Scalable RFID Security Protocols supporting Tag Ownership Transfer

Boyeon Song[a,1], Chris J. Mitchell[a,1]

[a] *Information Security Group, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, UK*

Abstract

We identify privacy, security and performance requirements for RFID protocols, as well as additional functional requirements such as tag ownership transfer. Many previously proposed protocols suffer from scalability issues because they require a linear search to identify or authenticate a tag. In support of scalability, some RFID protocols, however, only require constant time for tag identification, but, unfortunately, all previously proposed schemes of this type have serious shortcomings. We propose a novel scalable RFID authentication protocol based on the scheme presented in [1], that takes constant time to authenticate a tag. We also propose secret update protocols for tag ownership and authorisation transfer. The proposed protocols possess the privacy, security and performance properties and meet the requirements for secure ownership transfer identified here.

1. Introduction

Radio Frequency Identification (RFID) tags have been widely studied by both academia and industry [2, 3, 4]. Such tags can be attached to objects, including products, animals or people, and can subsequently be used to identify them using radio communications.

An RFID system consists of tags, readers and a back-end server. A tag is typically made up of an antenna for receiving and transmitting a radio-frequency (RF) signal, and an integrated circuit for modulating and demodulating the signal and storing and processing information. When a back-end server wants to identify one or more tags, a reader emits an RF signal via its antenna. Any tag within range of the signal responds with certain stored data, such as a tag identifier. The reader then passes the received tag data to the back-end server for further processing, including tag identification and information retrieval.

Key features of RFID systems include a lack of physical contact between readers and tags, and tag scanning out of the line of sight [2, 3]. Moreover,

Email addresses: b.song@rhul.ac.uk (Boyeon Song), c.mitchell@rhul.ac.uk (Chris J. Mitchell)

a smart tag possesses storage and processing capabilities, and can also perform lightweight cryptographic functions. These properties mean that RFID tags have many possible applications, such as product management, transport payments, livestock tracking, library book administration, patient medical care and e-passports. However, the technology also poses threats to user privacy, including the possibilities of user information leakage and location tracking.

A considerable volume of papers have been published providing possible solutions to these RFID security and privacy challenges. One approach to protecting against such privacy and security threats is to use a tag authentication scheme in which a tag is both identified and verified in a manner that does not reveal the tag identity to an eavesdropper. A large number of tag authentication protocols of this type have been proposed. Typically, pseudonyms are used to provide anonymity to tags; whenever a tag is queried, it responds with a different cryptographically derived pseudonym. In some of these pseudonym-based protocols, see for example [5, 6, 7, 8], a back-end server must perform a linear search of its database to identify a tag. That is, for each tag entry in the database in turn, it computes the pseudonym that would be produced by that tag (using stored secrets) and compares it with the received pseudonym. Such a linear search runs in $O(n)$ time, where n is the number of elements in the back-end database. Such a costly search function will potentially cause scalability issues as the tag population increases.

Scalability is a desirable property in almost any system, enabling it to handle growing amounts of work in a graceful manner [9]. A scalable RFID system should be able to handle large numbers of tags without undue strain, and a scalable RFID protocol should therefore avoid any requirement for work proportional to the number of tags.

In support of scalability, some RFID pseudonym schemes, see for example [10, 11, 12, 13], require only $O(1)$ work to identify a tag[1]. Most such schemes use look-up tables to match a value with a pseudonym received from a tag, thereby taking a constant time without the need for a linear search. However, all previously proposed schemes of this type possess significant security, privacy or performance shortcomings, as discussed in section 3.

An alternative means of improving the scalability of an RFID system is delegation. Tag delegation involves giving authorised entities the right to query and identify certain tags during a specified period. This clearly has the potential to reduce the back-end server's workload.

Another possible requirement for RFID systems is secure tag ownership transfer. In some applications, the owner of an RFID tag may change a number of times during its lifetime. Ownership transfer means that the server of the new owner takes over tag authorisation, and so needs to be given the necessary private information to securely interact with and identify the tag. Thus all in-

[1]Strictly speaking the work is $O(\log_2 n)$, where n is the number of tags, but in practice this amounts to constant work, since $\log_2 n$ is unlikely to exceed the word length of a modern PC; we therefore refer throughout to $O(1)$.

formation associated with the tag will need to be passed from the old to the new owner. However, at the moment of tag ownership transfer, both the old and new owners have the information necessary to authenticate a tag, and this fact may cause an infringement of tag owner privacy. More specifically, if the previous owner is malicious, it may still be able to read the tag using retained tag information after transfer, and/or trace the new owner's transactions with the tag. That is, the privacy of the new owner might be compromised by the previous owner. Conversely, if the new owner is malicious, then it might be able to trace the previous owner's past transactions with the tag. That is, the privacy of the previous owner might be compromised by the new owner.

The goal of this paper is to propose a scalable and efficient RFID pseudonym protocol having desirable privacy and security properties. To provide scalability, our novel protocol possesses two features, namely that a server takes only constant time to identify a tag, and tag delegation is straightforward. We also examine the requirements for secure tag ownership transfer, and propose RFID authentication protocols satisfying such requirements. These protocols originate from the scheme introduced in [8] (referred to here as the SM protocol).

We first identify desirable privacy, security and performance requirements for RFID protocols as well as the specific requirements for tag ownership transfer in section 2. Section 3 introduces related work and section 4 introduces a revised version of the SM protocol. Section 5 proposes a scalable RFID pseudonym protocol and section 6 proposes new protocols for secure tag ownership transfer, designed to provide the identified properties. We next analyse these protocols against the identified requirements. Finally we summarise the contributions of the paper.

2. RFID Protocol Requirements

2.1. Assumptions

The RFID protocols considered here operate under the following assumptions.

- The communicating parties are a server and a tag.
- The term server is used to mean a combination of a back-end server and its readers.
- A server and tag communicate via an insecure RF interface.
- A server maintains a secure database of information for the tags that it owns, and has a significantly greater processing capability than a tag.
- A tag has a rewritable memory that may not be tamper-resistant, and can perform lightweight cryptographic operations.

We next introduce privacy issues, security threats and performance characteristics relevant to RFID protocols.

2.2. Privacy

One of the main concerns of users of RFID systems is user privacy. Unprotected communications between a tag and a server over a wireless channel can disclose information about a tag, including its location (and, by implication, the location of its owner).

2.2.1. Threats

Two major privacy issues are as follows [2, 3, 14, 7, 15].

- **Tag Information Leakage**: in a typical RFID system, when a server queries a tag, the tag responds with its identifier. If unauthorised entities can also obtain a tag identifier, then they may be able to request and obtain the private information related to the tag held in the server database. For example, if the information associated with a tag attached to a passport, ID-card or medical record could be obtained by any server, then the damage would be very serious.

- **Tag Tracking**: if the responses of a tag are linkable to each other or distinguishable from those of other tags, then the location of a tag could be tracked by multiple collaborating unauthorised entities. For example, if the response of a tag to a server query is a static ID code, then the movements of the tag can be monitored, and the social interactions of an individual carrying a tag may be available to third parties without his or her knowledge.

2.2.2. Privacy Requirements

RFID systems should meet the following privacy requirements in order to mitigate the two threats described above.

- **Tag Information Privacy**: RFID systems should be able to resist tag information leakage. To protect against such a threat, RFID systems need to be controlled so that only authorised entities are able to access the information associated with a tag.

- **Tag Location Privacy**: RFID systems should be able to resist tag tracking attacks. If messages from tags are anonymous, then the problem of tag tracking by unauthorised entities can be avoided.

2.3. Security

Communications between a server and a tag via an insecure wireless channel are susceptible to eavesdropping. Security threats relevant to RFID systems are discussed below.

2.3.1. Attack Model

We divide possible attackers into two groups, as follows.

