Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

CRFID: An RFID system with a cloud database as a back-end server

Shuai-Min Chen [a], Mu-En Wu [b], Hung-Min Sun [a,*], King-Hang Wang [c]

[a] Department of Computer Science, National Tsing Hua University, Taiwan, ROC
[b] Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC
[c] Hong Kong Institute of Technology, Hong Kong

HIGHLIGHTS

- Proposing a secure and efficient privacy preserving RFID authentication protocol.
- Using a RFID back-end server with cloud database to reduce the search complexity as well as data consistency.
- Withstanding desynchronizing attacks and tracking attacks.
- Providing scalability with $O(\log N)$ search complexity.

ARTICLE INFO

Article history:
Received 5 January 2013
Received in revised form
1 April 2013
Accepted 17 May 2013
Available online xxxx

Keywords:
Cryptography
Cloud computing
Remote accessibility
RFID
Privacy
Security
Desynchronizing attack
Tracking attack

ABSTRACT

Radio-frequency identification (RFID) systems can benefit from cloud databases since information on thousands of tags is queried at the same time. If all RFID readers in a system query a cloud database, data consistency can easily be maintained by cloud computing. Privacy-preserving authentication (PPA) has been proposed to protect RFID security. The time complexity for searching a cloud database in an RFID system is $O(N)$, which is obviously inefficient. Fortunately, PPA uses tree structures to manage tags, which can reduce the complexity from a linear search to a logarithmic search. Hence, tree-based PPA provides RFID scalability. However, in tree-based mechanisms, compromise of a tag may cause other tags in the system to be vulnerable to tracking attacks. Here we propose a secure and efficient privacy-preserving RFID authentication protocol that uses a cloud database as an RFID server. The proposed protocol not only withstands desynchronizing and tracking attacks, but also provides scalability with $O(\log N)$ search complexity.

1. Introduction

Radio-frequency identification (RFID) systems have attracted much attention in recent years. They were originally designed for military use, but are now gradually replacing optical barcode systems. The traditional barcode system was adopted because the cost of adding a barcode to an item is almost zero; however, the volume of barcode storage is limited. RFID systems have several advantages over the barcode system, such as cloud data management, parallel processing, shorter access time, longer-distance contactless sensing, and rewritable properties. Moreover, RFID systems can parallel scan hundreds of items in a few seconds, which is a dramatic improvement over barcodes, which must be scanned individually. Since there are thousands of tags maintained in a system, related information for these tags should be maintained in a central site, such as a cloud database. RFID systems have been adopted in many applications, such as supply chains [1], car/door locks, product sales, and e-passports [2], because of their convenience and efficiency. An RFID system has three components: tags (transponders) containing electronic circuits, a back-end database server to maintain tag-related data, and an RFID reader (transceiver) that scans the tag and queries the database via a wireless connection. The back-end database server can be accessed by all RFID readers and has parallel computing ability to serve all readers at the same time. An RFID reader (hereafter called a reader) obtains data stored in a tag by transmitting electronic waves to interrogate the tag. When a tag is activated, related information is returned. The back-end server, also known as a cloud database, is responsible for parallel computing, storing detailed information, and maintaining a look-up table of hash values or identity and key pairs. Thus, an important task of the back-end server is key management. The reader, back-end server, and tags are connected wirelessly via an insecure

* Corresponding author. Tel.: +886 35742968.
E-mail addresses: hmsun@cs.nthu.edu.tw, bensonbessie@gmail.com (H.-M. Sun).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.05.004

channel. Hence, information transmitted from the tag to the reader and from the reader to the server is publicly accessible and may be subject to eavesdropping. Therefore, if information stored in the tags is not protected via proper security measures, it may be subject to security or privacy risks that are nonexistent in the barcode system.

A simple way for a tag to securely transmit data for authentication is to send a meaningless message, that is, encrypted data or a hash value. When a valid reader obtains this meaningless value, it will retrieve the tag identity and other related information through the back-end server. Since an invalid reader cannot connect to the back-end server, the attacker cannot retrieve the tag information from the meaningless message. Nevertheless, because the tag responds with the same hash value every time, an attacker can successfully trace the tag once the meaningless message is received again.

There are many types of logical operations that can be executed to provide secrecy for RFID systems, such as modular addition, shift, and bitwise operations and pseudo-random number generation [3]. In addition, some studies have used symmetric encryption, one-way hash functions, and AES encryption. It is believed that the next generation of tags can use these cryptographic functions [4,5].

Two types of severe attack are commonly launched on RFID systems: tracking attacks and desynchronizing attacks [4].

Tracking attack: Owing to the contactless access, an RFID reader can freely scan a tag if the tag is not protected properly. If a tag always responds with the same data when it is queried, privacy problems can occur, such as when an attacker becomes aware of the existence of the tag. In other words, the tag is tracked by the attacker. For example, when a tag attached to a book borrowed from a library is tracked by an attacker, the borrower's privacy can be compromised. Another example is if a thief can trace the location of a valuable item stored in a public locker, then the thief might break the locker and steal that valuable item.

Desynchronizing attack: To prevent a tag from always replying with the same message and attracting tracking attacks, the tag and the back-end database are required to synchronize an authentication key and refresh it after a period of time. After refreshing the key, both the reader and the tag change and save the latest key for the next authentication process. In other words, the tag will reply with a different message after synchronizing. Hence, if an attacker interrupts the update process, the tag will be unable to refresh its key. Instead, the tag will still use the invalid authentication key next time. Obviously, the reader will forbid the authentication request next time since the key that tag sent is different on the reader side. The function of the tag will no longer exist. In a desynchronizing attack, the attacker attempts to interfere with the communication between a tag and a reader so that the keys stored in the database and the tag will be out-of-sync. As a result, the reader will no longer be able to read the tag correctly. This attack is severe in some applications. For example, when a tag attached to a valuable item in a shop is desynchronized with the back-end database, a thief can steal it without triggering the RFID security alarm system.

