Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 153912, 12 pages
doi:10.1155/2012/153912

Research Article

Dynamic Key-Updating: Privacy-Preserving Authentication for RFID Systems

Li Lu,[1] Jinsong Han,[2] Lei Hu,[3] and Lionel M. Ni[4]

[1] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*
[2] *School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China*
[3] *State Key Laboratory of Information Security, Chinese Academy of Sciences, Beijing 100049, China*
[4] *Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China*

Correspondence should be addressed to Li Lu, lulirui@gmail.com

Received 14 December 2011; Accepted 14 February 2012

Academic Editor: Mo Li

The objective of private authentication for Radio Frequency Identification (RFID) systems is to allow valid readers to explicitly authenticate their dominated tags without leaking the private information of tags. In previous designs, the RFID tags issue encrypted authentication messages to the RFID reader, and the reader searches the key space to identify the tags. Without key-updating, those schemes are vulnerable to many active attacks, especially the compromising attack. We propose a strong and lightweight RFID private authentication protocol, SPA. By designing a novel key-updating method, we achieve the forward secrecy in SPA with an efficient key search algorithm. We also show that, compared with existing designs, (SPA) is able to effectively defend against both passive and active attacks, including compromising attacks. Through prototype implementation, we demonstrate that SPA is practical and scalable for current RFID infrastructures.

1. Introduction

The proliferation of RFID applications [1–5] raises an emerging requirement—protecting user privacy [6] in authentications. In most RFID systems, tags automatically emit their unique serial numbers upon reader interrogation without alerting their users. Within the scanning range, a malicious reader thus can perform bogus authentication on detected tags to obtain sensitive information. For example, without privacy protection, any RFID reader is able to identify a consumer's ID via the serial number emitted from the tag. In this case, a buyer can be easily tracked and profiled by unauthorized entities. Nowadays, many companies embed RFID tags with produced items. Those tags indicate the unique information of the items they are attached to. Thus, a customer carrying those items might easily get subject to silent track from unauthorized entities in a much larger span. Some sensitive personal information would thereby be exposed: the illnesses she suffers from indicated by the pharmaceutical products; the malls where she shops; the types of items she prefers, and so on. To prevent such unexpected leakage of private information, a secure RFID system must meet two requirements. First, a valid reader must be able to successfully identify the valid tags; on the other hand, misbehaving readers should be isolated from retrieving private information from these tags.

To address the above issue, researchers employ encryptions in RFID authentication. Each tag shares a unique key with the RFID reader and sends an encrypted authentication message to the reader. Instead of identifying the tag directly, the back-end database subsequently searches all keys that it holds to recover the authentication message and identify the tag. For simplicity, we will denote the reader device and back-end database by the "reader" in what follows. Two challenging issues on the reader side must be addressed in the key storage infrastructure and search algorithm: the search efficiency and the security guarantee. Searching a key should be sufficiently fast to support a large-scale system, while the maintained keys should be dynamically updated to meet security requirements.

Many efforts have been made to achieve efficient private authentication. To the best of our knowledge, the most

efficient protocols are tree based [7, 8]. They provide efficient key search schemes with logarithm complexity. In those approaches, each tag holds multiple keys instead of a single key. A virtual hierarchical tree structure is constructed by the reader to organize these keys. Every node in the tree, except the root, stores a unique key. Each tag is associated with a unique leaf node. Keys in the path from the root to the leaf node are then distributed to this tag. If the tree has a depth d and branching factor δ, each tag contains d keys and the entire tree can support up to $N = d^{\delta}$ tags. A tag encrypts the authentication message by using each of its d keys. During authentication, the reader performs a depth-first search in the key tree. In each hierarchy, the reader can narrow the search set within δ keys. Thus, the reader only needs to search $d\delta$ keys for each tag's authentication. Therefore, the key search complexity of identifying a given tag from N tags is logarithmic in N.

The tree-based approaches are efficient, nevertheless, not sufficiently secure due to the lack of a key-updating mechanism. Most tree-based approaches do not update keys of tags dynamically. Since the storage infrastructure of keys in tree-based approaches is static, each tag shares common keys with others. Consequently, compromising one tag will reveal information of other tags. To address this problem, we need to provide a dynamic key-updating mechanism to such approaches. The major challenge of dynamic key-updating in tree-based approaches is consistency. If a single tag updates its keys, some other tags have to update their keys accordingly. Till now, consistent and dynamic key-updating mechanisms have scarcely been seen in the literatures.

In this paper, we propose a strong and lightweight RFID private authentication protocol, (SPA), which enables dynamic key-updating for tree-based authentication approaches. Besides consistency, SPA also achieves forwarding secrecy without degrading key search efficiency. We also show that SPA outperforms existing designs in defending against both passive and active attacks, including the compromising attack.

The rest of this paper is organized as follows. We introduce related work in Section 2. We present the SPA design in Section 3. In Section 4, we analyze the security guarantee of SPA. We evaluate the performance of SPA via a prototype implementation in Section 5. We conclude this paper in Section 6.

2. Related Work

Many approaches have been proposed to achieve private authentication in RFID systems. Weis et al. [9] proposed a hash-function-based authentication scheme, HashLock, to avoid tags being tracked. In this approach, each tag shares a secret key k with the reader. The reader sends a random number r as the authentication request. To respond to the reader, the tag generates a hash value on the inputs of r and k. The reader then computes $h(k, r)$ of all stored keys until it finds a key to recover r, thereby identifying the tag. The search complexity of HashLock is linear to N, where N is the number of tags in the system. Most subsequent approaches in the literature are aimed at reducing the cost of key search. Juels [10] classifies these approaches into three types.

Tree-Based Approaches. Tree based approaches [7, 8, 11] improve the key search efficiency from linear complexity to logarithmic complexity. Molnar and Wagner proposed the first tree-based scheme, which employs a challenge-response scheme [8], which achieves *mutual* authentication between tags and readers. The protocol uses multiple rounds to identify a tag and each round needs three messages. Since it requires $O(\log N)$ rounds to identify a tag, the exchanged messages incur relatively large communication overhead. In [7], the authors provide a more efficient scheme which performs the authentication via one message from the tag to the reader and no further interactions. However, the tree-based approaches are often vulnerable to the *Tag Compromising Attack*. Because tags share keys with others in the tree structure, compromising one tag results in compromising secrets in other tags.

Synchronization Approaches. Synchronization approaches [12] make use of an incremental counter to enhance the authentication security. When successfully completing an authentication, the counter of a tag augments by one. The reader can compare the value of a tag's counter with the record in the database. If they match, the tag is valid and the reader will synchronize the counter record of this tag. However, incomplete authentications lead the tag's counter larger than the one held by the reader. To solve this problem, the reader keeps a window for each tag. Such a window limits the maximum value of the counter held by the tag. If a tag's counter is larger than the record held by the reader but within the window, the reader still regards this tag as valid. Such schemes are vulnerable to the *Desynchronization Attack*. In such an attack, an invalid reader can interrogate a tag many times so that the counter of this tag exceeds the window recorded in the valid reader. In [13], the authors proposed a protocol to mitigate the impact from desynchronization attacks by allowing tags to report the number of incomplete authentications since the last successful authentication with the reader. Dimitriou proposed a scheme in [14], in which a tag increases its counter only after successful mutual authentications. Those protocols, however, degrade the anonymity of tags. An attacker is still able to interrogate a given tag enough times so that the tag will be immediately recognized when replying with unchanged responses.