- A weak attacker (WA) is a malicious entity that can observe and manipulate communications between a server and a target tag, but cannot compromise the tag.
- A strong attacker (SA) is a malicious entity that has compromised a target tag, in addition to having the capabilities of a weak attacker.

The memory of a low-cost tag is unlikely to be tamper-resistant, and thus a tag's internal data are liable to exposure via side-channel attacks [16, 12]. Hence, threats by an SA as well as a WA should be considered in RFID protocol design.

Security threats to RFID systems can be classified into weak and strong attacks, in line with the attacker types defined above.

2.3.2. Weak attacks

The following attacks are feasible for a WA [2, 3, 15].

- **Tag Impersonation**: a WA could impersonate a target tag to a server without knowing the tag's internal secrets. It could communicate with a server instead of the tag and be authenticated as the tag.
- **Replay attack**: a WA could replay messages exchanged between a server and a tag without being detected, thereby performing a successful authentication between a tag and a server.
- **Man-in-the-Middle (MitM) attack**: a WA could interfere with messages sent between a server and a tag (e.g. by insertion, modification or deletion).
- **Denial-of-Service (DoS) attack**: a WA could block messages transmitted between a server and a tag. Such an attack could cause the server and the tag to lose synchronisation. For example, the server might update its shared secrets, while the tag does not; as a result, they would no longer be able to authenticate one another.

2.3.3. Strong attacks

An SA may be able to perform the following attacks, as well as the weak attacks described above [2, 12, 7].

- **Backward Traceability**: an SA might be able to trace past transactions between a server and a compromised tag using knowledge of the tag's internal state. That is, given knowledge of the internal state of a target tag at time t, the attacker is able to identify target tag interactions that occurred at time $t' < t$. The past transcripts of a tag may allow tracking of the tag owner's past behaviour.

- **Forward Traceability**: an SA might be able to trace future transactions between a server and a compromised tag using knowledge of the tag's internal state. That is, knowledge of a tag's internal state at time t can help to identify tag interactions that occur at time $t' > t$. The only way of maintaining future security once the current tag secrets have been revealed is to replace compromised secrets as soon as possible after they have been compromised (and hence undetected compromises remain an ongoing concern).

- **Server Impersonation**: an SA might be able to impersonate a legitimate server to a compromised tag using knowledge of the tag's internal state. This attack does not appear to have been widely discussed previously, despite its potential importance. The SA could, for example, ask the tag to update its internal state, with the effect that the legal server will no longer be able to communicate successfully with the tag, although the SA will. Details of such an attack can be found in [17].

Resistance to backward traceability is sometimes also referred to as *forward security* [5, 18, 7, 19]. In the paper, we use the terms backward traceability and forward traceability defined as in [12] in order to clearly distinguish between threats to past and future anonymity.

2.3.4. Security requirements

We identify the following security requirements for RFID systems designed to mitigate the threats of the weak and strong attacks described above.

- **Resistance to Tag Impersonation**: an adversary should not be able to impersonate a tag without compromising a tag.

- **Resistance to Replay attack**: an adversary should not be able to reuse messages exchanged between a server and a tag, thereby performing a successful session between the tag and the server.

- **Resistance to MitM attack**: an adversary should not be able to manipulate messages sent between a server and a tag without compromising a tag.

- **Resistance to DoS attack**: selective blocking of messages transmitted between a server and a tag should not mean that the server and the tag can no longer communicate successfully.

- **Backward Untraceability**: an adversary should not be able to to trace past transactions between a server and a tag, even if it compromises the tag.

- **Forward Untraceability**: an adversary should not be able to to trace future transactions between a server and a tag, even if it compromises the tag.

- **Resistance to Server Impersonation**: an adversary should not be able to impersonate a server to a tag, even if it compromises that tag.

2.4. Performance Requirements

RFID schemes cannot use computationally intensive cryptographic algorithms to provide privacy and security because tight tag cost requirements put severe limits on tag-side resources (such as processing power, memory and stored energy). RFID schemes should thus address the following performance issues [2, 16, 20, 14, 21, 15].

- **Storage Capacity**: the volume of data stored in a tag should be minimised.
- **Computation**: the complexity of tag computations should be minimised.
- **Communication**: the number and size of messages exchanged between a tag and a reader should be minimised.
- **Scalability**: the server should be able to handle a large tag population. It should be able to identify multiple tags using the same radio channel. Requiring a server to perform work proportional to the number of tags (e.g. as would be required for an exhaustive search to identify individual tags) becomes infeasible when the number of tags is large. Hence, it is desirable to design protocols for which the server complexity is $O(1)$ or $O(\log n)$ rather than $O(n)$, given a population of n tags.

2.5. Additional Functional Requirements

In this section, we introduce two functional requirements, tag delegation and ownership transfer, that are likely to be required in some RFID systems.

2.5.1. Tag Delegation

In RFID systems, a centralised back-end server is often in charge of a large number of tags. In some systems in which tag responses are anonymous, tag identification requires the back-end server to compute every possible tag output in turn until it finds a match [22]. This can seriously damage scalability [22].

A related issue is that many protocols require a reader to interact with the centralised back-end server in order to identify a tag [22]. In some applications, this reading latency can be an unacceptable overhead [22]. In addition, if the database becomes unavailable for some reason, such as network connectivity failure, all interactions with tags relying on that back-end server will be stopped [22].

Delegation is one possible solution to these performance issues [22]. Delegation enables a back-end server to delegate the right to identify and authenticate a tag to a specified entity, such as a reader [12, 23, 22].

Delegation may be permanent or temporarily [22]. In the first case, a reader is given permanent means to interact with a tag in its read range, and the back-end server is contacted only when a new tag arrives or an old tag leaves the

system [22]. In the second case, the back-end server temporarily transfers the right to interact with a set of tags for a limited number of queries, and updates or revokes the delegation capability according to a delegation policy [22].

2.5.2. Tag Ownership Transfer

Tag ownership means having authorisation to identity a tag and control all the related information [12, 23]. Tag ownership transfer implies a shift of such capabilities to a new owner [12, 23].

In certain RFID systems, changes of tag owner could occur frequently, and thus a secure and privacy-preserving means of tag ownership transfer is needed. This feature is supported by some RFID protocols.

The following requirements for secure tag ownership transfer have been identified [24, 12, 25]:

- **New owner privacy**: Once ownership of a tag has been transferred to a new owner, only the new owner should be able to identify and control the tag. The previous owner of the tag should no longer be able to identify or trace the tag.
- **Old owner privacy**: When ownership of a tag has been transferred to a new owner, the new owner of a tag should not be able to trace past interactions between the tag and its previous owner.
- **Authorisation recovery**: In some special cases, such as after-sales service for an RFID tagged object, the previous owner of a tag might need to temporarily recover the means to interact with it. In such a case the current owner of the tag should be able to transfer its authorisation rights over the tag to the previous owner.

The possible need for authorisation recovery in an RFID system was first raised in [24, 6]. However, it seems that no concrete protocol to address this possible requirement has been proposed. In the next section, we introduce RFID protocols for tag ownership transfer which support this requirement.

3. Related Work

We next introduce some RFID protocols that use a look-up table to identify a tag, thereby taking only $O(1)$ time, and hence have desirable scalability properties. We also outline shortcomings in these schemes.

Henrici and Müller [11] proposed a protocol for RFID tag identification (the HM scheme), in which the server only needs to perform $O(1)$ work to identify a tag. However, as described in [26, 5], the scheme allows a degree of tag tracking. In addition, if a tag is compromised, its previous identifiers can easily be computed, thereby allowing backward traceability [8].