This study focuses on the important security issue mentioned above to design a secure and efficient RFID authentication protocol with scalability. A simple solution to protect RFID systems from tracking attacks has been proposed [4], but this requires $O(N)$ searches in the back-end database, where N is the number of tags, and is thus inefficient for large-scale RFID systems. Some studies have provided lower search complexity for protecting RFID systems [6–9]. Lu et al. proposed an ACTION protocol [10] that provides $O(\log N)$ search complexity in the back-end database. It also preserves tag privacy even if a large number of tags are compromised. However, we found that the ACTION protocol may be vulnerable to tracking attacks; an attacker can trace a tag without compromising any tags. In particular, the authors proposed a key-updating mechanism to refresh the key stored in the back-end server and the tag. However, we found that this mechanism is vulnerable to desynchronization attacks [11].

The remainder of the paper is organized as follows. Related work is reviewed in Section 2. In Section 3 we present the proposed scheme and a security analysis and detailed discussion. Section 6 concludes.

2. Related work

RFID technology has been the focus of research for many years and many issues have been addressed. One of the most important issues in the evolution of RFID is providing security and privacy. Some studies have focused on lightweight operations, such as exclusive-or, addition, and shift operations [12–14]. These mechanisms are designed mainly for passive tags. Most of the computation load is transferred to a cloud server since these servers are usually equipped with more powerful computational hardware. Therefore, this reduces the computation load for lightweight tags and shortens the overall authentication time. Nevertheless, most of these approaches have security shortcomings [15–17].

To protect the privacy of tags, privacy-preserving authentication (PPA) was proposed [18]. This is a straightforward way to maintain all the secret keys for tags in the cloud database. However, the cloud database has to store a large number of keys, so PPA methods are not scalable. To solve this scalability problem, many types of PPA mechanism have been proposed. For example, the tree-based mechanism generates a pseudo tree graph to map all the tags to nodes at different positions. It uses a key-sharing mechanism in which keys may be preassigned to tags at pseudo-positions in the tree. Owing to the characteristics of the tree-based structure, schemes based on a balanced tree provide a level of privacy with relatively lower search complexity. Different key storage mechanisms are applied so that each tag has a different key or shares part of its key with other tags. Consequently, tag-compromising attacks may occur. This is the most critical problem for tree-based mechanisms.

A few schemes based on balanced trees have been proposed to address the RFID privacy issue [6–9]. Although a balanced tree mechanism can reduce the search complexity from $O(N)$ to $O(\log N)$, some tags might share a proportion of the same keys. Therefore, information may be leaked once the attacker compromises a few tags. Avoine showed that for a balanced tree approach, an attacker can trace all the tags with probability close to 100% once 20 tags have been compromised [19].

In 2009, Lu et al. proposed a balanced tree ACTION protocol [10] that provides $O(\log N)$ search complexity in the back-end database. In addition, the authors claimed that it preserves tag privacy even if a large number of tags are compromised. However, we found that the ACTION protocol is vulnerable to tracking attacks and desynchronizing attacks [11]. An attacker can track a tag without compromising any tags.[1]

3. The proposed protocol

We developed a scalable, secure and efficient RFID authentication protocol, called CRFID, that overcomes the security weaknesses of existing approaches. The notation used is listed in Table 1.

The proposed scheme contains two phases: **Initialization** and **Read**. In **Initialization**, each tag $\mathcal{T}_i$ is assigned a unique key pair k_i and s_i. At the same time, the reader constructs a pseudo key tree. Note that the tag itself does not consist of any peripheral output

[1] Owing to space limitations, readers should refer to the ACTION protocol and the proposed attack directly.

Table 1
Notation used.

Symbol	Description
$\mathcal{R}$	RFID reader
$\mathcal{T}$	RFID tag
n_i	Nonce i
l_n	Length of n_i
$\mathcal{H}()$	One-way hash function
l_h	Length of output $\mathcal{H}()$
k_i	Path key for $\mathcal{T}$
$k_i[x]$	The xth subkey of the path key
l_k	Length of a subkey
s_i	Secret key for $\mathcal{T}$
δ	Branching factor for the key tree
T	Key tree stored in database
d	Depth of the key tree
c	Number of times that *TagJoin* runs
N	Number of tags in the system
$\mathcal{V}$	Victim list

```
1. $\mathcal{R} \to \mathcal{T}$   : Request, $n_1$; $n_1 \xleftarrow{R} \{1\}^n$
2. $\mathcal{T}$                   : $U = \{n_2, \mathcal{H}(n_1||n_2||k_i[0]), \mathcal{H}(n_1||n_2||k_i[1]),$
                                     $\cdots, \mathcal{H}(n_1||n_2||k_i[d-1]), \mathcal{H}(n_1||n_2||s_i)\}$;$n_2 \xleftarrow{R} \{1\}^n$
3. $\mathcal{T} \to \mathcal{R}$   : $U$
4. $\mathcal{R}$                   :$\{i, k_i, s_i\} \leftarrow Identify(U)$
5. $\mathcal{R}$                   :Add $\{k_i, s_i\}$ into trash list $\mathcal{L}$.
                                   :$\{k'_i, s'_i\} \leftarrow KeyGen(k_i, s_i, i)$
6. $\mathcal{R} \to \mathcal{T}$   :$\sigma = \{\mathcal{H}(n_1||n_2||k_i) \oplus k'_i, \mathcal{H}(n_1||n_2||s_i||k'_i)\}$
7. $\mathcal{T}$                   :$\{k'_i, s'_i\} \leftarrow NewVerify(\sigma, n_1, n_2)$
                                   : Terminate if not verified
                                   : Update $k_i \leftarrow k'_i, s_i \leftarrow s'_i$
8. $\mathcal{T} \to \mathcal{R}$   :$\alpha = \mathcal{H}(n_1||n_2||k'_i||k_i)$
9. $\mathcal{R}$                   :Verify $\alpha \overset{?}{=} \mathcal{H}(n_1||n_2||k'_i||k_i)$
                                   :If not verified:
                                     re-read the tag.
                                   :Otherwise:
                                     update $k_i \leftarrow k'_i, s_i \leftarrow s'_i$
                                     For all record $(k''_i, s''_i)$ in $\mathcal{L}$,
                                       $TagLeave(k''_i, s''_i)$
                                     Empty $\mathcal{L}$
```

Fig. 1. Read for the CRFID protocol.

to indicate failure of the authentication process. Therefore, anyone with access to a reader cannot ascertain if the protocol has been successfully executed. To prevent this serious RFID shortcoming, we designed an important communication process in Step 8, as shown in Fig. 1. In addition, the reader re-reads the tag if it does not receive a confirmation message from the tag (Step 9). Besides the **Initialization** and **Read** phases, the proposed scheme also considers the situation when a new tag joins a group and verification of key updating (*keyGen* and *NewVerify*; Algorithms 1 and 2, respectively). In the following subsections, **Initialization** and **Read** are illustrated in detail.