Time-Space Tradeoff Approaches. Avoine converted the key search problem to an attempt at breaking a symmetric key [15]. In [16], Hellman studied the key-breaking problem and claimed that recovering a symmetric key k from a ciphertext needs to precompute and to store $O(N^{2/3})$ possible keys. Accordingly, the key search complexity is $O(N^{2/3})$ in key-breaking-based approaches. Obviously, those approaches are not efficient compared with tree-based approaches.

3. SPA Protocol

In this section, we first introduce the challenging issues of static tree-based private authentication approaches. We then present the design of SPA.

3.1. Challenges of Tree-Based Approaches. Existing tree-based approaches [7, 8, 11] construct a balanced tree to organize and store the keys for all tags. Each node stores a key and each tag is arranged to a unique leaf node. Thus, there exists a unique path from the root to this leaf node. Correspondingly, those keys on this path are assigned to the tag. For example, tag T_1 obtains keys k_0, $k_{1,1}$, $k_{2,1}$, and $k_{3,1}$, as illustrated in Figure 1. When a reader authenticates a tag, for example, T_1, it conducts the identification protocol shown in Figure 2. $h(k, r)$ denotes the output of a hash function h on two input parameters: a key k and a random number r. The identification procedure is similar to traversing a tree from the root to a leaf. The reader R first sends a nonce r to tag T_1. T_1 encrypts r with all its keys and includes the ciphertexts in a response. Upon the response from T_1, the reader searches proper keys in the key tree to recover r. This is equal to marking a path from the root to the leaf node of T_1 in the tree. At the end of identification, if such a path exists, R regards T_1 as a valid tag. Usually, the encryption is employed by using cryptographic hash functions.

From the above procedure, we see that tags will, more or less, share some nonleaf nodes in the tree. For example, T_1 and T_2 share $k_{2,1}$, while T_1, T_2, T_3, and T_4 share $k_{1,1}$. Of course all tags share the root k_0. Such a static tree architecture is efficient because the complexity of key search is logarithmic. For the example in Figure 1, any identification of a tag only needs $\log_2(8) = 3$ search steps. On the other hand, if the adversary tampers with some tags; however, it obtains several paths from the root to those leaf nodes of the corrupt tags, as well as the keys on those paths. Since keys are not changed in the static tree architecture, the captured keys will still be used by those tags which are not tampered (for simplicity, we denote those tags as *normal* tags). Consequently, the adversary is able to capture the secret of normal tags.

A practical solution is to update keys for a tag after each authentication so that the adversary cannot make use of keys obtained from compromised tags to attack normal ones. However, the static tree architecture makes it highly inflexible to provide consistent key-updating. Suppose we update the keys of T_1 in Figure 1, we have to change k_0, $k_{1,1}$, $k_{2,1}$, and $k_{3,1}$ partially or totally. Note that $k_{1,1}$ is also used by T_2, T_3, and T_4, and $k_{2,1}$ is used by T_2. To keep the updating consistent, the keys of all influenced tags must be updated and re-distributed. A challenging issue is that if the position of a key is close to the root, the key-updating would influence more tags. For example, updating $k_{1,1}$ would influence half of all tags in the system (T_1, T_2, T_3, and T_4). One intuitive idea is to periodically recall all tags and update the keys simultaneously. Unfortunately, such a solution is not practical in large-scale systems with millions or even hundreds of millions of tags. Another solution is collecting those influenced tags only and updating their keys. This is also difficult because we have to collect a large span of tags even though there is only one tag updating its keys.

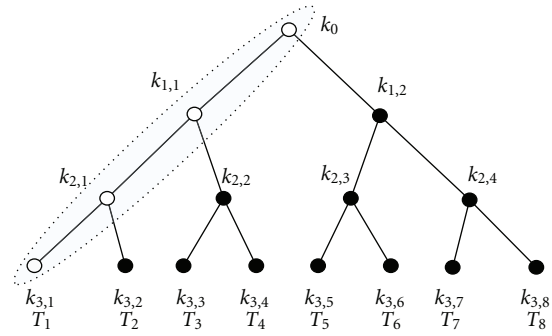


Figure 1: A binary key tree with eight tags.

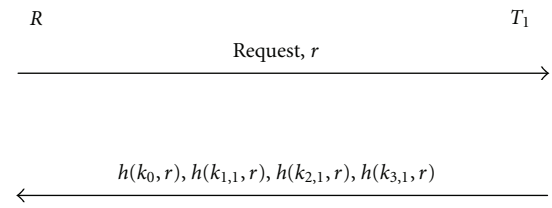


Figure 2: A basic RFID authentication procedure.

This problem motivates us to develop a dynamic key-updating algorithm for private authentication in RFID systems. This is where our proposed SPA enters the picture.

3.2. SPA Overview. SPA comprises four components: *system initialization*, *tag identification*, *key-updating*, and *system maintenance*. The first and second components are similar to tree-based approaches such as [7] and perform the basic identification functions. The key-updating is employed after a tag successfully performs its identification with the reader. In this procedure, the tag and the reader update their shared keys. This key-updating procedure will not break the validation of keys used by other tags. SPA achieves this using temporary keys and state bits. A temporary key is used to store the old key for each nonleaf node in the key tree. For each nonleaf node, a number of state bits are used in order to record the key-updating status of nodes in the subtrees. Based on this design, each nonleaf node will automatically perform key-updating when all its children nodes have updated their keys. Thus, SPA guarantees the validation and consistency of private authentication for all tags. SPA also eases the system maintenance in high dynamic systems where tags join or leave frequently by the fourth component.

3.3. System Initialization. For the simplicity of discussion, we use a balanced binary tree to organize and store keys, as shown by an example in Figure 3. Let δ denote the branching factor of the key tree (e.g., $\delta = 2$ when the key tree is a binary tree). We assume that there are N tags T_i, $1 \le i \le N$, and a reader R in the RFID system. The reader R assigns the N tags to N leaf nodes in a balanced binary tree S. Each nonleaf

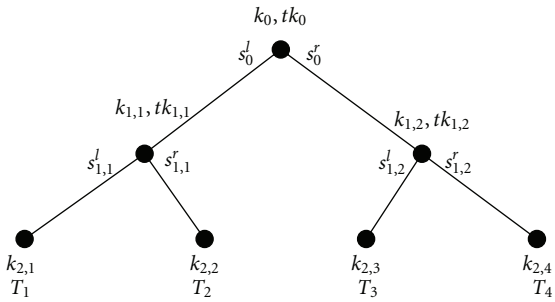


Figure 3: A key tree of a system with four tags ($N = 4$).

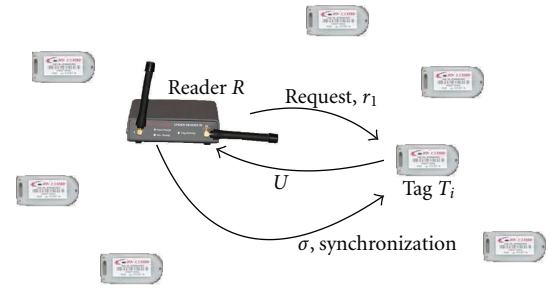


Figure 4: Authentication procedure in SPA. After receiving U, Reader R's operations are Step 1, identifying T_i and computes σ; Step 2, sending σ to T_i and key-updating. T_i also updates its keys after checking σ.

node j in S is assigned with two keys, a working key k_j and a temporary key tk_j. The usage of tk_j will be illustrated in Section 3.5. Initially, each key is generated independently randomly by the reader, and $tk_j = k_j$ for all nonleaf nodes.