Dimitriou [10] proposed an RFID authentication protocol (the D scheme), requiring $O(1)$ work for a server to authenticate a tag. However, a tag identifier might remain the same between valid sessions because, if an authentication

session is unsuccessful, a tag does not update its identifier. Tag tracking is thus partially possible [10], as in the HM scheme.

The RFID authentication protocol of Lim and Kwon [12] (the LK scheme) requires a server to maintain a precomputed table of tag information, used to authenticate tags. The scheme provides a range of security properties, covering backward and forward traceability and weak attacks. However, it does not provide location privacy, as described in [27]. Moreover, the scheme may involve significant on-line computations for tag authentication in a successful session [12], although it only requires $O(1)$ work for tag identification.

Tsudik [28] presented an RFID identification protocol (the T1 scheme) that provides only a basic level of tag identification using time-stamps, and proposed two further schemes (the T2 and T3 schemes) also providing tag authentication. The schemes use monotonically increasing time-stamps for tracking-resistant tag authentication. A server maintains a periodically updated hash table in which each row corresponds to a tag.

The T1 scheme only needs $O(1)$ operations to identify a tag, because a hash table is used for all look-ups. However, the scheme merely identifies a tag, and does not provide tag authentication. Additionally, the scheme is susceptible to a trivial DoS attack in which an attacker can easily incapacitate a tag by feeding it an inaccurate future time-stamp value [13]. Moreover, the scheme makes the important assumption that a given tag is never identified (interrogated) more than once within any time interval [13].

The T2 scheme adds tag authentication to T1 using a challenge-response method. This scheme also takes a constant time to identify and authenticate a tag because of its use of a look-up table. However, if a tag has been previously desynchronised by an attacker, the server must perform $O(n)$ operations to authenticate the tag. The T2 scheme is susceptible to DoS attacks, like the T1 scheme [13].

The T3 scheme mitigates the DoS vulnerability of T1 and T2 by using a hash-chain to generate a so-called epoch token, allowing a tag to ascertain that a time-stamp is not too far into the future. The server only needs to perform $O(1)$ operations to identify and authenticate a tag, if the tag reply is valid. If not, the server takes $O(n)$ time. Unfortunately, DoS attacks still remain a threat [13].

In addition, in T2 and T3, an adversary can potentially distinguish between synchronised and desynchronised tags by timing server responses, because a synchronised tag only requires a server to perform a fast table look-up, whereas a desynchronised tag requires it to perform an exhaustive search. Moreover, none of these schemes can resist backward traceability because they use a fixed key.

Burmester, de Medeiros and Motta [29] introduced an anonymous RFID authentication protocol (the BMM scheme) that supports constant key-lookup, using a pseudo-random function. However, the scheme has a location privacy weakness; if an authentication session fails, a tag re-uses the same pseudonym in the following session. Also, it does not provide backward untraceability because of the use of a fixed secret key [29].

4. The Revised SM protocol

Our novel protocols build on a revised version of the SM protocol [8]. We first outline the SM protocol and then describe the revised version. The XOR, concatenation, substitution, right circular shift and left circular shift operators are represented below by $\oplus$, $\|$, $\leftarrow$, $\gg$ and $\ll$, respectively.

Initially, a server S assigns a string s of l bits to each tag T and computes $t = h(s)$, where $h : \{0,1\}^* \rightarrow \{0,1\}^l$ is a hash function. T stores t, and S stores $(s, t, \hat{s}, \hat{t})$ for every tag it manages, where $\hat{s}$ and $\hat{t}$ are the most recent 'old' values of s and t.

The authentication protocol operates as follows (see also Figure 1).

1. S generates a random string r_1 of l bits and sends it to T.
2. T generates a random string r_2 of l bits as a temporary secret, and computes $M_1 = t \oplus r_2$ and $M_2 = f_t(r_1 \oplus r_2)$. T then sends M_1 and M_2 to S.
3. S searches its database for a value t for which $M_2 = f_t(r_1 \oplus M_1 \oplus t)$, where r_1 is the value sent by S in step 1. If no match is found, the session terminates. If a match is found, S has authenticated T. S computes $r_2 = M_1 \oplus t$ and $M_3 = s \oplus (r_2 \gg l/2)$, and sends M_3 to T. S then updates stored data for tag T from $(\hat{s}, \hat{t}, s, t)$ to $(s, t, (s \ll l/4) \oplus (t \gg l/4) \oplus r_1 \oplus r_2, h((s \ll l/4) \oplus (t \gg l/4) \oplus r_1 \oplus r_2))$.
4. T computes $s = M_3 \oplus (r_2 \gg l/2)$ and checks that $h(s) = t$. If the check succeeds, T has authenticated S, and sets $t \leftarrow h((s \ll l/4) \oplus (t \gg l/4) \oplus r_1 \oplus r_2)$. If the check fails, the session terminates.

In [30, 31], attacks are described on this protocol, which arise because the XOR operation is used to construct each of messages M_1, M_2 and M_3. Cai et al. [30] present a revised scheme in which the construction of M_2 uses concatenation instead of XOR, and M_3 is computed using a hash function as $h(r_2)$ instead of $(r_2 \gg l/2)$. We present here a further revision of the SM protocol. In our revised scheme, the construction of M_2 is the same as in the scheme introduced by Cai et al. [30], i.e. $M_2 = f_t(r_1 \| r_2)$, and M_3 is changed to $s \oplus f_t(r_2 \| r_1)$. Figure 1 summarises the revised SM protocol.

5. An RFID Pseudonym Protocol

We introduce here an RFID pseudonym protocol. This protocol is a revised version of the scheme presented in [1].

5.1. Main Features

The protocol has the following main features:

- To improve scalability, the protocol makes use of a precomputed look-up table for tag authentication, as in the schemes described in section 3. As a result, the server takes $O(1)$ work to identify and authenticate a tag, without needing a linear search.

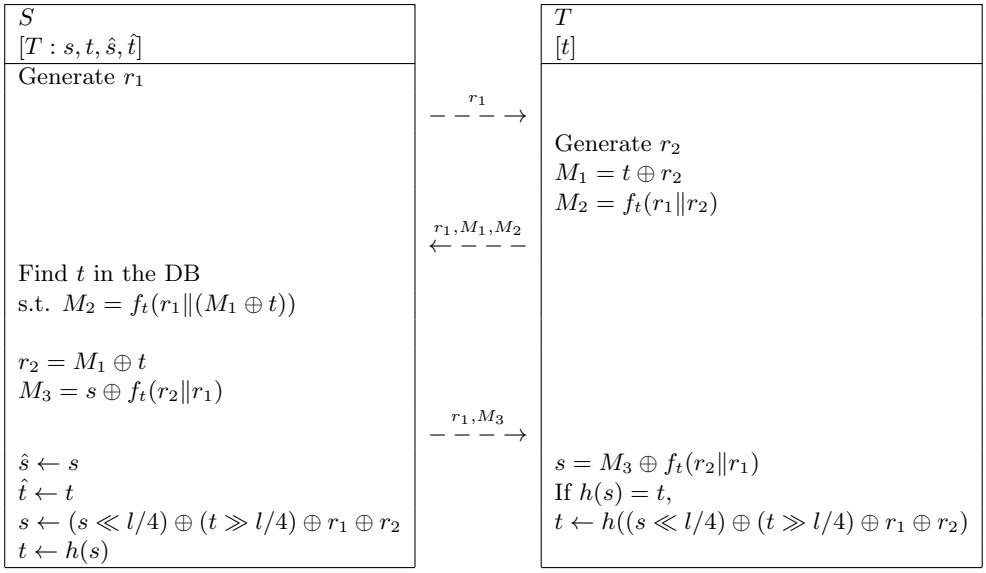

Figure 1: The revised SM protocol

- The look-up table contains a number of entries for each tag, one for each element of a tag-specific hash-chain. Elements from this hash-chain are used as tag identifiers (and as database keys to identify tags). A keyed hash function is used to generate each hash-chain, using a secret key shared by the tag and server. The hash-chain length, m, determines the number of tag identifiers that can be produced using any one key.
- The operation of the protocol (described in detail in section 5.3) can be divided into three cases, as follows (see also Table 1):
  1. Case 1: for each of the first $m-1$ queries of a tag, the protocol process only involves tag authentication and requires just two messages. To authenticate a tag, the server searches a look-up table, taking constant time.
  2. Case 2: on the mth query of a tag, the protocol updates the secrets shared by the server and tag, as well as providing tag authentication. This process requires an additional message. The server takes $O(1)$ work to authenticate a tag, as in case 1.
  3. Case 3: if a tag is queried more than m times, which should not normally happen, then a revised version of the SM protocol is performed; this requires the server to perform a linear search with complexity $O(n)$.
- For server authentication (in cases 2 and 3), for each tag the server holds a secret s that only it knows, as in the schemes presented in [12, 8].