3.1. *Initialization*

Some parameters should be determined before using the system. These include the key tree depth d, the branching factor δ, and the bit length for each subkey l_k of the path key. The length of the path key is $d \times l_k$. Instead, δ and l_k are selected independently according to the system and the security concern. A recommended choice of parameters (d, δ, l_k) is $(4, 32, 60)$, as described in Section 5.1.

Suppose there are N RFID tags and *KeyGen* is executed for each tag with the inputs tag identity i and two random string values k_i and s_i. The algorithm for *KeyGen* is shown in Algorithm 1. The output values k'_i and s'_i are securely imprinted to tag i, for example by writing it inside a radio-shield metal box. After processing every tag, a key tree T is formed. Each leaf on T represents a small number of tags (probably only one).

Algorithm 1 *KeyGen*

```
1: INPUT: Tag ID $i$, Path Key $k_i$, Secret Key $s_i$. ($k_i$ and $s_i$ are random strings for Initialization .)
2: OUTPUT: New Path Key $k'_i$, Secret Key $s'_i$.
3: Internal Variable: Key Tree $T$ with depth $d$, victim list $\mathcal{V}$ .
4: Let $M$ be the root of $T$.
5: for $j \leftarrow 0, 1, \ldots, d-1$ do
6:     if $deg(M) = \delta$ then
7:         $r \leftarrow \{1 \cdots \delta\}$
8:         $M \leftarrow M_r$, where $M_r$ is the $r$th child of $M$.
9:     else
10:        Create node $M'$ and attach it to $M$.
11:        Define $label(M') \leftarrow R\{1\}^{l_k}$.
12:        $M \leftarrow M'$
13:    end if
14:    $k'_i[j] \leftarrow label(M)$
15: end for
16: if $label(M)$ is the same with one of list in $\mathcal{V}$ then
17:     do for loop.
18: end if
19: $s'_i \leftarrow \mathcal{H}(k_i||k'_i||s_i)$
20: return $k'_i$, $s'_i$
```

3.2. *Read*

When a reader $\mathcal{R}$ tries to communicate with a tag $\mathcal{T}$, it executes the protocol shown in Fig. 1 as follows:

(*Step* 1) $\mathcal{R}$ first picks a random number n_1 of length n and sends it to $\mathcal{T}$.

(*Step* 2) $\mathcal{T}$ generates a random number n_2 of length n via its built-in PRNG generator, and computes U using Eq. (1).

(*Step* 3) $\mathcal{T}$ replies with U to $\mathcal{R}$.

$$U = \{n_2, \mathcal{H}(n_1\|n_2\|k_i[0]), \mathcal{H}(n_1\|n_2\|k_i[1]), \ldots, \mathcal{H}(n_1\|n_2\|k_i[d-1]), \mathcal{H}(n_1\|n_2\|s_i)\}. \quad (1)$$

(*Step* 4) $\mathcal{R}$ identifies $\mathcal{T}$ using the algorithm *Identify*. Basically, starting from the root node, *Identify* iteratively examines nodes to see which child of the target node M has a tree label that satisfies Eq. (2).

$$\mathcal{H}(n_1\|n_2\|label\, M_r) = \mathcal{H}(n_1\|n_2\|k_i[j]). \quad (2)$$

Then it moves to next iteration by setting the target node as the child node M_r. At the end of the search, a leaf node on T that represents a small number of tags is revealed. $\mathcal{R}$ identifies the tag $\mathcal{T}$ and obtains the corresponding values for k_i and s_i. This is done by finding $\hat{s}_i$ for each tag on the leaf that matches $\mathcal{H}(n_1\|n_2\|s_i)$.

(*Step* 5) The value (k_i, s_i) is inserted into a trash list $\mathcal{L}$. $\mathcal{R}$ executes *keyGen*(k_i, s_i, i) to update the key tree T and obtains k'_i and s'_i. Note that the victim list $\mathcal{V}$ in *keyGen* is used by the administrator to manually store specious tag information, such as compromised tags. The purpose of $\mathcal{V}$ is to avoid generating a path key that is dangerous.

(*Step* 6) $\mathcal{R}$ sends $\sigma = \{\mathcal{H}(n_1\|n_2\|k_i) \oplus k'_i, \mathcal{H}(n_1\|n_2\|s_i \parallel k'_i)\}$ to $\mathcal{T}$.

(*Step* 7) $\mathcal{T}$ receives σ and executes *NewVerify*(σ, n_1, n_2) (Algorithm 2) to verify σ. If σ is authentic, the algorithm outputs the new key pair k'_i and s'_i. $\mathcal{T}$ then executes key updating whereby k_i is updated to k'_i and s_i is updated to s'_i; otherwise, $\mathcal{T}$ terminates the protocol.

(*Step* 8) $\mathcal{T}$ sends $\alpha = \mathcal{H}(n_1\|n_2\|k'_i \parallel k_i)$ to $\mathcal{R}$.

(*Step* 9) $\mathcal{R}$ verifies if $\sigma \overset{?}{=} \mathcal{H}(n_1\|n_2\|k'_i \parallel k_i)$. If σ is false, $\mathcal{R}$ re-reads the tag; otherwise, $\mathcal{R}$ updates k_i to k'_i and s_i to s'_i. In addition, for all entries (k''_i, s''_i) in $\mathcal{L}$, $\mathcal{R}$ executes *TagLeave*(k''_i, s''_i). Finally, it empties the trash list $\mathcal{L}$.

