readers a *prechall* has to be stored. With $|prechall| = |\xi| + |i| + |\epsilon| + |c|$, and $|c| = \log_2 50 \approx 6$ bit, $|\epsilon| = 8$ bit, $|i| = \log_2 3 \approx 2$ bit, and $|\xi| = \log_2 \frac{10^8}{10} \approx 24$ bit (worst case: all packs bought are 10 coin packs), $|prechall| = 40$ bit worst case. So, the two Δs for the two epochs require $2 \cdot 20,000 \cdot 10^8 \cdot 40$ bit ≈ 18 TByte worst case. There are a total of 10^8 (ID, c_i) pairs, and for each pair Σ stores $Name$ and *spent*. For the two epochs, this requires $2 \cdot 10^8 \cdot (32+1)$ bit ≈ 790 MByte storage. While the resulting 18 TByte is certainly a huge amount of storage, we claim this is still affordable for a bank.

Computational complexity for the bank is low, too: preparation of a new epoch has to be done only once a month. The workload consisting of generating hash outputs for all coins and their Golomb encoding can be distributed among the $\sigma = 20$ proxy devices.

6. SECURITY AND PRIVACY ANALYSIS

The challenge of having secure and private payment with RFID tags is due to the lack of asymmetric cryptography and blind signatures. Authentication and encryption based on symmetric keys shared by tags and readers are not suitable either, since a globally shared key would allow the adversary to jeopardize the whole system through the compromise of a single tag. Sharing a different key per tag would affect privacy. As a result, PSP's mechanism of reader authentication uses precomputed challenge-response pairs.

6.1 Protection against fraudulent payment

Generally, it should not be possible for an adversary to spend coins he did not pay for and to spend the same coin more than once. The basic idea behind PSP's two staged payment procedure is that a valid *coin* without an according *precoin* is worthless for the adversary. A simpler version of the protocol solely based on the exchange of *coin* would allow the adversary to intercept message 2 of Figure 1 and deny its delivery to the reader. The adversary would then have successfully stolen one coin he could later use for his own payment. Consequently, T_{Alice} will only send *precoin* in message 4 after it authenticated the reader in message 3. If the adversary denies delivery of message 4, he might get a valid *coin*, *precoin* pair, but he still cannot use it, as the reader has already added *coin* on its spentList after message 2. Also, the adversary cannot use this *coin* with another reader, as coins are reader dependent by using RID.

Inventing coins. An adversary cannot "invent" new coins. If he bought a pack π of coins, he can only spend these π coins, because more coins are not in the readers' Bloom filters. So, knowledge of valid IDs does not give any advantage for the adversary over guessing IDs. The probability P_{invent} for an adversary to *invent* a coin correctly is the probability to really *guess* a single valid coin by chance ($P_{\text{chance}} = \frac{\text{valid coins}}{\text{possible coins}} = \frac{10^8}{2^{128}} \approx 2^{-100}$) plus the false-positive probability $P = 2^{-22}$, $P_{\text{invent}} = P_{\text{chance}} + (1 - P_{\text{chance}}) \cdot P$). P_{invent} is still very close to the false-positive rate P. As P_{chance} is negligible small, inventing a coin by guessing or brute-forcing a valid ID is unlikely. A rational adversary will focus on exploiting the false-positive property of Bloom filters. However, the adversary has to carry out such an attack *online*, by being close to the reader. Here, every time the adversary guesses a coin incorrectly, the reader sleeps for a reasonable amount of time (and possibly raises an alarm) such that this kind of attack quickly becomes too time consuming for the adversary. We claim that 2^{-22} probability to guess *one single* valid coin is secure enough in this scenario.