Table 1: Operation of the protocol

Query number	$1, \cdots, (m-1)$	m	$(m+1), \cdots$
Operation	Case 1	Case 2	Case 3
State	Regular state		Irregular state

- In normal operation (cases 1 and 2), a tag does not need to generate pseudo-random numbers; however, in case 3, a pseudo-random number is needed to prevent tag tracking.

5.2. Initialisation

The server S chooses values for l, l_r and l_m and functions e, f, g and h as follows.

- l is the bit-length of a tag identifier and a shared key. It should be large enough to ensure that an l-bit key is a strong cryptographic key for the keyed hash functions, and in particular that an exhaustive search to find an l-bit tag identifier is computationally infeasible.
- l_r ($\leq l$) is the bit-length of a random string. It should be large enough to ensure that an exhaustive search to find an l_r-bit value is computationally infeasible.
- l_m is the bit-length of an integer m, that defines the length of the hash-chain used for tag identifiers.
- $e : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^l$, $f : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^l$ and $g : \{0,1\}^* \times \{0,1\}^l \rightarrow \{0,1\}^{2l+l_m}$ are keyed hash functions.
- h is a hash function, $h : \{0,1\}^* \rightarrow \{0,1\}^l$.
- e, f, g and h should be one-way, collision-resistant, and suitable for implementation in a low-cost tag. (A variety of work on developing such hash functions is ongoing; see, for example, [32].)

The server S builds a look-up table which is used for tag identification. The table definition process involves the following steps for each tag T managed by S.

- S chooses a random l-bit string s, and computes the l-bit key $k = h(s)$, where s is used for server authentication and k is used as input to the keyed hash functions e, f and g.
- S chooses a random l-bit string x_0, and computes the hash-chain values $x_i = e_k(x_{i-1})$ for $1 \leq i \leq m$, where the values x_i are used as one-time tag identifiers and m is the length of the hash-chain.

- S stores s, k and the identifiers $x_0, x_1, \cdots, x_m$ as the entries for T in its look-up table.

Each tag T stores k, x and a counter c, where x is initially set to x_0 and functions as T's identifier, and c is set to m.

5.3. Authentication and Secret Update

The novel protocol has three different stages, in line with the cases described in section 5.1: tag authentication, secret update (I) and secret update (II). The stages are as follows (see also Figure 2).

Case 1: Tag Authentication
S generates a random l_r-bit string r, and sends r to T.

1. When T receives r, it checks its counter c. If $c \neq 0$, then the following steps are performed.
   (a) T computes $M_T = f_k(r\|x)$ and updates its identifier x to $e_k(x)$ and its counter c to $c-1$. T sends r, M_T and x back to S. If the counter c is equal to 0, T waits for a server response, keeping r and M_T in short term memory.
   (b) When S receives M_T and x, it performs the following steps.
      i. S searches its look-up table for a value x_i equal to the received value of x. If such a value is found, S identifies T. Otherwise, the session terminates.
      ii. S checks that $f_k(r\|x_{i-1})$ equals the received value of M_T, where k is the key belonging to the identified tag T. If this verification succeeds, then S authenticates T. Otherwise, the session terminates.
      iii. If $x \neq x_m$, then the authentication session terminates successfully.

      Case 2: Secret Update (I)
      iv. If $x = x_m$, then S performs the following steps to update the secrets for T.
         A. S chooses a random l-bit string s' and an integer m', and computes a key $k' = h(s')$ and a sequence of m' identifiers $x'_i = e_{k'}(x'_{i-1})$ for $1 \leq i \leq m'$, where x'_0 is set to x. (These values can be precomputed.)
         B. S computes $M_S = g_k(r\|M_T) \oplus (s\|k'\|m')$, and sends r and M_S to T.
         C. S updates the set of stored values for T from ($\hat{s}$, $\hat{k}$, s, k, x_0, $x_1, x_2, \cdots, x_m$) to (s, k, s', k', x, $x'_1, x'_2, \cdots, x'_{m'}$), where $\hat{s}$ and $\hat{k}$ are the most recent previous values of s and k, respectively.

(c) When T receives r and M_S, it computes $(s\|k'\|m') = M_S \oplus g_k(r\|M_T)$. If $h(s)$ is equal to k, T authenticates S and updates k and c to k' and m', respectively. (The secret update session then terminates successfully.) Otherwise, the session terminates.

Case 3: Secret Update (II)

2. When T receives r, if T's counter c is equal to 0, then the following steps are performed. (This irregular case arises if T did not update its shared secrets correctly in the previous session, that is, if the secret update (I) step fails.)
   (a) T generates a random number r_T as a session secret, and computes $M_1 = f_k(r\|r_T)$ and $M_2 = r_T \oplus x$. T sends r, M_1 and M_2 back to S with a request for an update of the shared secrets. T waits for a server response, keeping r, r_T and M_1 in short term memory.
   (b) When S receives M_1 and M_2, the following steps are performed.
      i. S searches its look-up table for a value $x = x_m$ or $x = x_0$ for which $M_1 = f_k(r\|(M_2 \oplus x))$. If such a value is found, S authenticates T. Otherwise, the session terminates.
      ii. If $x = x_m$, S performs the following steps. (This case arises when, although T sent $x = x_m$ to S in the previous session, S did not receive it correctly. Thus, neither S nor T have updated their shared secrets.)
         A. S chooses a random l-bit string s' and an integer m', and computes a key $k' = h(s')$ and a sequence of m' identifiers $x'_i = e_{k'}(x'_{i-1})$ for $1 \leq i \leq m'$, where x'_0 is set to x. (These values can be precomputed.)
         B. S computes $r_T = M_2 \oplus x$ and $M_S = g_k(r\|r_T) \oplus (s\|k'\|m')$, and sends r and M_S to T.
         C. S updates the set of stored values for T from ($\hat{s}$, $\hat{k}$, s, k, x_0, $x_1, x_2, \cdots, x_m$) to (s, k, s', k', x, $x'_1, x'_2, \cdots, x'_{m'}$).
      iii. If $x = x_0$, S computes $r_T = M_2 \oplus x$ and $M_S = g_{\hat{k}}(r\|r_T) \oplus (\hat{s}\|k\|m)$ and sends r and M_S to T. (This case arises if M_S did not reach T correctly in the previous session, and thus T did not update its secrets, although S did. That is, this step resynchronises S and T.)
   (c) When T receives r and M_S, it computes $(s\|k'\|m') = M_S \oplus g_k(r\|r_T)$. If $h(s)$ is equal to k, T authenticates S and updates k and c to k' and m', respectively. (The secret update session then terminates successfully.) Otherwise, the session terminates.

6. Tag Delegation and Ownership Transfer

As we now describe, the protocol described in section 5 (referred to here as $\mathbf{P}_1$) can also be used to provide tag delegation and ownership transfer in an effective way.