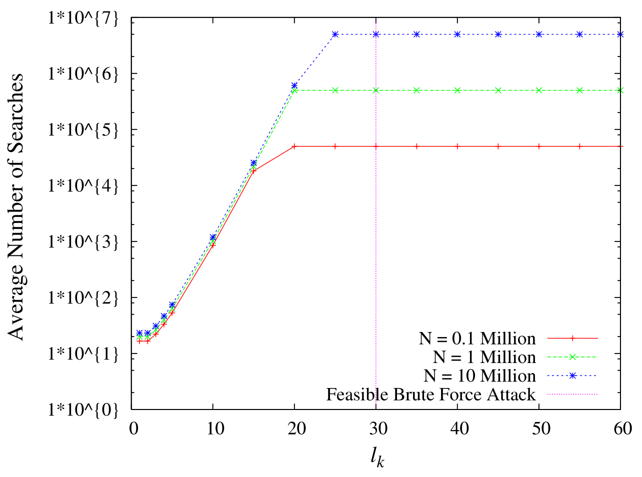

Fig. 2. Search complexity versus the length of $k_i[]$ and l_k.

4. Security

In this section, we show that (1) **Read** is a secure authentication protocol for CRFID and (2) privacy is preserved in CRFID. First, CRFID protocol updates the path key and secret key once an authentic read has been completed. Second, the complexity of the number of keys stored in tags is $\mathcal{O}(1)$. Third, readers are required to match d hash values from U to identify the leaf node. Each hash value requires at most δ hash computations. Since each node only contains a few tags, the overall search complexity is bounded by $\mathcal{O}(\delta \times d)$.[2] Thus, CRFID has three properties: resilience to compromising attacks, a constant key size, and high searching efficiency. CRFID withstand the two major types of RFID attack as follows.

1. *Resilient to tracking attacks*. CRFID significantly increases the length of each subkey of k_i from the suggested 4 bits to 60 bits while maintaining the same δ and thus the same search complexity. This improvement strengthens the protocol in avoiding tracking attacks. An attacker can no longer extract the value of each subkey from σ. Fig. 2 shows the average number of searches versus l_k. If l_k is increased to 30, for which the value can still be inverted by brute force, the average number of searches required by the reader for $N = 0.1$, 1, or 10 millions tags is the same as the trivial solution described by Weis et al. [4].
2. *Resilient to desynchronization attacks*. Step 8 in Fig. 1 is added to the protocol to avoid desynchronization attacks. When the message generated in Step 8 is received by $\mathcal{R}$, we can assert that the keys stored in $\mathcal{T}$ have been updated. If the message does not arrive at $\mathcal{R}$, either (1) $\mathcal{T}$ did not update its key and terminated the protocol, or (2) $\mathcal{T}$ updated its key to (k_i', s_i') (the new key pair that was also generated at $\mathcal{R}$). In any case, $\mathcal{R}$ will re-read $\mathcal{T}$ until the message sent in Step 8 is received. Meanwhile, all key pairs generated in previous trials are trashed to avoid phantom tags (records that do not exist).

Assume the attacker is allowed to compromise any number of tags except those she wants to attack. When a tag is compromised, the secret key and key path are revealed to the attacker. She is also allowed to eavesdrop, transmit, and overwrite messages during **Read**.

4.1. Authenticity

A protocol is said to be a secure authentication protocol if an attacker cannot be authenticated by the other protocol participants (in our case, the reader and a tag) on behalf of another protocol participant that was not been compromised by the attacker. This also implies that the attacker cannot reveal the secret key pair for a tag that it does not compromise.

We construct a simulator that contains an attacker who breaks the protocol. The simulator shows that the attacker has limited ability to break the protocol. The simulator interacts with the attacker as follows. When the reader is required to send a message M to a tag (or vice versa), the simulator outputs M to the attacker, who modifies it to M' and replies to the simulator. When the attacker wants to compute a hash function, she issues a hash query to the simulator. The simulator maintains a hash list that tracks the query records. If the query was never made before, the simulator replies with a hash value of length l_h and adds the query and random string to the hash list. If the query is in the hash list, the simulator simply replies with the value from the hash list. If the attacker wants to compromise a tag $\mathcal{T}_j$, the key pair (k_j, s_j) stored inside the tag is given to the attacker.

Suppose the tag being targeted is $\mathcal{T}_i$, which contains the key pair (k_i, s_i). The simulator starts **Read** and sends a random number n_1 to the attacker. The attacker responds with n_1'. The simulator then replies with $U = (n_2, U_0, U_1, \dots, U_{d-1}, U_s)$, where $(U_0, \dots, U_{d-1})$ are computed according to the protocol and U_s is a random string of length l_h. Note that l_h is the length of the hash $\mathcal{H}()$. The attacker needs to output $U' = (n_2', U_0', U_1', \dots, U_s')$ to the simulator.

The simulator rejects the message and terminates $\mathcal{R}$ if $U' \neq U$ or $n_1' \neq n_1$. We denote by E_1 the event whereby the simulator wrongly rejects the message (i.e., the attacker successfully breaks the protocol but the simulator rejects it). If the message is rejected, the attacker may opt to output σ' to fool the tag.

If $U' = U$ and $n_1' = n_1$, the simulator accepts the message and continues the protocol. It outputs $\sigma = \{\sigma_1, \sigma_2\}$ to the attacker, where both σ_1 and σ_2 are random numbers of length l_h. Again, the attacker responds with σ'. The simulator rejects σ' if it does not equal σ. If U' is rejected by the simulator, the simulator rejects σ' as well. We denote by E_2 the event whereby σ' is in fact legitimate according to Algorithm 2, but the simulator wrongly rejects it.