When a tag T_i is introduced into the system, the reader distributes the $(\lceil \log N \rceil + 1)$ keys to T_i. Those keys are corresponding to the path from the root to tag T_i (for a nonleaf node j at the path, if $tk_j \neq k_j$, tag T_i is assigned with k_j). For example, the keys stored in tag T_1 are k_0, $k_{1,1}$ and $k_{2,1}$, as illustrated in Figure 3. Hereafter, we use d to denote the depth of the tree and $(k_0^i, k_1^i, \ldots, k_d^i)$ to denote the secret keys distributed to T_i.

3.4. Tag Identification. The basic authentication procedure between the reader and tags includes three rounds, as illustrated in Figure 4. In the first round, R starts the protocol by sending a "*Request*" and a random number r_1 (a nonce) to tag T_i, $1 \leq i \leq N$. In the second round, upon receiving the request, T_i generates a random number r_2 (a nonce) and computes the sequence $(h(k_0^i, r_1, r_2), \ldots, h(k_d^i, r_1, r_2))$, where $h(k, r_1, r_2)$ denotes the output of a hash function h on three inputs: a key k and two random numbers r_1 and r_2. T_i replies R with a message $U = (r_2, h(k_0^i, r_1, r_2), \ldots, h(k_d^i, r_1, r_2))$. For simplicity, we denote the elements in U as $u, v_0, \ldots, v_d$. R identifies T_i according to U.

R executes the basic identification procedure to identify T_i, represented as Step 1 in Figure 4. From the root, the reader first encrypts r_1 by using k_0 and compares the result with $h(k_0, r_1, r_2)$ from T_i. If they match, R invokes a recursive algorithm, Algorithm 1, as illustrated in Algorithm 1 to identify T_i. For the key tree in Figure 3, the reader starts from the root and encrypts r_1 by using $k_{1,1}$(*or* $tk_{1,1}$) and $k_{1,2}$ (or $tk_{1,2}$). Having the results, the reader compares them with received $h(k_1^i, r_1, r_2)$. If $h(k_1^i, r_1, r_2)$ is equal to the result computed from $k_{1,1}$ (or $tk_{1,1}$), the tag belongs to the left sub-tree; otherwise, it belongs to the right sub-tree.

Level by level, R figures out the path of T_i originated from the root by invoking Algorithm 1. Suppose the path reaches an intermediate node j at level l $(1 \leq l \leq d)$ in the tree. At this point, R computes all hash values $h(k_{l+1}, r_1, r_2)$ and $h(tk_{l+1}, r_1, r_2)$ by using the keys of node j's children, then compares them with v_l. Note that v_l is in the authentication message U received from T_i. If there is a match, T_i must belong to the sub-tree of the matched j's child node. Therefore, R extends the path to that node and continues the identification procedure until reaching a leaf node.

Identifying a tag is similar to traversing from the root to a leaf in the key tree. The path is determined by using Algorithm 1. Instead of using only one key for each node, Algorithm 1 uses both the working key k and the temporary key tk.

3.5. Key-Updating. After successfully identifying T_i, R invokes the Key-updating algorithm in Step 2, as shown in Figure 4.

When generating new keys, SPA still makes use of the hash function h. Let k_j be the old key of node j. The reader computes a new key k_j' from the old key k_j as $k_j' = h(k_j)$. The key-updating algorithm for the key tree is shown in Algorithm 2. To remain consistent, the nonleaf node j uses temporary key tk_j to store j's old key. In this way, the key-updating of a tag will not interrupt the authentication procedures of other tags belonging to j's sub-tree.

Two important issues must be addressed when updating keys. First, R should update the keys of the identified tag T_i without interrupting the identification of other tags. This is because the keys stored in nonleaf nodes are shared by multiple tags. Those keys should be updated in a consistent manner. Second, each nonleaf node should automatically update its keys when all its children have updated their keys.

To address the two issues, SPA introduces a number of state bits to each nonleaf node. The basic idea behind this mechanism is that each nonleaf node uses these bits to reflect the key-updating status of its children. Once a child has updated its key, the corresponding bit is set to 1. Each node updates its own key when all its state bits become 1.

Without losing generality, we still use balanced binary key tree S to illustrate this mechanism. Each nonleaf node j in S is assigned with two state bits, denoted as s_j^l and s_j^r, s_j^l, $s_j^r \in \{0, 1\}$, where s_j^l (s_j^r) represents the state whether or not the left (right) child of node j has updated its keys. When initializing the key tree S, $s_j^l = s_j^r = 0$ for all nonleaf nodes. At any time, if the key of node j's left (right) child is updated, SPA sets s_j^l (s_j^r) to 1.

```
Fix d ← log N;
SUCCEED ← false;
l ← DepthofNode(n);
  if (v_l = h(k_n, r_1, r_2) ∨ v_l = h(tk_n, r_1, r_2))
    if (l ≠ d)
      if v_l = h(tk_n, r_1, r_2)
        Record n in Synchronization
        Message;
      for i = 1 to δ
        m ← FindChildren(n, i);
        Identification (U, m);
    else if l = d
      SUCCEED ← true;
 if (¬SUCCEED)
    Fail and output 0;
else Accept and output 1;
```

Algorithm 1: Identification (U, node n). Tree-based identification.

```
if n is a nonleaf node
    Store the old key tk_n ← k_n;
Generate a new key k_n ← h(k_n);
m ← FindParent(n);
if n is the left child of m
    Set s^l_m ← 1;
else if n is the right child of m
    Set s^r_m ← 1;
if (s^l_m = 1 ∧ s^r_m = 1)
  Reset s^l_m and s^r_m to 0, and record m in
  Synchronization message;
if m is not the root node
  n ← m;
  Key-updating (n);
```

Algorithm 2: Key-updating (node n). Tree-based key-updating.

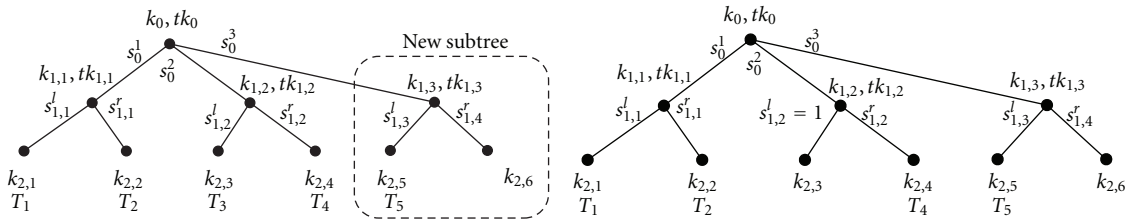
When R finishes key-updating, it sends a message $\sigma = h(k_d^i, r_1, r_2)$, as shown in Figure 4, and a *synchronization* message to T_i. The former one is used by T_i to authenticate R. The latter one includes the information of the levels on which the nodes have updated their keys in the key tree. After receiving these messages, T_i first verifies whether or not $\sigma = h(k_d^i, r_1, r_2)$. If yes, T_i updates its keys according to the synchronization message. For example, in Figure 3, suppose that R has updated keys $k_{1,1}$ and $k_{2,2}$ at levels 1 and 2 after identifying T_2. The synchronization message is (1, 2). Accordingly, T_2 updates $k_{1,1}$ as $k'_{1,1} = h'(k_{1,1})$ and $k_{2,2}$ as $k'_{2,2} = h(k_{2,2})$, respectively. This algorithm guarantees that the key-updating is consistent and feasible under arbitrary tag access patterns.

The key-updating algorithm is suitable for an arbitrary balanced tree with $\delta > 2$. In such a tree, there are δ state bits maintained in each nonleaf node to indicate the key-updating states of δ children.