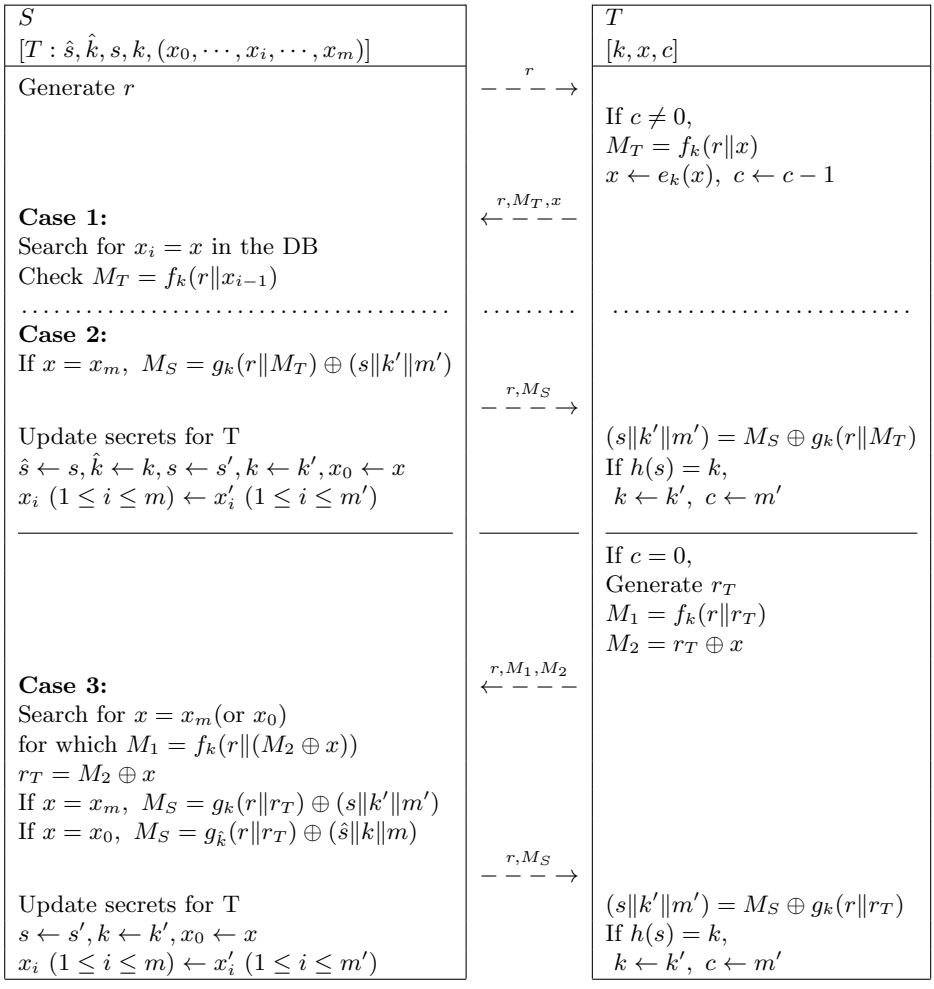

Figure 2: RFID authentication and secret update

6.1. Tag Delegation

Tag delegation enables a server to delegate the right to identify and authenticate a tag to a specified entity for a limited time period [12, 23]. Such a procedure could be used to reduce the computational load on a server.

In $\mathbf{P}_1$, tag delegation is straightforward. When S wants to delegate T to an entity, it transfers the secret k and the identifiers $x_0, x_1, \cdots, x_m$ for T to the entity via a secure channel. As a result, the entity can authenticate T a maximum of m times. However, the entity receiving the delegation right cannot update the tag secrets, as it does not know s.

Multiple delegations of a tag T are also possible. If S transfers the secret k and the identifiers $x_0, x_1, \cdots, x_m$ for T to multiple entities, then these entities can all authenticate T during the same limited period, that is, until $x = x_m$ is reached.

If the delegated tag T is queried m times, then S will need to update T's secret and identifiers and, if necessary, S can now delegate the right to query the tag again. Note that it is always necessary for S to update the tag secret and identifiers, since, as noted above, only S knows s.

6.2. Tag Ownership Transfer

Unlike delegation, tag ownership transfer means that the tag owner transfers all rights over the tag to a new owner [12, 23]. In order to achieve ownership transfer of a tag T using protocol $\mathbf{P}_1$, S must transfer the secrets s and k and the identifier x for T, along with any other necessary information, to the new owner via a secure channel. This transfer should only take place after the old owner has updated the secrets and identifiers for T, in order to protect the privacy of previously conducted transactions against possible tracking by the new owner. The server of the new owner should also update the tag secrets after receiving them from the old owner, in order to protect the privacy of future transactions against possible tracking by the old owner. This latter update needs to take place in an environment where there is no possibility of eavesdropping by the old owner. Once this is complete, only the server of the new owner will be able to authenticate T and update the secrets for T.

We introduce a protocol to update tag secrets for secure tag ownership transfer, as well as a protocol to provide authorisation recovery, in line with the requirements identified in section 2.5.2. These protocols are revised versions of the schemes described in a unpublished presentation given at the RFIDSec 08 workshop [33].

6.2.1. Secret Update Protocol

When tag ownership is to be transferred, a new owner can perform $\mathbf{P}_1$ to update the tag secrets. However, we propose using a novel secret update protocol (referred to as $\mathbf{P}_2$) to improve performance, in which the cryptographic function computations of a tag and the messages exchanged between a server and a tag are less than in $\mathbf{P}_1$. Such efficiency gains are possible because the server knows the identity of the newly acquired tag.

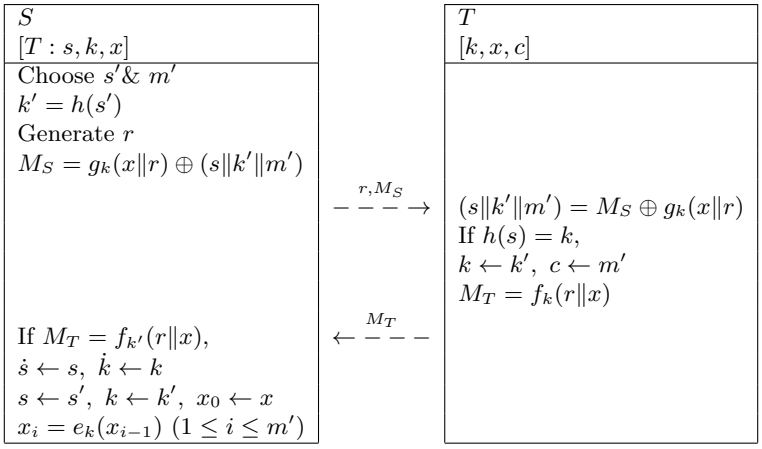

Figure 3: The secret update protocol

Prior to running $\mathbf{P}_2$, we assume that the server S of the new owner has secrets (s, k, x) for a tag T, obtained as a result of the tag information transfer described above; we also suppose that T has secrets (k, x, c). The secret update protocol $\mathbf{P}_2$ involves the following steps.

1. S chooses a new secret s' of l bits for T and an integer m', and computes $k' = h(s')$. S generates a random string r, computes $M_S = g_k(x\|r) \oplus (s\|k'\|m')$, and then sends r and M_S to T.
2. When T receives r and M_S from S, it performs the following steps:
   (a) T computes $(s\|k'\|m') = M_S \oplus g_k(x\|r)$.
   (b) If $h(s) = k$, T has authenticated S as an authorised server. Otherwise, the session terminates.
   (c) T updates its secret k to k' and its counter c to m'.
   (d) T computes $M_T = f_k(r\|x)$ using the new secret k, and sends M_T to S.
3. S checks that M_T is equal to $f_{k'}(r\|x)$. If the validation succeeds, S now knows that T has received the new secret k', and updates secrets s and k for T to s' and k', respectively. S computes the hash-chain values, $x_i = e_k(x_{i-1})$ for $1 \le i \le m'$, where x_0 is set to x. Otherwise, S goes to step 1, and starts a new session.