Algorithm 2 *NewVerify*

```
1: INPUT: $\sigma = \{\sigma_1, \sigma_2\}, n_1, n_2$.
2: OUTPUT: New Path Key $k_i'$, Secret Key $s_i'$.
3: Internal Variable: Current Path Key $k_i$, Secret Key $s_i$.
4: $k_i' \leftarrow \sigma_1 \oplus \mathcal{H}(n_1||n_2||k_i)$
5: if $\sigma_2 \neq \mathcal{H}(n_1||n_2||s_i||k_i')$ then
6:     return Fail;
7: end if
8: $s_i' \leftarrow \mathcal{H}(k_i||k_i'||s_i)$
9: return $k_i', s_i'$
```

Finally, the simulator outputs a random α of length l_h to the attacker. The attacker responds with α'. The simulator rejects α' if it does not equal α. We denote by E_3 the event whereby α' should not be rejected, but the simulator rejects it anyways.

If all messages are not changed by the attacker, the simulator assigns a random key pair (k_i, s_i) to the tag.

By definition, the probability that the adversary successfully breaks the protocol is bounded by $\Pr(E_1) + \Pr(E_2) + \Pr(E_3)$. Next, we prove that each of the above terms is negligibly small.

$\Pr(E_1)$: The event E_1 only happens if U_s' is legitimate. This may be due to the followings reasons:

1. If the same n_1 and n_2 have been used in previous failed reads.[3] This happens with probability $q_R \times 2^{-l_n}$, where q_R is the total

[2] If $N \ll \delta^d$, it would be more precise to denote the complexity as $\mathcal{O}(\log N)$.

[3] Failed reads would not change the key inside the tag, and thus U was sent in the previous session if the same n_1 and n_2 are applied.

number of reads allowed by the adversary and n is the length of n_1 and n_2. Note that even though n_2 can be chosen by the adversary, n_1 is randomly chosen by the simulator and cannot be predicted.

2. The attacker has queried hash values with the correct key. Since all the messages in the protocol are not related to key s_i, this only happens with probability of at most $q_H \times 2^{-l_k}$, where q_H is the total number of hash queries made by the attacker.[4]
3. The attacker outputs a legitimate U_s' that was not obtained from hash queries or previous communications. This only happens with probability 2^{-l_h}.

Thus, we can estimate the upper bound of $\Pr(E_1)$ as

$$\Pr(E_1) \le q_R \times 2^{-l_n} + q_H \times 2^{-l_k} + 2^{-l_h}. \tag{3}$$

$\Pr(E_2)$: E_2 only occurs when σ_2 is legitimate. We can further divide this event into three cases.

1. If the same n_1 and n_2 are used in the previous round, which only happens in $q_R \times 2^{-l_n}$, the same σ can be reused by the attacker.
2. If the attacker has queried a hash in the form $\mathcal{H}(n_1 \| n_2 \| s_i \parallel x)$ for any value x, she can also produce a legitimate σ. However, since all messages in our protocol are not related to s_i, this will only happen with probability $q_H \times 2^{-l_k}$.
3. Other than the above two cases, the attacker generates a legitimate σ_2 without querying the hash function. The probability of this happening is bounded by 2^{-l_h}.

Thus, we conclude that the upper bound of $\Pr(E_2)$ is given by

$$\Pr(E_2) \le q_R \times 2^{-l_n} + q_H \times 2^{-l_k} + 2^{-l_h}. \tag{4}$$

$\Pr(E_3)$: E_3 only happens when α is legitimate. Similar to the above analysis, this occurs only when:

1. α is a previous simulator output, which will only happen $q_R \times 2^{-2l_n}$ since both n_1 and n_2 cannot be controlled by the adversary.
2. The attacker queries a hash value in the form $\mathcal{H}(n_1 \| n_2 \| k_i' \parallel k_i)$, which only happens with probability $q_H \times 2^{-2l_k}$.
3. α is unrelated to the hash queries for any past communication. This α is legitimate with probability 2^{-l_h}.

Thus, $\Pr(E_3)$ is bounded by

$$\Pr(E_3) \le q_R \times 2^{-2l_n} + q_H \times 2^{-2l_k} + 2^{-l_h}. \tag{5}$$

To summarize, the overall probability for breaking of protocol $\Pr(\mathcal{A})$ is given by

$$\begin{aligned}\Pr(\mathcal{A}) \le\ & q_R \times 2^{-l_n}(2^{-l_n} + 2) \\ & + q_H \times 2^{-l_k}(2^{-l_k} + 2) + 3 \cdot 2^{-l_h},\end{aligned} \tag{6}$$

which is negligible for reasonably large l_n, l_k, l_h and polynomial numbers of hash queries q_H and online reads q_R.

Note that (6) is independent of the number of tags compromised, since each tag has a different secret key s_i. By fitting some realistic value such as $(l_n, l_k, l_h, q_R, q_H) = (60, 60, 60, 2^{16}, 2^{30})$, the maximum probability for an attacker breaking CRFID is less than 2^{-29}.

4.2. Privacy

To tackle tracking attacks, the protocol must provide less information to attackers when a tag is illegally read. If an attacker does not compromise any tags to reveal path keys k_i, she will unable to finish the protocol and/or to extract any subkey of k_i. In this case the attacker has insufficient information to mount a tracking attack. Thus, the following discussion is only relevant if the attacker has compromised a set of tags and revealed some path keys for the system.

When a tag is read illegally, an attacker can conclude a possible set of tags $\mathcal{S}$ that may generate the message. We use the privacy function $P(c)$ to measure the privacy provided by each scheme. $P(c)$ is defined as $\frac{1}{|\mathcal{S}|}$, where c is the number of compromised tags. An ideal privacy-preserving scheme would generate a function $P(c) = 1/(N - c), \forall c \le N - 1$, where N is the total number of tags in the system.

In both legitimate reads (passive overhearing) and illegal reads (active scanning), the message $U = (n_2, U_0, U_1, \ldots, U_{d-1}, U_s)$ only provides information to an attacker to identify the tag. If the attacker does not compromise any tags, it is almost infeasible to confirm whether two communication flows belong to the same tag or not unless the attacker can guess the path key k_i or the secret key s_i from the communication, or if the same n_1 and n_2 were used in two failure reads (which is very unlikely). Thus, we can denote $P(0) \approx \frac{1}{N}$.