Reader Impersonation. An adversary might try to impersonate a reader, initiate communication with T_{Alice}, and guess the ω verification bits ν for *chall*. If his guess is wrong, T_{Alice} sends $prefake$ to him. If he is right, he receives *precoin* and has successfully stolen a valid {*coin*, *precoin*} pair. However, he does not now whether he guessed correctly as he cannot distinguish whether the data he receives is for the valid coin or the fake coin. He just knows that $h(received)$ matches either *coin* or $fake$. The pair he can compute is therefore either {$fake$, $prefake$} or {*coin*, *precoin*}. If he tries to pay with this pair, he always succeeds with probability $P_{\text{steal}} = 2^{-\omega} + (1 - 2^{-\omega}) \cdot P$, triggering an alarm etc. With $\omega = 1$, the adversary can steal successfully with $P_{\text{steal}} \approx 50\%$. This probability can be decreased by increasing ω at the cost of additional storage requirements on the tag. With $\omega = 1$, $\omega \cdot \gamma_{\max}$=10 byte are required. If the adversary should be able to steal a coin with only 2^{-10} probability, 100 byte of storage would be required. Generally, security can be adjusted depending on the physical properties of the tag. Stealing a coin from a tag requires much more effort from the adversary than just randomly generating coins and sending them to a reader: the adversary has to be physically close to the tag to send and receive messages. So, we claim a 50% probability of stealing *one single* valid coin to be reasonable, because the adversary does

never know whether each single coin is a fake or not. We claim that triggering an alarm with 50% probability per coin and the difficulty of mounting such an attack will prevent the adversary from stealing coins in practice.

Replay of coins. An adversary eavesdropping payment cannot replay a coin at the same reader, because the reader stores all spent coins either in its spentList (same day), or the reader stores spent coins in its spentBF Bloom filter (after maintenance, if spentBF is still used – see Section 5.7). In any case, the reader will reject this coin. If the adversary eavesdropped a payment at a reader, he cannot replay and pay with this coin at a different reader, because coins are reader dependent. If a malicious user spends a valid coin, i.e., using an (ID, c) pair, on one reader today and re-uses this pair with another reader on *another day* after maintenance, this reader will reject the coin as it is already on its spentBF. Only if a malicious user spends a valid coin on the *same day* with two *different* readers, readers cannot immediately detect cheating and will accept this coin. However, during the periodic maintenance at night, the bank will spot this kind of double spending, identify the malicious user using Algorithm 5, and ask for compensation.

6.2 DoS-Attacks and Reimbursement

PSP is clearly vulnerable against Denial-of-Service attacks: if the adversary repeatedly initiates communication with T_{Alice} and stops the protocol after the first message, the tag will increase counter c_i until, eventually, it cannot create new coins anymore. The tag is "exhausted" and refuses to operate until charged with money again. It is important to point out that the tag *must* increase c_i every time, because otherwise it will re-send the same *coin* on two subsequent protocol runs, therewith making it traceable.

Yet, the adversary is never able to steal money, but only to render all coins on the tag useless, i.e., a denial-of-service attack against all *coins* on T_{Alice}. As T_{Alice}'s coins are not spent, Alice can get reimbursed by the bank: the bank can verify that Alice's coins have never been added to spentBF Bloom filters. If the adversary replays $\{coin, fake\}$ pairs received from initiating communication with T_{Alice} to a reader, the reader will then add *coin* to its spentList, therewith marking this coin as "spent" in the whole system. To cope with this, each reader maintains reimburseLists. If a coin is spent, but the protocol is not successfully finished, the reader adds *coin* to its reimburseList. This allows the bank to easily reimburse Alice her money during periodic maintenance, cf., Algorithm 5. So in conclusion, Alice never loses her money.

6.3 Privacy

According to the definition of the privacy game in Juels and Weis [18], PSP does not guarantee *strong privacy*: in the LEARNING phase, the adversary calls the SETKEY oracle to compromise $(\tau - 2)$ tags, so two tags, T_0 and T_1, remain uncompromised. He now initiates communication with T_0 a total of $\gamma_{\max}$ times, but stops protocol execution after receiving T_0's first message each time. Eventually, T_0 is "exhausted", cannot produce additional coins, and, for example, refuses to operate until recharge. Now, in the CHALLENGE phase, the adversary is presented with one of the two tags. If this tag is replying to his communication, he knows with 100% probability that it is T_1, otherwise it is T_0.