3.6. System Maintenance. If a tag T_i is added to the RFID system, R checks whether or not there exists an empty leaf node in the key tree S. If yes, T_i is assigned to an empty leaf node. T_i's keys are then predistributed according to the path from the leaf node to the root of S. If there is no any empty leaf node in the tree S, R creates a new balanced tree S' with the branching factor δ and depth $d - 1$. R then initializes S' by employing the system initialization component described in Section 3.3.

After initialization, R connects the root of S to the root of S'. In this way, S' becomes a sub-tree of S. Hereby, T_i is assigned to an empty leaf node in S' and T_i's keys are pre-distributed according to the path from the leaf node to the root of S. For example, in Figure 5(a), R has 4 tags and each leaf node is occupied in R's key tree. If a new tag T_5 is added to the RFID system, R creates a new sub-tree and assigns a leaf node in this sub-tree to T_5. T_5's keys are k_0, $k_{1,3}$, and $k_{2,5}$.

For any empty leaf node i in the key tree, SPA lets the i's parent node j to set the corresponding state bit s_j to 1. Further, s_j is locked until the node i is assigned to a new tag T_i. This constraint is for protecting the key-updating of other tags from being interrupted. Indeed, since the node i is



(a) A new tag T_5 joins the RFID system

(b) A tag T_3 leaves the RFID system

Figure 5: Tag joining and leaving.

empty, no tag will trigger the change of s_j. Therefore, if s_j is 0, it will never change, such that node j will never update keys. To avoid such an interruption, SPA sets the s_j to 1 to allow other tags update their keys continuously and consistently.

If a tag leaves, R sets the leaf node of the leaving tag to be empty. For example, in Figure 5(b), if the tag T_3 leaves, R empties the leaf node distributed to T_3 and let $s_{1,2}^l = 1$.

3.7. An Example of Key-Updating. For the ease of understanding, we use an example to explain the key-updating algorithm. Here we show one reader R and four tags a, b, c, d (i.e., four leaf nodes in the key tree). We assume the sequence of tag authentication is (a, b, c, d, a). The original state of the key tree is shown in Figure 6(a). When tag a is identified, R sets the corresponding state bit of a's parent to 1. Meanwhile, R generates a new key of the leaf a as shown in Figure 6(b).

When tag b is identified, the corresponding state bit of b's parent is set to 1, and R updates the keys of the leaf node b and b's parent as shown in Figure 6(c).

When tag c is identified, the situation is similar to that of tag a. Since all state bits of the parent node of a and b are set to 1, R clears the state bits (i.e., reset the state bits to 0) in the key-updating operation as illustrated in Figure 6(d).

When tag d is identified, all state bits of d's parent and the root are set to 1. Thus, R updates the keys of the path from the leaf node of d to the root as shown in Figure 6(e).

Since tag a does not know that the keys of a's parent and the root have been updated, it will still use the old keys for identification. As indicated in the description of Algorithm 2 (Algorithm 2 in Section 3.5), each node stores the old key as the temporary key. After identifying tag a, R informs tag a to update its keys, according to the new keys from the leaf node of a to the root in the tree, as shown in Figure 6(f).

4. Discussion

In this section, we first discuss the security requirements for designing private authentication protocols in RFID systems. To evaluate the security of SPA, we propose an attack model to represent existing attacking scenarios. We then demonstrate the ability of SPA to meet those requirements and to defend against attacks.

4.1. Security Requirements. A private authentication protocol should meet the following security requirements [7].

Privacy. The private information, such as tag's ID, user name, and other private information should not be leaked to any third party during authentication.

Untraceability. A tag should not be correlated to its output authentication messages; otherwise, it may be tracked by attackers.

Cloning Resistance. Attackers should not be able to use bogus tags to impersonate a valid tag. Also, the replay attack should be resisted.

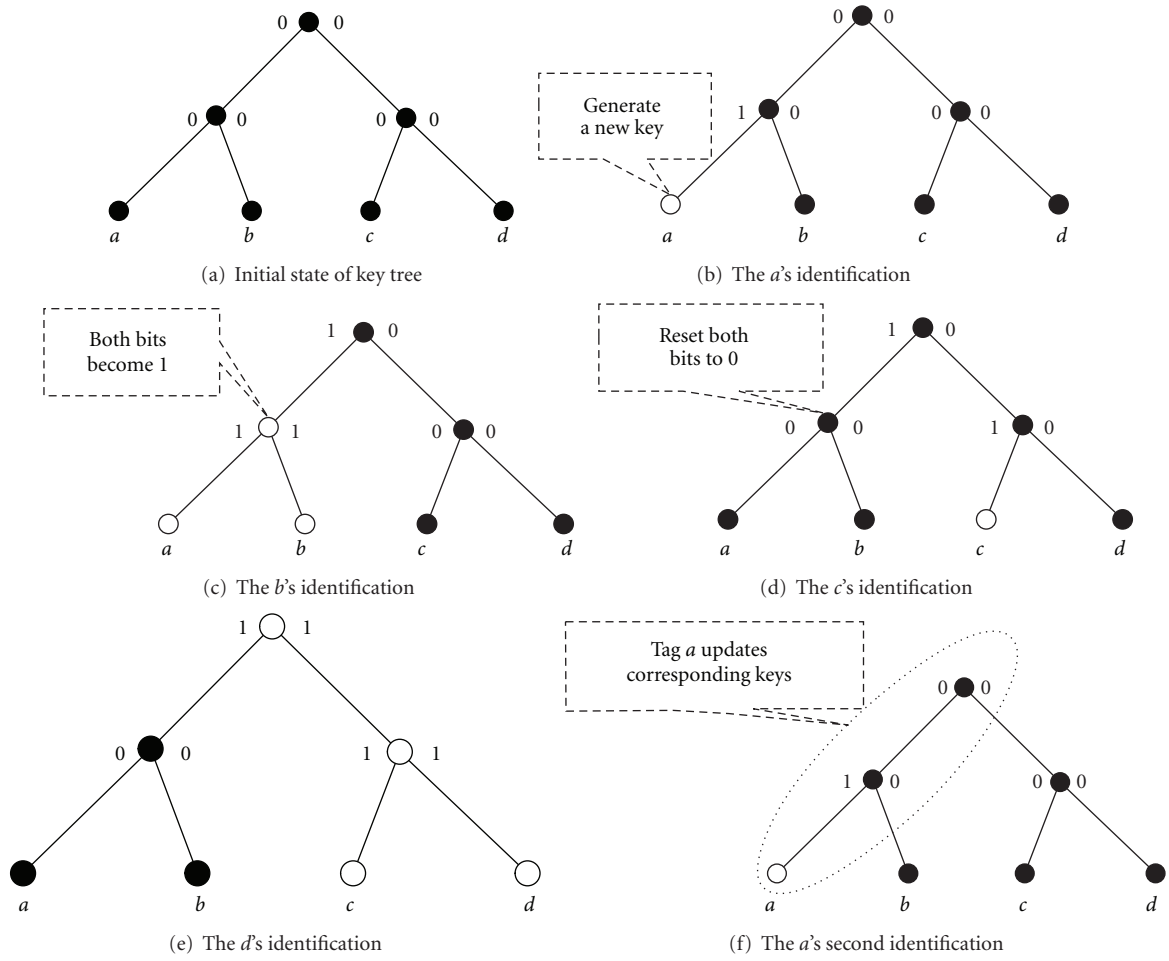
Forward Secrecy. Attackers can compromise a tag to obtain the keys stored in it. In this case, those keys should not reveal the previous outputs of the captured tag.

Compromising Resistance. The privacy of normal tags is threatened if they share some keys with compromised tags. Thus, the number of affected tags should be minimized after a successful compromising attack.