Note that all keys stored in a tag are updated after a legitimate read. The communication leaks no information to the attacker to identify this tag. Thus, we only focus our discussion on illegal reads that provide information to the attacker.

The more tags compromised by the attacker, the more path keys are revealed to the attacker. Let $\mathcal{C}$ be the set of compromised tags and let $\mathcal{K}$ be the union of all subkeys of path keys from set $\mathcal{C}$. Assume that tag $\mathcal{T}_i$ has key pair k_i and s_i and outputs the messages $U_0, U_1 \cdots U_{d-1}$ during an illegal read by the attacker. We define the event $E_{x,y}$ to be the case whereby $k_i[0], k_i[1], \ldots, k_i[x-1] \in \mathcal{K}$ but $k_i[x], k_i[x+2], \ldots, k_i[d-1] \notin \mathcal{K}$; in addition, y nodes under the node corresponding to $k_i[x]$ are compromised.

Given the event $E_{x,y}$, on average there are $\bar{N}_{x,y}$ out of the remaining $N - c - 1$ in the set $\mathcal{S}$, where $\bar{N}_{x,y}$ can be calculated as

$$\bar{N}_{x,y} = 1 + \frac{(\delta - y)(N - c - 1)}{\delta^x}. \tag{7}$$

To compute the probability of $E_{x,y}$, the following analogy is used. There are δ^x bins with c balls to fill. Let $\mathbf{C}$ be a group of δ bins. We want to find the probability that exactly y bins in $\mathbf{C}$ are not empty such that one particular bin in $\mathbf{C}$ (the node that generates the message) cannot be filled by any ball. Let β be the number of non-empty bins after filling some bins. $\Pr(E_{x,y})$ can be calculated as $f(c, y, 0)$ according to

$$\begin{aligned} f(c, y, \beta) \ =\ & \frac{\delta - \beta - 1}{\delta^x} f(c-1, y-1, \beta+1) \\ & + \frac{\delta^x - \delta + \beta}{\delta^x} f(c-1, y, \beta) \end{aligned} \tag{8}$$

$$f(1, 1, \beta) = \frac{(\delta - \beta - 1)}{\delta^x} \tag{9}$$

$$f(c, 0, \beta) = \left(\frac{\delta^x - \delta + \beta}{\delta^x}\right)^c \tag{10}$$

$$f(u, v, \beta) = 0, \quad \forall u < v. \tag{11}$$

Eq. (8) can be viewed as follows. When there are c balls, β bins in $\mathbf{C}$ are filled and y bins in $\mathbf{C}$ need to be filled. This can be accomplished in two ways. In the first case, an empty bin in $\mathbf{C}$ is filled with a ball, so there are $\beta + 1$ bins filled with $c - 1$ balls and $y - 1$ bins left. In the second case, if a ball is placed in neither an empty bin in $\mathbf{C}$ nor a forbidden bin, this leaves $c - 1$ balls and y bins to fill. Eq. (9) states that when there is one ball left for one bin, the ball can be placed in any one of the empty bins in $\mathbf{C}$ except the forbidden one. Thus, there are $\delta - \beta - 1$ bins. Eq. (10) represents the case whereby there are c balls left but no additional empty bins need to be filled. Then all balls must be placed either outside the

[4] This part is not considered in the proof of ACTION. As the result, the protocol suffers from tracking attacks.

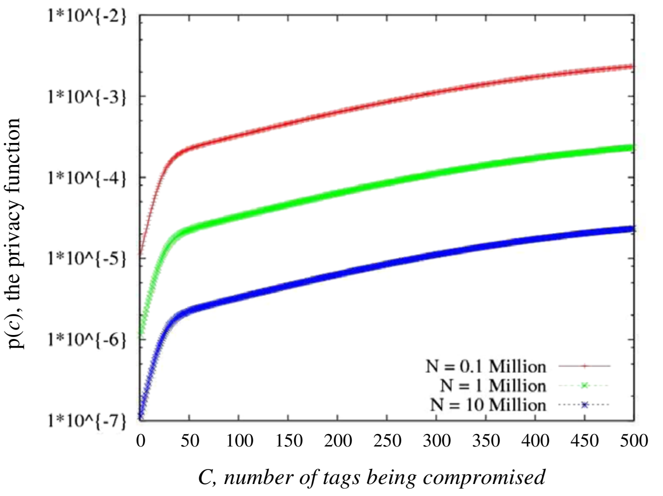

Fig. 3. Privacy level provided by the scheme as a function of the number of compromised tags for 0.1M-tag, 1M-tag, and 10M-tag systems.

group **C** or into non-empty bins, where there are $\delta^x - \delta + \beta$ bins in total. Eq. (11) addresses the case whereby there are not enough balls to fill the remaining bins; the probability for this situation is zero.

The privacy function $P(c)$ can be computed by combining (7)–(11):

$$P(c) = \frac{1}{\sum_{x=1}^{d-1}\sum_{y=1}^{\delta-1} \Pr(E_{x,y})\bar{N}_{x,y}}. \tag{12}$$

Fig. 3 shows privacy as a function of the number of tags being compromised for systems of different scales.

Note that a tracking attack only works for active scans before a tag is read legitimately again. Once a tag is read by the reader, the key pair stored in the tag is updated and the information tracked by the attacker will no longer be useful.

5. Discussion

5.1. Parameter setting

In this section, we discuss how the system parameters should be chosen for high scalability, quick searching, short authentication times, and high security levels for CRFID. Four parameters are discussed in detail: δ, d, N, and l_k. The total number of tags N depends on the application. We will see that δ and d should be set as a function of N. Setting of l_k is independent of the other parameters.