If $\mathbf{P}_2$ completes successfully (and the old owner does not eavesdrop on the messages), S and T share new secrets known only to them, and the old owner is no longer able to identify or trace T. Both the old and new owners can also keep the pair of the tag secrets provided by the old owner, denoted by $(\dot{s}, \dot{k})$, for use in the event that the old owner needs to identify T again. $\mathbf{P}_2$ is summarised in Figure 3.

6.2.2. Authorisation Recovery Protocol

As discussed above, the previous owner of a tag may need to temporarily interact with the tag again. The following authentication protocol (referred to as $\mathbf{P}_3$) enables this to occur.

$\mathbf{P}_3$ allows the server S to make T change its secret back to the value it had when S took ownership of T from the old owner. Prior to running $\mathbf{P}_3$, we assume that S stores the following information for tag T: the current secrets (s, k), the most recent secrets $(\hat{s}, \hat{k})$, and the old secrets also known to the old owner $(\dot{s}, \dot{k})$; we also suppose that tag T has secrets (k, x, c).

$\mathbf{P}_3$ is the same as $\mathbf{P}_2$ except that k' and m' are set equal to $\dot{k}$ and 0, respectively, without the step generating new secrets s' and k'. After successful execution of $\mathbf{P}_3$, T stores $\dot{k}$ as its secret. As a result, the old owner can identify T again by executing the SM protocol.

The current owner can recover authorisation rights over T from the old owner by executing $\mathbf{P}_2$.

7. Analysis

7.1. Privacy and Security

$\mathbf{P}_1$ involves performing a tag authentication (TA) process to authenticate a tag. When a tag is queried for the mth time, the server and tag update their shared secrets using the secret update (I) (SU_1) process. If SU_1 does not complete successfully, in the following session the secret update (II) (SU_2) process is performed. SU_1 and SU_2 make use of a key transfer protocol and involve mutual authentication.

Note that both TA (case 1) and SU_1 (case 2) are 'normal' cases of the protocol, but SU_2 (case 3) will only occur if the protocol fails to operate as it should. This case arises if a message transfer in SU_1 fails.

The security of $\mathbf{P}_1$ relies on the tag secrets k and s and the hash functions e, f, g and h. Under the assumption that the l-bit key k is a strong cryptographic key for e, f and g, an exhaustive search to find the l-bit values s and x is computationally infeasible. Also, given that hash functions e, f, g and h are one-way and collision-resistant, as stated in section 5, $\mathbf{P}_1$ has the following privacy and security properties.

- Tag Information Privacy (P1): we assume in section 2 that the server database is secure. Thus only the server that has the secrets related to a tag can identify the tag and access the tag information.
- Tag Location Privacy (P2): a tag reply (x, M_T) is anonymous to an eavesdropper that does not know k, because x is updated to $e_k(x)$ in every query and M_T depends on r and x. A tag reply (M_1, M_2) in SU_2 is also anonymous to an eavesdropper, because M_1 and M_2 are computed using the key k and a session secret r_T. As a result, an adversary cannot track the location of a tag simply by eavesdropping on tag messages.

The protocol resists the following attacks feasible for a WA.

- Tag Impersonation (W1): to impersonate a tag, a WA needs to compute x and M_T (or M_1 and M_2). However, a WA cannot compute them without knowing k.
- Replay Attack (W2): a WA cannot reuse messages used in previous sessions because each response is a cryptographic function of a fresh random number. More specifically, M_T and M_S in TA and SU_1 depend on r, and M_1, M_2 and M_S in SU_2 depend on r and r_T.
- MitM Attack (W3): a WA cannot interfere with the exchanged messages by inserting or modifying messages, because of the use of the secrets k and s and a random number r.
- DoS Attack (W4): if the second or third message in SU_1 is blocked, SU_2 will be performed in the following session. If the third message M_S in SU_2 is blocked, the server and tag will become desynchronised, because the server will update the shared secrets but the tag will not. However, in the next session, the server will detect such an event, because the tag will send as identifier the value x_0 in the server's look-up table. The server can thus recover synchronisation with the tag.

We next consider the degree to which $\mathbf{P}_1$ can resist the security threats posed by an SA, identified in section 2.

- Backward Traceability (S1): one significant feature of the protocol is that, when $x = x_i$ in TA, M_T is computed as a function of x_{i-1}. As a result, it is difficult for an SA to trace transactions in previous sessions except for the immediately previous session in which x_i is included in the tag reply. An SA could intercept a tag identifier from a previous transaction, and compute the compromised identifier x by iteratively applying keyed hash function e to the previous identifier. However, the previous transactions were anonymous to the attacker at that time. Thus, in practice, tracing past transactions will not be simple. Obviously, if tag past transactions were computed using keys different from the compromised key k, it will be infeasible for an SA to trace them, because the previous keys will have no relation to the key k.
- Forward Traceability (S2): an SA can trace future transactions in which the compromised key is used. However, once the server and the compromised tag update their shared secrets, and assuming that the SA does not intercept the values of either r or M_S, it will not be able to compute the updated secrets and thus will no longer be able to trace tag transactions. Therefore, a server should immediately replace the tag secrets if it suspects that a tag may have been compromised.

- Server Impersonation (S3): an SA could try to update the secrets of a target tag by impersonating a legitimate server. If such an attack was possible, then the legitimate server would no longer be able to identify the tag, whereas the attacker would. One advantage of the protocol is that such a server impersonation attack is not straightforward. The reason for this is that an SA cannot compute M_S just by compromising a tag, because s is known only by the server. An SA must perform a more sophisticated attack in which it intercepts M_S in order to learn s.

$\mathbf{P}_2$ and $\mathbf{P}_3$ are mutual authentication protocols. In these schemes, when tag T receives r and M_S from server S, T authenticates S by obtaining s from the messages and checking that $h(s) = t$. This works because s is a secret for T known only by S. S authenticates T by checking that the received M_T is correct, since it is computed using a shared secret k which is known only to T and S. S can also confirm that T has the same new secret k as S.

The schemes protect against tag information leakage because T's responses are a function of its secret k, and thus only the server that knows the secret is able to identify T and access the tag information. The schemes protect against tag location tracking because T's responses are anonymous, since they are a hash of r and x, and are independent of one another.

The messages exchanged between the server and tag are computed using a random string r, secrets k and s, and keyed hash functions f and g. Thus, the protocols can resist replay attacks and man-in-the-middle attacks. To impersonate a tag, an attacker must be able to compute a valid response M_T. However, it is difficult to compute such a message without knowledge of the secret k, because an attacker cannot learn the updated key k' from the message M_S sent by S and thus cannot compute M_T since it is a function of k'.

Denial-of-service attacks on $\mathbf{P}_2$ and $\mathbf{P}_3$ are not practical. S knows the identity of a tag when it starts a session with the tag, and the purpose of these schemes is to update the tag secrets. Thus, if S does not receive a tag reply to its query, it can immediately detect the error and fix it. Suppose that message M_T does not reach S in $\mathbf{P}_2$ (or $\mathbf{P}_3$). Then T will update its identifier, but S will not. However, S knows the updated value of k, and can use it to recover synchronisation with T.