Existing private authentication approaches are able to defend against passive attacks (i.e., eavesdropping), but are vulnerable to active attacks (i.e., cloning and compromising attacks). Therefore, our discussion will focus on how SPA protects tags from active attacks. From the attacker's perspective, two metrics are important for evaluating the capability of SPA in defending against active attacks: (a) *past-exposing probability,* the probability of successfully identifying the past outputs of a compromised tag—this metric reflects the forward secrecy property of an authentication scheme; (b) *correlated-exposing probability,* the probability of successfully tracing a tag when some other tags in the system are compromised.

4.2. Attack Model. Avoine [17] proposes an attack model for RFID system and introduces the concept *untraceability*. The model well-reflects the attack behaviors and impacts on the authentication protocols. Our discussions are mainly based on this model with slight modification as follows.

In our model, the attackers and the RFID system are abstracted into two participants: the *Adversary A* (the attackers) and the *Challenger C* (the RFID system). So, the model is like a game between A and C. A first informs C that A will start to attack. C then chooses two tags to perform SPA protocols. If A can successfully distinguish any



(a) Initial state of key tree
(b) The a's identification
(c) The b's identification
(d) The c's identification
(e) The d's identification
(f) The a's second identification

Figure 6: An example of the key-updating procedure.

tag from another one based on their outputs by performing passive or active attacks on the RFID system, we claim that A successfully compromises the privacy of the system. For simplicity, Let P denote the SPA authentication procedure.

We define four oracles, *Query, Send, Executive,* and *Reveal*, to abstract the attacks on each T or R. Thus, any attack on a given R or T can be represented as A's calling on its oracles.

Query (T, m_1, m_3). A sends a request m_1 to T. Subsequently, A receives a response from T. R then sends the message m_3 to T. Note that m_1 and m_3 represent the messages sent from A in the first round and third round in a SPA authentication procedure, respectively.

Send (R, m_2). A sends a message m_2 to R and receives a response. Note that m_2 represents the message sent from A in the second round in a SPA authentication procedure.

Execution (T, R). A executes an instance of P with T and R, respectively. A then modifies the response messages and relays them to both sides accordingly.

Reveal (T). After accessing this oracle of T, A compromises T, which means A obtains T's keys. Note that A can distinguish any given tag T from other tags if it can obtain T's keys.

We claim that an authentication protocol is resistant to attacks A-O, if the protocol is resistant to A when the adversary has access to the oracles $O \subset \{Query, Send, Executive, Reveal\}$. Based on these oracles, the detailed procedure of a game between A and C is formalized by following steps.

A tells C that the game begins. C chooses two tags T_0 and T_1.

For two tags T_0 and T_1 chosen by C, A accesses the oracles of T_0 and T_1. For T_0 and T_1, let O_{T_0} and O_{T_1} denote the sets of accessed oracles, respectively.

C selects a bit $b \in \{0, 1\}$ uniformly at random, and then provides the oracles of the corresponding tag T_b for A to access. For simplicity, we denote T_b as T. A then accesses T's oracle. Let the set of accessed oracles of T, be O_T.

Based on the results from O_{T_0}, O_{T_1} and O_T, A outputs a bit b'. If $b' = b$, A successfully distinguishes T_0 and T_1; otherwise, A loses. Note that A can access the oracles in O_{T_0}, O_{T_1}, and O_T in polynomial times. Indeed, if A can

distinguish T_0 from T_1, it means that A can track all tags in an RFID system.

A passive adversary who can only eavesdrop on the messages delivered by T or R has no access to any oracle. For an active adversary, it can access arbitrary oracles introduced above. For instance, if an adversary applies a cloning attack, it means that it can access the *Query*, *Send*, and *Executive* oracles in this attack model. If an adversary can apply compromising attack, it means that it can access the *Reveal* oracle.

4.3. Security Analysis. In this subsection, we show how SPA achieves the security goals.

Privacy. Due to the pseudorandomness and one-way properties of the cryptographic hash functions, it is safe to claim that the output of the hash function can be seen as a random bit string. Note that the pseudo-randomness of a hash function means no adversary can distinguish the output of the hash function from a real random bit string. Therefore, the messages sent by the reader and tags provide no useful information to an adversary. None of the passive adversaries is able to deduce the original messages based on the output of hash functions, unless it can invert the hash function. It is well known that the probability of inverting a hash function is negligible.

Untraceability. Since the authentication of SPA does not enroll the ID of a tag and all authentication messages are encrypted by a cryptographic hash function, any passive adversary cannot distinguish the tags from others based on their encrypted messages. That is, it cannot track a tag.

Cloning Resistance. In a cloning attack, an adversary monitors a tag, records its messages and resends the message repeatedly [9]. Similar to most previous protocols, in SPA, the reader and the tags generate random numbers r_1 and r_2 to defend against the cloning attack. Since the random numbers r_1 and r_2 are generated uniformly at random, the adversary has to guess them to recover the content of the messages. If the length of r_1 (r_2) is sufficiently long (e.g., 40 bits), the probability of successfully guessing the random numbers is negligible. Thus, SPA is not subject to the cloning attack.

Forward Secrecy. If a tag is captured, the adversary might obtain the tag's current keys. However, the adversary cannot trace back the tag's past communications because the keys are updated by a cryptographic hash function in each authentication procedure. In this way, the adversary still cannot retrieve the past outputs of the tag, unless it is able to invert the cryptographic hash function. Therefore, we can consider that the past-exposing probability of SPA upon the forward secrecy approaches 0. On the contrary, the static key tree protocols [7, 8, 11] cannot update the keys in the system. When an adversary compromised a tag, it can identify the past outputs of the tag from the obtained keys. Thus, the past-exposing probability of the key tree protocols approaches 1.

4.4. Compromising Attack. As we discussed in Section 3.1, a compromised tag may reveal some of the keys of other tags in static tree-based protocols. The adversary is then aware of some paths from the root to the leaf nodes of the compromised tag. Based on those paths, the adversary partially compromises the tree infrastructure. Knowing the "positions" of those nonleaf nodes, the adversary can further identify a sub-tree to which T_i might belong.

Now we use the attack model to discuss the impact of a compromising attack on SPA. The following analysis is based on Avoine's work [18]. The game procedure comprises six phases.

Phase 1. The adversary A has compromised t tags and obtained their complete secret keys.

Phase 2. The system (*challenger*) C chooses two normal tags T_0 and T_1.

Phase 3. A calls oracles O_{T_0} and O_{T_1} (except the *Reveal* oracle), and then receives the results (Note that A cannot compromise T_0 and T_1).

Phase 4. C selects a bit $b \in \{0, 1\}$ uniformly at random, and then provides the oracle O_T (denote T_b as T) to A for accessing (except the *Reveal* oracle).

Phase 5. A calls oracle O_T (except the *Reveal* oracle) and receives the results.

Phase 6. A outputs a bit b', if $b' = b$, A has successfully distinguished T_0 or T_1 from another. Otherwise, A loses.

We assume that A cannot carry out an exhaustive search over the key space. Suppose that A has compromised t tags except T_0 and T_1. Thus, A is aware of several paths from the root to the leaf nodes of those tags, as well as the relevant keys of the nonleaf nodes on those paths. Let M denote the set of those compromised nonleaf nodes in key tree. Let M_i denote the subset of M in which the nodes have the same level i in the key tree. Clearly, $M = \bigcup_{i=1}^{d} M_i$. Correspondingly, let $\overline{M_i}$ denote the set of nodes at level i which have not been compromised by A.