Our design idea involves the lowest calculation and waiting times for both $\mathcal{R}$ and $\mathcal{T}$. Thus, $\mathcal{R}$ waits when $\mathcal{T}$ is calculating; similarly, $\mathcal{T}$ waits when $\mathcal{R}$ is calculating. To avoid too many tags sharing the same path key (to reduce the search time for legitimate reading and prevent tracking attacks when tags are compromised), δ^d should be proportional to N. δ is the branching factor for the key tree. A small δ value can speed up the search time when a tag is read; however, it may also lead to a value of d that is too large. Thus, $\mathcal{T}$ and $\mathcal{R}$ would have to perform the hash operation more times and the total time for the authentication process would increase. Conversely, a large δ value can reduce the tag computation load, but will increase the $\mathcal{R}$ search time when identifying $\mathcal{T}$. Besides, the key size stored in $\mathcal{T}$ is $d \times l_k$, which also relies on d. Rewritable storage on some types of RFID tags can be a key constraint when choosing parameters.

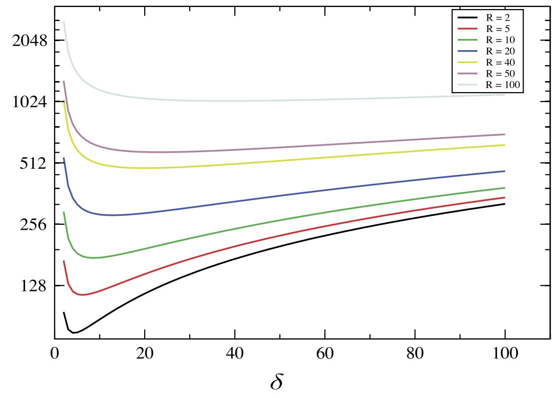

Fig. 4. Authentication cost as a function of the computation ratio δ. (y-axis represents the authentication cost.)

According to the proposed scheme, $\mathcal{R}$ performs one exclusive-or operation and a maximum of $d \times \delta + 5$ hash operations; $\mathcal{T}$ performs one exclusive-or operation and $d + 5$ hash operations. Since the exclusive-or operation has a relatively minor computation load compared with the hash operation, we only consider the total hash operation performed. An important issue we consider is that the computational abilities of $\mathcal{R}$ and $\mathcal{T}$ differ because of their hardware features. $\mathcal{R}$ has more powerful computation ability than $\mathcal{T}$. Since $\mathcal{R}$ only authenticates one tag at a time within an extremely short period, each the time for each authentication event should be minimized. To balance the execution time as evenly as possible between the reader side and the tag side, the minimum cost for $\mathcal{R}$ and $\mathcal{T}$ should be calculated. Hence, we examined different *computation ratios* R between $\mathcal{R}$ and $\mathcal{T}$. $R = 1$ means that the time spent for $\mathcal{R}$ to perform one hash operation is the same as for $\mathcal{T}$. $R = 100$ means that $\mathcal{R}$ is 100 times faster than $\mathcal{T}$ when performing one hash operation. Then we estimated the authentication cost for different R values to obtain the best δ and d values. Fig. 4 shows the authentication cost ($Auth\_cost$) as a function of δ for different R values according to Eq. (13).

$$Auth\_cost = (\delta \times d + 5) + (d + 5) \times R. \tag{13}$$

5.2. Case study

The total numbers of tags N may vary in different applications. For simplicity, we take $N = 2^{20}$ (i.e., a million) as an example and discuss the other parameter settings accordingly. For the case of $\delta = 16$ and $d = 5$, the key tree contains $16^5 (=2^{20})$ leaf nodes. This means that there are 2^{20} different combinations for the tree path. Since $\delta = 16$, every internal node of the tree is assigned 16 indices, each of which is a l_k-bit random number. The path key for $\mathcal{T}$ is a combination of the corresponding index from the root to the leaf node. For the CRFID protocol, only $\mathcal{R}$ knows these 16 randomly generated values for each internal node; nevertheless, for the ACTION protocol, the subkey is exactly the values of $1, 2, \ldots, 2^{l_k}$, which is publicly known. In addition, the search complexity is related to the choice of δ.

Fig. 3 shows that l_k should be greater than 60 bits to maintain the security requirement. In other words, each subkey is a 60-bit-long random number. Although there are still only 16 different subkeys for each node ($\delta = 16$), an attacker does not know these 16 possible values since each subkey is an unrelated large 2^{60}-bit value. Therefore, an attacker cannot enumerate all 2^{60} possible values to retrieve 16 possible hash values. The probability of guessing a correct subkey decreases from 2^{-4} to 2^{-60}. Thus, if it takes 1 s to break 2^{20} hash values, the total time needed will be 2^{40} s instead of 2^{-4}. This significantly increases the breaking time. Consequently, the proposed scheme not only maintains the same search complexity of $O(\log N)$ but also greatly decreases the breaking probability.

5.3. *Scalability*

The proposed scheme was originally designed so that each leaf node is associated with only one tag. Hence, for a key tree with $(\delta, d) = (32, 4)$, there are 2^{20} leaf nodes, corresponding to a tag capacity of 2^{20}. In reality, the system can hold more than 2^{20} tags. That is, a leaf node may be associated with more than one tag. It is also guaranteed that the tolerance of the situation that the amounts of tags are greater than 2^{20}.

Although these two tags have the same path key, their secret keys are different. Moreover, the key updating mechanism refreshes the path key and secret key on log-in. Therefore, if a tag is compromised, the attacker is unable to trace the other tags that belong to the same leaf node since the path key is changed immediately. Moreover, the secrecy of a tag is provided by its secret key, which is different from that of other tags. Hence, the proposed scheme holds more than 2^{20} tags. Note that 2^{20} tags is enough for most current applications.

6. Conclusion

We proposed a secure and efficient RFID authentication scheme and presented a formal security analysis and detailed discussion. The parameters and security strength were confirmed by experimental results. Our improved solution maintains the same search complexity and significantly improves the security strength of each part of the path key. Most importantly, to protect against desynchronizing attacks, the reader reads the tag again after updating the key. If the tag responds with the message created by the previous key, this indicates that key updating has not yet been completed.