Suppose that a tag T is compromised after $\mathbf{P}_2$ has been performed. The protocols do not enable backward traceability, because, in $\mathbf{P}_2$, T's new secret k' does not have any relationship to previous keys, and then, in $\mathbf{P}_1$, it is updated using a non-invertible hash function h. The schemes also resist forward traceability if an adversary does not obtain the values of either r or M_S exchanged between the server and tag in $\mathbf{P}_1$. Even if an adversary has compromised T, it cannot impersonate a legitimate server in $\mathbf{P}_2$ or $\mathbf{P}_3$ without additional information. This is because the server's message M_S is a function of the secret s known only to the server, and the adversary cannot obtain the value, even if it compromises T. To succeed in such an attack, the adversary must first perform other attacks to obtain s.

Table 2 summarises the privacy and security properties of the protocols $\mathbf{P}_1$,

Table 2: Privacy and security properties

		P1	P2	W1	W2	W3	W4	S1	S2	S3
HM		√	·	·	·	·	√	·	·	·
D		√	·	√	√	√	·	√	·	·
LK		√	·	√	√	√	√	√	*	*
T1		√	√	√	√	√	·	·	·	·
T2		√	√	√	√	√	·	·	·	·
T3		√	√	√	√	√	·	·	·	·
BMM		√	·	√	√	√	√	·	·	·
$\mathbf{P}_1$	TA	√	√	√	√	√	√	*	*	*
	SU_1	√	√	√	√	√	√	*	*	*
	SU_2	√	√	√	√	√	√	√	*	*
$\mathbf{P}_2$ & $\mathbf{P}_3$		√	√	√	√	√	√	√	*	*

√ : resists such an attack

∗ : partially resists such an attack, under certain assumptions

· : does not protect against such an attack

$\mathbf{P}_2$ and $\mathbf{P}_3$, and compares the protocols to the prior art introduced in section 3.

7.2. Performance

$\mathbf{P}_1$ has the following performance characteristics.

- Scalability: a server uses a look-up table for tag identification. As a result, a server can match a received anonymous identifier to a tag using its look-up table in $O(1)$ time, without needing a linear search. The protocol is scalable in the sense that a server only takes constant time to authenticate a tag, and tag delegation is straightforward, as stated in section 6. However, if a tag is queried more than m times without updating the tag secrets (case 3), the tag will reply with M_1 and M_2, and in this case the server needs to perform a linear search to authenticate the tag.

- Computation: in normal operation, i.e. when using TA and SU_1, a tag does not need to generate any pseudo-random numbers. However, in SU_2, a tag needs to generate a pseudo-random number in order to prevent it being traced. A tag needs to perform two hash function computations (which are significantly more computationally complex than arithmetic and logical operations) in the most common case (TA), four hash function computations in SU_1, and three hash function computations in SU_2. A server performs only one hash function computation in TA. In SU_1 and SU_2, a server must perform m' hash function computations in order to generate a new secret and new identifiers for a tag; fortunately these values can be precomputed.

- Communication: TA involves only two messages. SU_1 and SU_2 require one additional message.

Table 3: Performance properties

		C1	C2	C3	C4
HM		I, a, a'	3HF	0	3
D		I	4HF	1	3
LK		s, w, c	4PRF	1	3
T1		k, t, t_m	1HF	0/1	2
T2		k, t, t_m	2HF	1/2	2
T3		k, t, t_m	(ν+2)HF	1/3	2
BMM		k, r, q, b, c	1/2PRF	0	3
$\mathbf{P}_1$	TA	k, x, c	2HF	0	2
	SU_1		4HF	0	3
	SU_2		3HF	1	3
$\mathbf{P}_2$ & $\mathbf{P}_3$		k, x, c	3HF	0	2

C1 : The type of secrets stored in a tag

C2 : The type and number of cryptographic function computations required in a tag

C3 : The number of pseudo-random numbers required in a tag

C4 : The number of exchanged messages

- Storage Capacity: a tag needs a long term memory of $2l + l_m$ bits to store k, x and c.

$\mathbf{P}_2$ and $\mathbf{P}_3$ are efficient in terms of computation and communication, because a tag does not need to generate any pseudo-random numbers, and only two messages need to be exchanged to provide mutual authentication between the server and tag.

$\mathbf{P}_2$ and $\mathbf{P}_3$ have modest computational requirements. The only cryptographic functions required by $\mathbf{P}_2$ or $\mathbf{P}_3$ are at most three hash function computations in the tag and the server. In $\mathbf{P}_2$, both T and S need to compute each of h, f and g once. In $\mathbf{P}_3$, T needs to compute each of h, f and g once, and S needs to compute f and g once.

Therefore, in order to update a tag's secrets after transfer of ownership, performing $\mathbf{P}_2$ is more economical than performing $\mathbf{P}_1$ again.

The performance of the protocols $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ is compared to the prior art in Table 3. The comparison shows that the performance of the proposed protocol compares favourably with existing schemes. In Table 3, HF is a hash function computation, PRF is a pseudo-random function computation, I is a tag identifier, a is a transaction number, a' is the last successful transaction number, s is a tag secret, w is a server validator, c is a counter, k is a key, k is a time-stamp, t_m is the highest possible time-stamp, ν is the number of successive iterations of a hash function, r is a one-time pseudonym, q is a seed, b is a boolean variable mode, and / denotes or.

7.3. Tag Ownership Transfer

The protocols proposed in section 6.2 meet the three requirements for tag ownership transfer identified in section 2.5.2.

First, $\mathbf{P}_2$ is designed to protect the privacy of the new owner from the old owner of a tag T. That is, future interactions between the new owner and T are secure against tracing by the old owner. When ownership of T is transferred to the new owner, the new owner and T establish new secrets using $\mathbf{P}_2$. As a result, the old owner is no longer able to read T.

Next, the protocols also protect the privacy of the old owner from the new owner of T. The old owner should update secrets s and k for T, before transferring the updated secrets to the new owner. As a result, the new owner cannot trace previous transactions between the old owner and the tag since it only knows the updated secrets.

Finally, $\mathbf{P}_3$ provides authorisation recovery, the third requirement described in section 2.5.2. $\mathbf{P}_3$ causes T to change its secret k to $\dot{k}$, which the new owner received from the old owner when ownership of T was transferred. As a result, the old owner recovers authorisation rights for T, and thus can read T again and look up the tag information.

8. Conclusions

We have identified desirable privacy, security and performance properties for RFID authentication protocols, and have examined three requirements for secure and privacy-preserving tag ownership transfer: new owner privacy, old owner privacy, and authorisation recovery. We have reviewed previously proposed scalable RFID identification and authentication protocols which take only constant time to identify a tag using a look-up table. All these schemes have significant security or performance drawbacks.

One of the main contributions of this paper is to propose a scalable RFID pseudonym protocol ($\mathbf{P}_1$) that meets the identified requirements. The protocol has two features supporting scalability; a server takes only $O(1)$ work for tag authentication, and tag delegation is straightforward.

In some RFID applications it is necessary to allow for transfer of tag ownership. We have proposed novel RFID authentication protocols for tag ownership transfer that meet the three identified requirements: a secret update protocol ($\mathbf{P}_2$) and an authorisation recovery protocol ($\mathbf{P}_3$).

We have also analysed and compared $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ to the prior art. The schemes satisfy the identified privacy and security requirements and have desirable performance characteristics; a tag does not need to generate any pseudo-random numbers and has reasonable storage and cryptographic function computation overheads.

[1] B. Song, C. J. Mitchell, Scalable RFID Pseudonym Protocol, in: 3rd International Conference on Network & System Security — NSS 2009, IEEE Computer Society, Gold Coast, Queensland, Australia, 2009, pp. 216–224.

[2] G. Avoine, Cryptography in radio frequency identification and fair exchange protocols, Ph.D. thesis, Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland (December 2005).

[3] A. Juels, RFID security and privacy: A research survey, IEEE Journal on Selected Areas in Communications 24 (2006) 381–394.