However, we claim DoS-attacks like the above exhaustion of coins in the *strong privacy* model to be unrealistic: Alice would notice her tag not working anymore and become suspicious, and the metro cannot afford DoS attacks against their customers' tags, as customers would quickly start complaining. So, in the absence of DoS-attacks, the information sent from tags to readers, i.e., *coin*s, *chall*s, *precoin*s, looks completely random for an adversary as well as for the metro in each protocol run. Only ϵ will repeat, spoiling *strong privacy*. Yet, as there is potentially a large number of tags having the same ϵ for many *coin*s, we claim this to offer a good enough privacy in the set of all tags.

7. CONCLUSION

Secure, privacy-preserving, offline electronic payments only using tiny RFID tags is a new and challenging problem. In this paper, we presented PSP, a solution minimizing computational requirements for the tag, but still offering protection against overspending and privacy against payees. Adversaries cannot invent new coins, replay, or steal coins from legitimate users of the system. Payees, e.g., a metro system, cannot trace or link subsequent transactions of users to the same tag. Privacy is assured. Tags are only supposed to evaluate a hash function and store 84 byte in non-volatile memory. Readers can be offline most of the time and connect only rarely for synchronization. Future work will adopt and extend PSP to different ecash scenarios, for example, supporting multiple different, untrusted payees.

References

[1] Alien Technology. Rfid tags, 2009. `http://www.alientechnology.com/tags/index.php`.

[2] G. Avoine, E. Dysli, and P. Oechslin. Reducing time complexity in rfid systems. In *Proceedings of Selected Areas in Cryptography*, pages 291–306, Kingston, Canada, 2005. ISBN 978-3-540-33108-7.

[3] S. Brands. Untraceable off-line cash in wallets with observers. In *Proceedings of Annual International Cryptology Conference*, pages 302–318, Santa Barbara, USA, 1993. ISBN 3-540-57766-1.

[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4): 485–509, 2003. ISSN 1542-7951.

[5] D. Chaum. Blind signatures for untraceable payments. In *Proceedings of Annual International Cryptology Conference*, pages 199–203, Santa Barbara, USA, 1982.

[6] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings of Annual International Cryptology Conference*, pages 319–327, Santa Barbara, USA, 1988. ISBN 3-540-97196-3.

[7] Y. Choi, M. Kim, T. Kim, and H. Kim. Low power implementation of sha-1 algorithm for rfid system. In *Proceedings of Tenth International Symposium on Consumer Electronics*, pages 1–5, St. Petersburg, Russia, 2006. ISBN 1-4244-0216-6.

[8] R. Cramer and I. Damgård. Introduction to secure multi-party computations. In *Contemporary Cryptology: Advanced Courses in Mathematics*, pages 41–87. Birkhauser, 2005. ISBN 3-7643-7294-X.

[9] Y. Desmedt, C. Goutier, and S. Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *Proceedings of Annual International Cryptology Conference*, pages 21–39, Santa Barbara, USA, 1987. ISBN 3-540-18796-0.

[10] T. Dimitrou. rfiddot: Rfid delegation and ownership transfer made simple. In *Proceedings of International Conference on Security and privacy in Communication Networks*, Istanbul, Turkey, 2008. ISBN 978-1-60558-241-2.

[11] F.D. Garcia, G.K. Gans, R. Muijrers, P. Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs. Dismantling mifare classic. In *Proceedings of European Symposium on Research in Computer Security*, pages 97–114, Malaga, Spain, 2008. ISBN 978-3-540-88312-8.

[12] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *Proceedings of World Wide Web Conference*, pages 604–618, Boston, USA, 1995.

[13] Greater London Authority. Mayor to give away 100,000 free oyster cards, 2007. `http://www.london.gov.uk/view_press_release.jsp?releaseid=11611`.

[14] G.P. Hanke. Practical attacks on proximity identification systems. In *Proceedings of Symposium on Security and Privacy*, pages 328–333, Oakland, USA, 2006. ISBN 0-7695-2574-1.

[15] G.P. Hanke and M.G. Kuhn. An rfid distance bounding protocol. In *Proceedings of International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 67–73, Athens, Greece, 2005. ISBN 0-7695-2369-2.

[16] InformationWeek. Visa debuts rfid-enabled card-payment system, 2005. `http://www.informationweek.com/news/mobility/RFID/showArticle.jhtml?articleID=60403344`.