References

[1] K. Michael, L. McCathie, The pros and cons of RFID in supply chain management, in: Proceedings of the International Conference on Mobile Business, 2005, pp. 623–629.
[2] A. Juels, D. Molnar, D. Wagner, Security and privacy issues in e-passports, in: Proceedings of the International Conference on Security and Privacy for Emerging Areas in Communications Networks, 2005, pp. 74–88.
[3] Electronic Product Code Global Inc., http://www.epcglobalinc.com.
[4] S. Weis, S. Sarma, R. Rivest, D. Engels, Security and privacy aspects of low-cost radio frequency identification systems, in: Proceedings of the 1st International Conference on Security in Pervasive Computing, Springer, 2004, pp. 201–212.
[5] A. Juels, S.A. Weis, Defining strong privacy for RFID, in: Workshop of the 5th Annual IEEE International Conference on Pervasive Computing and Communications, 2007, pp. 342–347.
[6] T. Dimitriou, A secure and efficient RFID protocol that could make big brother (partially) obsolete, in: Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications, 2006, pp. 269–275.
[7] D. Molnar, D. Wagner, Privacy and security in library RFID: issues, practices, and architectures, in: Proceedings of the 11th ACM Conference on Computer and Communications Security, ACM, New York, NY, USA, 2004, pp. 210–219.
[8] D. Molnar, A. Soppera, D. Wagner, A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags, in: Selected Areas in Cryptography, Vol. 3897, 2006, pp. 276–290.
[9] L. Lu, J. Han, L. Hu, Y. Liu, L. Ni, Dynamic key-updating: privacy-preserving authentication for RFID systems, in: Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications, 2007, pp. 19–23.
[10] L. Lu, J. Han, R. Xiao, Y. Liu, ACTION: breaking the privacy barrier for RFID systems, in: INFOCOM 2009, IEEE, 2009, pp. 1953–1961.
[11] H.M. Sun, S.M. Chen, K.H. Wang, Cryptanalysis on the RFID action protocol, in: International Conference on Security and Management, 2011.
[12] H.-Y. Chien, SASI: a new ultralightweight RFID authentication protocol providing strong authentication and strong integrity, IEEE Transactions on Dependable and Secure Computing 4 (4) (2007) 337–340.
[13] S. Piramuthu, Lightweight cryptographic authentication in passive RFID-tagged systems, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38 (3) (2008) 360–376.
[14] B. Song, C.J. Mitchell, RFID authentication protocol for low-cost tags, in: Proceedings of the 1st ACM Conference on Wireless Network Security, ACM, New York, NY, USA, 2008, pp. 140–147.
[15] H.-M. Sun, W.-C. Ting, K.-H. Wang, On the security of Chien's ultralightweight RFID authentication protocol efficient time-bound hierarchical key management, IEEE Transactions on Dependable and Secure Computing 8 (2011) 315–317.
[16] R.-W. Phan, Cryptanalysis of a new ultralightweight RFID authentication protocol—SASI, IEEE Transactions on Dependable and Secure Computing 6 (4) (2009) 316–320.
[17] P. Rizomiliotis, E. Rekleitis, S. Gritzalis, Security analysis of the Song–Mitchell authentication protocol for low-cost tags, IEEE Communications Letters 13 (4) (2009) 274–276.
[18] P. Robinson, M. Beigl, Trust context spaces: an infrastructure for pervasive security in context-aware environments, in: Lecture Notes in Computer Science, 2004, pp. 157–172.
[19] G. Avoine, E. Dysli, P. Oechslin, Reducing time complexity in RFID systems, in: Proceedings of Selected Areas in Cryptography SAC, 2005.

Shuai-Min Chen received a B.S. degree in computer science from Chinese Culture University in 2001 and an M.S. degree in computer science and engineering from Fu Jen Catholic University in 2004. He is currently a Ph.D. student in computer science at National Tsing Hua University. His research interests include RFID, heterogeneous networks, information security, and cryptography.

Mu-En Wu received a B.S. degree in mathematics from National Tsing-Hua University in 2002, an M.S. degree in applied mathematics from National Chiao-Tung University in 2004, and a Ph.D. degree in computer science from National Tsing-Hua University in 2009. He has now joined the Institute of Information Science (IIS) in Academic Sinica, Taiwan. His research interests include cryptography, information security, market prediction, and algorithmic game theory.

Hung-Min Sun received B.S. and M.S. degrees in applied mathematics from National Chung-Hsing University in 1988 and 1990, respectively, and a Ph.D. degree in computer science and information engineering from National Chiao-Tung University in 1995. He was an Associate Professor in the Department of Information Management, Chaoyang University of Technology from 1995 to 1999, the Department of Computer Science and Information Engineering, National Cheng-Kung University from 2000 to 2002, and the Department of Computer Science, National Cheng-Kung University from 2002 to 2008. Currently, he is a Professor with the Department of Computer Science, National Tsing Hua University. He has published more than 150 international journal and conference papers. He was the Program Co-chair of the 2001 National Information Security Conference, and has served as a program committee member for many international conferences. He was the Honorary Chair of the 2009 International Conference on Computer and Automation Engineering, the 2009 International Conference on Computer Research and Development, and the 2009 International Conference on Telecom Technology and Applications. He serves as a member of the editorial board of many international journals and of the program committee of international conferences. He has won many Best Paper awards in academic journals and conferences, including the 2003 annual Best Paper award for the Journal of Information Science and Engineering, the Best Paper award at MobiSys09, NSC05, NISC06, NISC07, CISC09, and ICS'2010. Professor Sun won the Y.Z. Hsu Scientific Paper Award, Far Eastern Y.Z. Hsu Science and Technology Memorial Foundation, 2010. His research interests include network security, cryptography, and wireless networks.

King-Hang Wang received a B.S. degree in information engineering from the Chinese University of Hong Kong in 2002 and a Ph.D. degree in computer science from the National Tsing Hua University in 2010. He is currently a lecturer in the Hong Kong Institute of Technology. His research interests include information security, digital rights management, steganography, and mobile authentication. He has been an IEEE member since 2006.