In Phase 5, A impersonates the reader and queries T, T_0, and T_1 with the keys obtained in the RFID system. At this point, there are three possible cases: (1) If neither T_0 nor T_1 has a nonleaf in M, A completely fails. (2) if either T_0 or T_1 (but not both) has a nonleaf node in M, the key subset of T_0 or T_1 including all the keys from the root to this nonleaf node have been compromised. Thus, the adversary can determine which one is T and obtain a correct answer in Phase 6. In this case, A succeeds. (3) If both T_0 and T_1 have an identical nonleaf node in M, A cannot directly distinguish T_0 or T_1 from another. In this case, A can move down to the lower level of the key tree from the current nonleaf node. We denote the keys of T, T_0, and T_1 by $[k_0, \ldots, k_d]$, $[k_0^0, \ldots, k_d^0]$, and $[k_0^1, \ldots, k_d^1]$, respectively, where d is the depth of the key tree. For a given level i, suppose two nodes $n_{i,0}$ and $n_{i,1}$ have an identical parent $n_{i-1,0}$ at the lever $i - 1$, and their keys are

k_i^0 and k_i^1, respectively. Let S_{i-1} denote the sub-tree of key tree S rooted at $n_{i-1,0}$. Thus, $n_{i,0}$ and $n_{i,1}$ are both in S_{i-1}. Let K_i denote the set of keys of the nodes in the interaction of $S_{i-1} \cap M_i$. Let V_i denote the set of the nodes in the interaction of $S_{i-1} \cap \overline{M_i}$. For example, suppose that R maintains a key tree with eight leaf nodes in Figure 7. A has compromised tags T_3, T_5, and T_8. In this case, for the sub-tree S_1, $K_i = \{k_{2,2}, k_{2,3}, k_{2,4}\}$ and $K_i = \{k_{2,1}\}$. Let t_i be the number of keys in K_i, and let δ be the branching factor of the key tree. We also denote a as the number of keys belonging to a nonleaf node (in SPA, any nonleaf node stores two keys k and tk, therefore $a = 2$). We consider the following five cases.

Case 1. If $C_i^1 = ((k_i^0 \in K_i) \wedge (k_i^1 \in V_i))$, then A succeeds.

Case 2. If $C_i^2 = ((k_i^0 \in V_i) \wedge (k_i^1 \in K_i))$, then A succeeds.

Case 3. If $C_i^3 = ((k_i^0 \in K_i) \wedge (k_i^1 \in K_i) \wedge (k_i^0 \neq k_i^1))$, then A succeeds.

Case 4. If $C_i^4 = ((k_i^0 \in V_i) \wedge (k_i^1 \in V_i))$, then A definitively fails.

Case 5. If $C_i^5 = ((k_i^0 \in K_i) \wedge (k_i^1 \in K_i) \wedge (k_i^0 = k_i^1))$, then A fails at level i but it can move to level $i+1$ to continue its attack.

For $1 \leq i \leq d$, we have

$$\Pr[C_i^1] = \Pr[C_i^2] = \frac{t_i}{a\delta}\left(1 - \frac{t_i}{a\delta}\right),$$
$$\Pr\left[C_i^3\right] = \left(\frac{t_i}{a\delta}\right)^2\left(1 - \frac{1}{t_i}\right), \tag{1}$$
$$\Pr\left[C_i^5\right] = \left(\frac{t_i}{a\delta}\right)^2 \cdot \frac{1}{t_i}.$$

Therefore, $\Pr[C_i^1 \vee C_i^2 \vee C_i^3] = (t_i/(a\delta)^2)(2a\delta - t_i - 1)$.

The probability that A succeeds is given by

$$\begin{aligned}
&\Pr[\text{Attack Succeeds}] \\
&= \Pr[C_1^1 \vee C_1^2 \vee C_1^3] \\
&\quad + \sum_{i=2}^{d}\left(\Pr\left[C_i^1 \vee C_i^2 \vee C_i^3\right] \times \prod_{j=1}^{i-1}\Pr\left[C_j^5\right]\right) \\
&= \frac{t_1}{(a\delta)^2}(2a\delta - t_1 - 1) \\
&\quad + \sum_{i=2}^{d}\left(\frac{t_i}{(a\delta)^2}(2a\delta - t_i - 1) \times \prod_{j=1}^{i-1}\frac{t_i}{(a\delta)^2}\right).
\end{aligned} \tag{2}$$

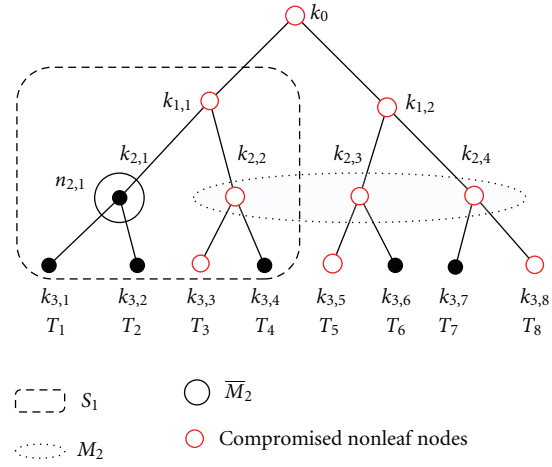


Figure 7: An example of the compromising attack.

In (2), t_i, the number of keys known by the adversary at level i, is given by

$$t_1 = \delta\left(1 - \left(1 - \frac{1}{a\delta}\right)^t\right),$$
$$t_i = \delta\left(1 - \left(1 - \frac{1}{a\delta}\right)^{f(t_i)}\right) \quad 1 < i \leq d, \tag{3}$$

where $f(t_i) = t\prod_{j=1}^{i-1} 1/t_j$.

Equation (2) shows that the correlated-exposing probability is mainly determined by three key parameters: (a) t, the number of compromised tags; (b) δ, the branching factor of the key tree; (c) a, the number of keys belonging to each nonleaf node. Note that if $a = 1$, (2) can also be used to evaluate the security of static tree-based approaches. In Figure 8, we show the theoretical evaluation on the security of SPA in a typical RFID system.

We assume that the system contains 2^{20} tags and the depth of key tree is 20. In the worst case, the adversary A can *simultaneously* compromise t tags at a given time. Then, A immediately starts attacks following the game strategy with challenger C. In addition, we assume there are only T_0 and T_1, which are chosen by C, performing authentication with the reader at this moment. Thus, we can use (2) to compute the correlated-exposing probability for A attacking SPA and static tree-based approaches.

As shown in Figure 8, SPA outperforms static tree-based approaches in defending against compromising attacks. In SPA, although A captures a number of keys shared by some normal tags, those tags are still secure if they update their keys. In contrast, normal tags in static tree-based approaches would be more vulnerable because the keys obtained by A will still be in use. This would ease A's tracking attempts.