[4] S. Weis, Security and privacy in radio-frequency identification devices, Master's thesis, Massachusetts Institute of Technology (MIT), Massachusetts, USA (May 2003).

[5] H. Chien, C. Chen, Mutual authentication protocol for RFID conforming to EPC class 1 generation 2 standards, Computer Standards & Interfaces 29 (2) (2007) 254–259.

[6] S. Fouladgar, H. Afifi, A simple privacy protecting scheme enabling delegation and ownership transfer for RFID tags, Journal of Communications 2 (6) (2007) 6–13.

[7] M. Ohkubo, K. Suzki, S. Kinoshita, Cryptographic approach to "privacy-friendly" tags, in: RFID Privacy Workshop, MIT, MA, USA, 2003, `http://www.rfidprivacy.us/2003/agenda.php`.

[8] B. Song, C. J. Mitchell, RFID authentication protocol for low-cost tags, in: V. D. Gligor, J. Hubaux, R. Poovendran (Eds.), ACM Conference on Wireless Network Security — WiSec '08, ACM Press, Alexandria, Virginia, USA, 2008, pp. 140–147.

[9] A. Bondi, Characteristics of scalability and their impact on performance, in: the 2nd International Workshop on Software and Performance — WOSP 2000, ACM Press, Ottawa, Ontario, Canada, 2000, pp. 195–203.

[10] T. Dimitriou, A lightweight RFID protocol to protect against traceability and cloning attacks, in: Conference on Security and Privacy for Emerging Areas in Communication Networks — SecureComm 2005, IEEE, Athens, Greece, 2005, pp. 59–66.

[11] A. Henrici, P. Müller, Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers, in: R. Sandhu, R. Thomas (Eds.), International Workshop on Pervasive Computing and Communication Security — PerSec 2004, IEEE Computer Society, Orlando, Florida, USA, 2004, pp. 149–153.

[12] C. Lim, T. Kwon, Strong and robust RFID authentication enabling perfect ownership transfer, in: P. Ning, S. Qing, N. Li (Eds.), Conference on Information and Communications Security — ICICS '06, Vol. 4307 of Lecture Notes in Computer Science, Springer-Verlag, Raleigh, North Carolina, USA, 2006, pp. 1–20.

[13] G. Tsudik, A family of dunces: Trivial RFID identification and authentication protocols, in: N. Borisov, P. Golle (Eds.), Privacy Enhancing Technologies, 7th International Symposium — PET 2007, Vol. 4776 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Ottawa, Canada, 2007, pp. 45–61.

[14] P. Najera, J. Lopez, RFID: Technological Issues and Privacy Concerns, in: A. Acquisti, S. Gritzalis, C. Lambrinoudakis, S. di Vimercati (Eds.), Digital Privacy: Theory, Technologies and Practices, Taylor & Francis Group, 2008, Ch. 14, pp. 285–306.

[15] S. Weis, S. Sarma, R. Rivest, D. Engels, Security and privacy aspects of low-cost radio frequency identification systems, in: D. Hutter, G. Müller, W. Stephan, M. Ullmann (Eds.), International Conference on Security in Pervasive Computing — SPC 2003, Vol. 2802 of Lecture Notes in Computer Science, Springer-Verlag, Boppard, Germany, 2003, pp. 201–212.

[16] G. Avoine, P. Oechslin, A scalable and provably secure hash based RFID protocol, in: International Workshop on Pervasive Computing and Communication Security — PerSec 2005, IEEE Computer Society Press, Kauai Island, Hawaii, USA, 2005, pp. 110–114.

[17] B. Song, Server Impersonation Attacks on RFID Protocols, in: Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies — UBICOMM 08, IEEE Computer Society, Valencia, Spain, 2008, pp. 50–55.

[18] D. N. Duc, J. Park, H. Lee, K. Kim, Enhancing security of EPCglobal gen-2 RFID tag against traceability and cloning, in: Symposium on Cryptography and Information Security — SCIS 2006, The Institute of Electronics, Information and Communication Engineers, Hiroshima, Japan, 2006.

[19] T. van Le, M. Burmester, B. de Medeiros, Universally composable and forward-secure RFID authentication and authenticated key exchange, in: R. Deng, P. Samarati (Eds.), ACM Symposium on information, Computer and Communications Security — ASIACCS '07, 2007, pp. 242–252.

[20] S. Karthikeyan, N. Nesterenko, RFID security without extensive cryptography, in: Workshop on Security of Ad Hoc and Sensor Networks — SASN '05, ACM Press, Alexandria, Virginia, USA, 2005, pp. 63–67.

[21] I. Vajda, L. Buttyán, Lightweight authentication protocols for low-cost RFID tags, in: Second Workshop on Security in Ubiquitous Computing — Ubicomp 2003, Seattle, WA, USA, 2003.

[22] Y. Zhang, P. Kitsos, Security in RFID and Sensor Networks, Auerbach Publications, 2009.

[23] D. Molnar, A. Soppera, D. Wagner, A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags, in: B. Preneel, S. Tavares (Eds.), Selected Areas in Cryptography — SAC 2005, Vol. 3897 of Lecture Notes in Computer Science, Springer-Verlag, Kingston, Canada, 2005, pp. 276–290.

[24] S. Fouladgar, H. Afifi, An efficient delegation and transfer of ownership protocol for RFID tags, in: First International EURASIP Workshop on RFID Technology, Vienna, Austria, 2007.

[25] K. Osaka, T. Takagi, K. Yamazaki, O. Takahashi, An efficient and secure RFID security method with ownership transfer, in: Y. Wang, Y. Cheung, H. Liu (Eds.), Computational Intelligence and Security — CIS 2006, Vol. 4456 of Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 778–787.

[26] G. Avoine, P. Oechslin, RFID traceability: A multilayer problem, in: A. Patrick, M. Yung (Eds.), Financial Cryptography — FC'05, Vol. 3570 of Lecture Notes in Computer Science, IFCA, Springer-Verlag, Roseau, The Commonwealth Of Dominica, 2005, pp. 125–140.

[27] K. Ouafi, R. C.-W. Phan, Traceable Privacy of Recent Provably-Secure RFID Protocols, in: S. Bellovin, R. Gennaro, A. Keromytis, M. Yung (Eds.), 6th International Conference on Applied Cryptography and Network Security — ACNS 2008, Vol. 5037 of Lecture Notes in Computer Science, Springer-Verlag, New York City, New York, USA, 2008, pp. 479–489.

[28] G. Tsudik, YA-TRAP: Yet another trivial RFID authentication protocol, in: Fourth IEEE Annual Conference on Pervasive Computing and Communications — PerCom 2006, IEEE Computer Society, Pisa, Italy, 2006, pp. 640–643.

[29] M. Burmester, B. de Medeiros, R. Motta, Anonymous RFID authentication supporting constant-cost key-lookup against active adversaries, Journal of Applied Cryptography 1 (2) (2008) 79–90.

[30] S. Cai, Y. Li, T. Li, R. Deng, Attacks and Improvements to an RFID Mutual Authentication Protocol and its Extensions, in: Second ACM Conference on Wireless Network Security — WiSec'09, ACM Press, Zurich, Switzerland, 2009, pp. 51–58.

[31] T. van Deursen, S. Radomirović, Attacks on RFID Protocols, Cryptology ePrint Archive, Report 2008/310 (July 2008).

[32] A. Shamir, SQUASH — A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags, in: K. Nyberg (Ed.), Fast Software Encryption: 15th International Workshop — FSE 2008, Revised Selected Papers, Vol. 5086 of Lecture Notes in Computer Science, Springer-Verlag, Lausanne, Switzerland, 2008, pp. 144–157.

[33] B. Song, RFID Tag Ownership Transfer, in: Workshop on RFID Security — RFIDSec 08, Budapest, Hungary, 2008.