[17] InformationWeek. Rfid helps feed parking meters, 2005. `http://www.informationweek.com/news/mobility/RFID/showArticle.jhtml?articleID=174900727`.

[18] A. Juels and S.A. Weis. Defining strong privacy for rfid. In *PerCom Workshops*, pages 342–347, White Plains, USA, 2007. ISBN 978-0-7695-2788-8.

[19] C.H. Lim and T. Kwon. Strong and robust rfid authentication enabling perfect ownership transfer. In *Proceedings of Conference on Information and Communications Security*, pages 1–20, Raleigh, USA, 2006. ISBN 3-540-49496-0.

[20] S. Micali and R.L. Rivest. Micropayments revisited. In *Proceedings of RSA conference*, pages 149–163, San Jose, USA, 2003. ISBN 3-540-43224-8.

[21] NXP Semiconductors. Mifare 1k, 2009. `http://mifare.net/products/smartcardics/mifare_standard1k.asp`.

[22] R. Di Pietro and R. Molva. Information confinement, privacy, and security in rfid systems. In *Lecture Notes in Computer Science, Volume 4734*, pages 187–202, 2007. ISBN 978-3-540-74834-2.

[23] F. Putze, P. Sanders, and J. Singler. Cache-, hash- and space-efficient bloom filters. In *Proceedings of Workshop on Experimental Algorithms*, pages 23–36, Rome, Italy, 2007. ISBN 978-3-540-72844-3.

[24] R.L. Rivest. Peppercoin micropayments. In *Proceedings of Financial Cryptography*, pages 2–8, Key West, USA, 2004. ISBN 3-540-22420-3.

[25] R.L. Rivest and Adi Shamir. Payword and micromint–two simple micropayment schemes. In *Proceedings of International Workshop on Security Protocols*, pages 69–87, Paris, France, 1997. ISBN 3-540-64040-1.

[26] A. Shamir. Squash — a new mac with provable security properties for highly constrained devices such as rfid tags. In *Proceedings of Fast Software Encryption (FSE)*, pages 144–157, Lausanne, Switzerland, 2008. ISBN 978-3-540-71038-7.

[27] The Guardian. Oyster data use rises in crime clampdown, 2006. `http://www.guardian.co.uk/technology/2006/mar/13/news.freedomofinformation`.

[28] Transport for London. London busses, 2009. `http://www.tfl.gov.uk/corporate/modesoftransport/1548.aspx`.

[29] Transport for London. Oyster online, 2009. `https://oyster.tfl.gov.uk/oyster/entry.do`.

[30] Transport for London. Key facts, 2009. `http://www.tfl.gov.uk/corporate/modesoftransport/londonunderground/1608.aspx`.

[31] Transport for London. London travel report 2007, 2008. `http://www.tfl.gov.uk/assets/downloads/corporate/London-Travel-Report-2007-final.pdf`.

[32] G. Tsudik. Ya-trap: yet another trivial rfid authentication protocol. In *Proceedings of International Conference on Pervasive Computing and Communications Workshops*, Pisa, Italy, 2006. ISBN 0-7695-2520-2.

[33] H.C.A. van Tilborg, editor. *Encyclopedia of Cryptography and Security*. Springer Verlag, 2005. ISBN 038723473X.

[34] S. Vaudenay. On privacy models for rfid. In *Proceedings of ASIACRYPT*, pages 68–87, Kuching, Malaysia, 2007. ISBN 978-3-540-76899-9.

[35] G. Venkataramani and S. Gopalan. Mobile phone based rfid architecture for secure electronic payments using rfid credit cards. In *Proceedings of International Conference on Availability, Reliability and Security*, pages 610–620, Vienna, Austria, 2007. ISBN 0-7695-2775-2.

[36] Visa USA. Visa paywave, 2009. `http://usa.visa.com/personal/cards/paywave/index.html`.

[37] S.A. Weis, S.E. Sarma, R.L. Rivest, and D.W. Engels. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing*, pages 201–212, Boppard, Germany, 2003. ISBN 3-540-20887-9.

[38] Wired. Mcdonald's tries out new rfid-enabled pay-by-phone coupons, 2008. `http://blog.wired.com/gadgets/2008/05/mcdonalds-tries.html`.