In both SPA and static tree-based approaches, the correlated-exposing probability is reduced when enlarging the branching factor δ. This is because enlarging δ leads

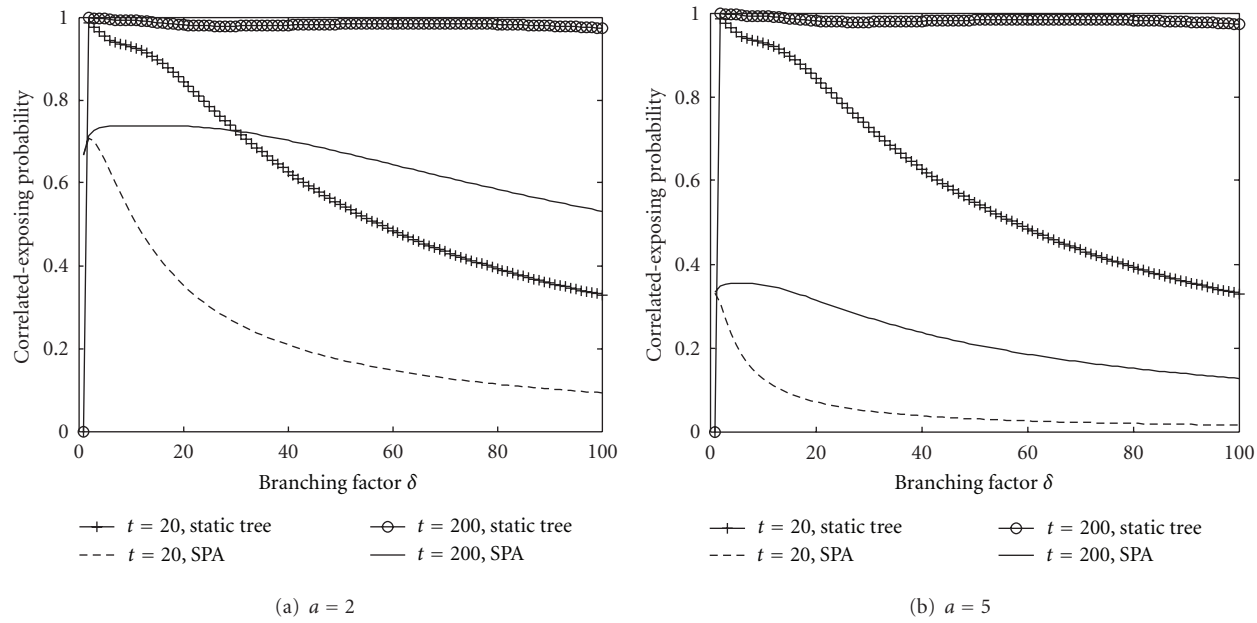


Figure 8: Defending against the compromising attack.

attackers to capturing fewer keys shared by tags which are not tampered with.

The static tree base approaches are extremely vulnerable to compromising attacks when t is sufficiently large. We find the correlated-exposing probability is close to 1 when $t = 200$ in static tree-based approaches. In this case, enlarging δ does not help much. On the contrary, SPA can decrease the probability by increasing a. The curves of $t = 200$ in Figure 8 show that SPA is more secure under compromising attacks and flexible enough to meet different security concerns.

We here explain why increasing a can enhance the security of SPA. In an RFID system, a part of tags may not be accessed by the reader for a long time. Keys in those tags hereby cannot be updated. The reader must maintain these keys in nonleaf nodes' temporal keys for future use. As discussed in Section 3.2, temporal keys tks store those old keys. The working key k is computed from the temporal keys after these temporal keys have been updated. Therefore, keys of some nonleaf nodes will not be updated if a number of tags are not identified by the reader for a long time. For example, we assume that each nonleaf node has only a single temporal key. If some tags lie in the sub-tree are not accessed by the reader in a period, other tags belong to this node's sub-tree can update keys for only once. That will make the key tree in SPA degenerate into the static tree, thereby alleviate SPA's resilience to compromising attacks. To ease the impact of this problem, we increase the number of temporal keys. We assume that each nonleaf node has one working key k and $a - 1$ temporal keys, $tk_1, \ldots, tk_{a-1}$. Thus, even if a number of tags are not accessed by the reader, keys of other tags in the system can be updated for at least $a - 1$ times. When we enlarge the parameter a, we can enhance the system's capability on resilience to compromising attacks, as shown in Figure 8.

5. Prototype Implementation

In this section, we introduce our experience on SPA prototype implementation. We also evaluate the performance of SPA and compare it with existing private authentication protocols.

5.1. Experiment Setup. We have implemented the SPA protocol on Mantis-series 303 MHz asset tags and Mantis II reader manufactured by RF Code [19]. In terms of its coding, this system is able to support over one trillion tags. A tags' typical transmission range is 300 feet, and the reader can communicate with them at distances of more than 1000 feet depending on the antenna configurations. The back-end database is implemented on the desktop PC with the following configurations: Pentium M 3.2 G dual core CPU, 1 GBytes memory, and 40 G hard disk. We use SHA-1 algorithm as the secure hash function.

In this implementation, the system is able to maintain up to $N = 2^{20}$ tags. For each test, we randomly distribute 40 tags into leaf nodes in the key tree. We perform 1000 independent runs and report the average. We employ a balanced binary tree as the key tree. Each nonleaf node is assigned with two keys, that is, $a = 2$. The length of each key is 64 bit, which is sufficiently long to resist brute-force attacks.

A fundamental concern upon SPA is the latency of key-updating. We use the metric *key-updating Latency* as the time required for the reader to update a tag's keys to evaluate the performance of SPA. On the other hand, the key-updating of SPA should guarantee that the keys are secure enough over their lifetime. We focus on two metrics in the experiments.

(a) Key-Updating Latency. It reflects the computational overhead of key-updating. In SPA, the whole processing

Table 1: Experiment settings.

Parameters	Values
δ	2
a	2
d	20
l	64
p	0.1
a_f	10

overhead for an authentication procedure includes two components: *identification* and *key-updating*. Since the static key tree approaches only perform the identification function, we focus more on the computational overhead caused by key-updating of SPA.

(b) Key Security Degree (KSD). It reflects the possibility of keys being obtained when an adversary attacks a RFID system. Let f denote the tag accessing frequency, which means how many tags interact with the reader per second. Let l be the length of a key, let n be the number of updated keys in one key-updating procedure, and let d be the depth of the key tree. We denote p as the probability of an adversary successfully gaining the n keys, and a_f as the attack frequency (i.e., the number of attacks occurred per second). Thus, the key security degree KSD is defined by

$$\text{KSD} = \frac{f \times n \times l \times d}{a_f \times p}. \tag{4}$$

Because the KSD computed from (4) will be a large value, we use a small real number ε to make the KSD value not too large, where $\varepsilon = 0.0001$ is a system parameter.

KSD reflects the comprehensive ability of a protocol on defending against the active attacks. A higher KSD value represents a more secure protocol. Parameters in our experiments are summarized in Table 1.

5.2. Results. Figure 9 plots the average key-updating latency of SPA. With the increase of the tag accessing frequency, which means how many times a tag is accessed per second, the key-updating latency increases. The processing speed of SHA-1 is 1.73 MByte per second. We find that the latency of key-updating does not exceed 1.7 ms even when the tag accessing frequency approaches 10. Since we construct a tree with the depth of 20 in this experiment, each tag is assigned with 20 keys. Thus, the curve of key-updating is enclosed within two lines: one represents the upper bound (20 keys in a tag are updated) and another represents the lower bound (only one key is updated). The short key-updating latency of SPA enables a reader to support dense access patterns. Due to page limitation, results from other experiments are not reported here.

Figure 10 shows the change of KSD of SPA with different tag access frequencies. The curve of KSD fluctuates when increasing the frequency. In Figure 10, two lines show the upper and lower bounds ($n = 20$ and $n = 1$) of the KSD

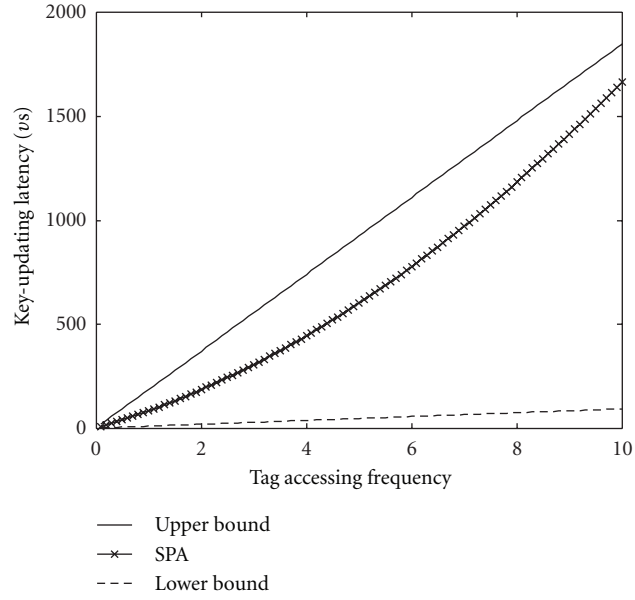


Figure 9: Key-updating latency of SPA.

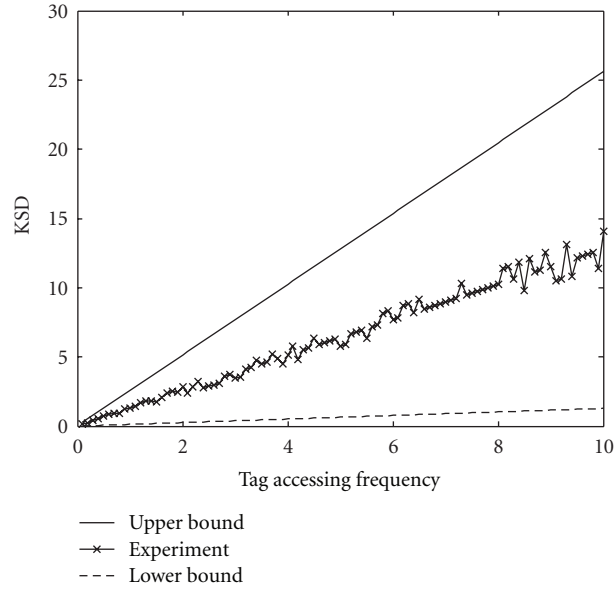


Figure 10: Key security degree of SPA.

curve of SPA. We can see that the KSD of SPA increases when enlarging the tag accessing frequency. Clearly, the design of SPA has two advantages. First, tag holders do not need to update the keys of their tags specially. Second, the RFID system is highly secure when tag holders use their tags frequently.

In SPA, the main overhead is caused by SHA-1 computations on the side of R. Therefore, the number of SHA-1 computations reflects the computation overhead of SPA. Figure 11 shows the relationship between the computation overhead and KSD in different tag accessing frequencies. In Figure 11, the y-axis refers to the number of

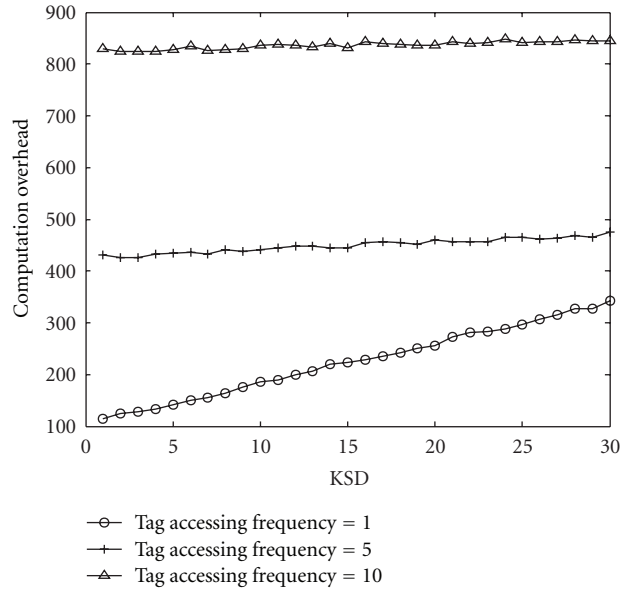


Figure 11: Computation overhead versus KSD.

SHA-1 computations. We see that a high tag access frequency results in a smooth curve of overhead. We also find that the computation overhead caused by SPA does not exceed 900 times of SHA-1 even the tag accessing frequency is high, for example, R accessing 10 tags per second, while the PC used in our experiments can perform 86,500 SHA-1 computations per second. It indicates that a significant incensement of KSD only incurs small computation overhead to the system. Hence, SPA is scalable when providing high secure private authentication services.

6. Conclusion

We proposed a privacy-preserving authentication protocol, SPA, to support secure and efficient tag-reader transactions in RFID systems. By using a dynamic key-updating algorithm, SPA enhances the security of existing RFID authentication protocols. SPA is lightweight with high authentication efficiency: a reader can identify a tag within $O(\log N)$ tree walking steps. Compared with previous works, SPA can effectively defend against both passive and active attacks. Through the prototype implementation, we demonstrated that SPA is scalable and practical in large-scale RFID systems.

Acknowledgment

This research was supported by NSFC 60903155 and NSFC 61173171.

References

[1] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "LANDMARC: indoor location sensing using active RFID," in *Proceedings of the1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pp. 407–415, March 2003.

[2] Y. Li and X. Ding, "Protecting RFID communications in supply chains," in *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS'07)*, pp. 234–241, March 2007.

[3] T. Kriplean, E. Welbourne, N. Khoussainova et al., "Physical access control for captured RFID data," *IEEE Pervasive Computing*, vol. 6, no. 4, pp. 48–55, 2007.

[4] C. Qian, H.-L. Ngan, and Y. Liu, "Cardinality estimation for large-scale RFID systems," in *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'08)*, pp. 30–39, March 2008.

[5] L. Xiao, Y. Liu, W. Gu, D. Xuan, and X. Liu, "Mutual anonymous overlay multicast," *Journal of Parallel and Distributed Computing*, vol. 66, no. 9, pp. 1205–1216, 2006.

[6] P. Robinson and M. Beigl, "Trust context spaces: an infrastructure for pervasive security in context-aware environments," in *Proceedings of International Conference on Security in Pervasive Computing (SPC'03)*, 2003.

[7] 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 (PerCom'06)*, pp. 269–274, March 2006.

[8] D. Molnar and D. Wagner, "Privacy and security in library RFID issues, practices, and architectures," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pp. 210–219, October 2004.

[9] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels, "Security and privacy aspects of low-cost radio frequency identification systems," in *Proceedings of International Conference on Security in Pervasive Computing (SPC'03)*, 2003.

[10] A. Juels, "RFID security and privacy: a research survey," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 381–394, 2006.

[11] D. Molnar, A. Soppera, and D. Wagner, "A scalable, delegatable pseudonym protocol enabling owner-ship transfer of RFID tags," in *Proceedings of the Selected Areas in Cryptography (SAC'05)*, 2005.

[12] M. Ohkubo, K. Suzuki, and S. Kinoshita, "Efficient hash-chain based RFID privacy protection scheme," in *Proceedings of the UbiComp, Workshop Privacy*, 2004.

[13] A. Juels, "Minimalist cryptography for low-cost RFID tags," in *Proceedings of International Conference on Security in Communication Networks (SCN'04)*, 2004.

[14] T. Dimitriou, "A lightweight RFID protocol to protect against traceability and cloning attacks," in *Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm'05)*, pp. 59–66, September 2005.

[15] G. Avoine and P. Oechslin, "A scalable and provably secure hash-based RFID protocol," in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom'05)*, pp. 110–114, March 2005.

[16] M. E. Hellman, "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406, 1980.

[17] G. Avoine, "Adversarial model for radio frequency identification," Report 2005/049, 2005, Cryptology ePrint Archive.

[18] G. Avoine, E. Dysli, and P. Oechslin, "Reducing time complexity in RFID systems," in *Proceedings of Selected Areas in Cryptography (SAC'05)*, 2005.

[19] RFCode, Inc., http://www.rfcode.com/Solutions/Asset-Management/Products-Overview.html.