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

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

[1]*State Key Laboratory of Information Security, Chinese Academy of Sciences*
[2]*Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology*
{luli,hu}@is.ac.cn, {jasonhan, liu, ni}@cse.ust.hk

Abstract

*The objective of private authentication for Radio Frequency Identification (RFID) systems is to allow valid readers to explicitly authenticate their dominated tags without leaking tags' private information. To achieve this goal, RFID tags issue encrypted authentication messages to the RFID reader, and the reader searches the key space to locate the tags. Due to the lack of efficient key updating algorithms, previous schemes are vulnerable to many active attacks, especially the compromising attack. In this paper, we propose a **S**trong and lightweight RFID **P**rivate **A**uthentication 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 observe that SPA is practical and scalable in current RFID infrastructures.*

1. Introduction

The proliferation of RFID applications [11] raises an emerging requirement – protecting user privacy [13] in RFID 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 can perform bogus authentication with detected tags to retrieve sensitive information. For example, without privacy protection, any reader can identify a consumer's ID via the emitted serial number from the tag. As a result, a buyer can be tracked and profiled by unauthorized people. In addition, many companies usually embed tags in items. Those tags indicate the unique information of the items to which they attach. Thus, a customer carrying those tags is subject to silent track from unauthorized readers. Some sensitive personal information would thereby be exposed: the illnesses she may suffer from indicated by the pharmaceutical products; the malls where she shops; the types of items she prefers to buy, and so on. Therefore, a secure RFID system must meet two requirements. On one hand, a valid reader must successfully identify the valid tags; on the other hand, misbehaving readers should not be able to retrieve 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. First, searching a key should be sufficiently fast to support a large scale system. Second, the keys should be updated dynamically for security concerns.

Many efforts have been made to achieve efficient private authentication. To the best of our knowledge, the most efficient protocols are tree-based [5, 10]. They provide an efficient key search scheme with *logarithm* complexity. In such 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 secure due to the lack of a key-updating mechanism. Most, if not all, tree-based approaches never update tags' keys dynamically. Since the key storage infrastructure of tree-based approaches is static, each tag, more or less, shares some common keys with others. Consequently, compromising one tag might 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. To our knowledge, consistent and dynamic key-updating mechanisms have scarcely been seen in the literature.

In this paper, we propose a **S**trong and lightweight RFID **P**rivate **A**uthentication protocol, SPA, which enables dynamic key-updating for tree based authentication approaches. Besides consistency, SPA also achieves forward 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. [14] 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 [8] classifies these approaches into three types.

Tree based approaches: tree based approaches [5, 9, 10] improve the key search efficiency from linear complexity to logarithmic complexity. Molnar et al. proposed the first tree-based scheme, which employs a challenge-response scheme [3], 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 [4], 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 [7], 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 [4], 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 [3]. In [6], Hellman studied the key-breaking problem and claimed that to recover a symmetric key k from a ciphertext needs to pre-compute 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 [5] 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 Fig. 1. When the reader R authenticates T_1, it 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 non-leaf 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 Fig. 1, any identification of a tag only needs $\log_2(8) = 3$ search steps.

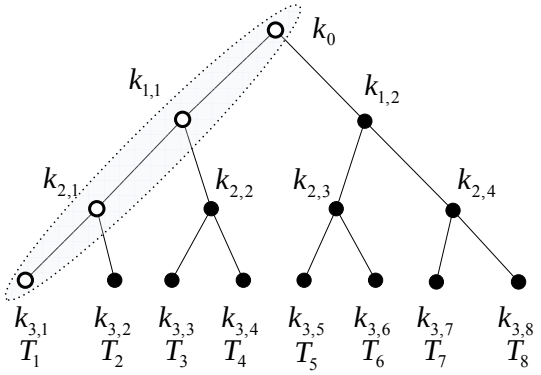


Figure 1. A binary key tree with eight tags.

If the adversary compromises some tags, however, it obtains several paths from the root to those leaf nodes of the compromised tags, as well as the keys on those paths. Since keys are never changed in the static tree architecture, the captured keys will still be used by uncompromised tags. As a result, the adversary captures the secret of uncompromised 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 uncompromised ones. However, the static tree architecture is not capable of solving the key-updating problem. Suppose we update the keys of T_1 in Fig. 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 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 redistributed. 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 the tags in the system (T_1, T_2, T_3, and T_4). One intuitive solution 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 need to collect a lot of tags even though there is only one tag updating its keys.

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 [5] 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 via two techniques: temporary keys and state bits. A temporary key is used to store the old key for each non-leaf node in the key tree. For each non-leaf node, a number of state bits are used in order to record the key-updating status of nodes in the sub-trees. Based on this design, each non-leaf 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 using the fourth component.

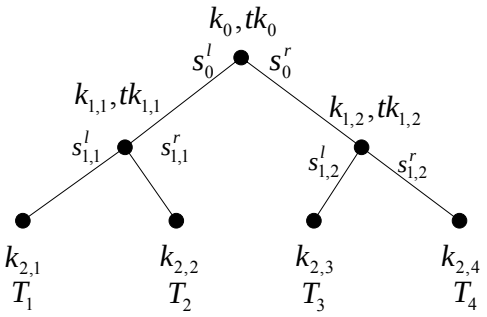


Figure 2. A key tree with four tags (*N* = 4).

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 Fig 2. Let δ denote the branching factor of the key tree (e.g., if the key tree is a binary tree, $\delta = 2$). 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 non-leaf 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 subsection 3.5. Initially, each key is generated randomly and independently by the reader, and $tk_j = k_j$ for all non-leaf 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 non-leaf 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 Fig. 2. From now on, 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 comprises three rounds, as illustrated in Fig. 3. In the first round, R starts the protocol by sending a "Request" and a random number r_1 (a nonce) to tag T_i, $1 \le i \le N$. In the second round, upon request, T_i generates a random number r_2 (a nonce) and computes the sequence $(h(k_0^i, r_1), \ldots, h(k_d^i, r_1))$, where $h(k, r)$ denotes the output of a hash function h on two inputs: a key k and a random number r. T_i replies R with a message $U = (r_2, h(k_0^i, r_1), \ldots, h(k_d^i, r_1))$. For simplicity, we denote the elements in U as $u, v_0, \ldots, v_d$. Upon U, R begins to identify T_i.

R executes the basic identification procedure to identify T_i, as represented Step 1 in Fig. 3. From the root, the reader first encrypts r_1 by using k_0, and compares the result with $h(k_0, r_1)$ from T_i. If they match, R invokes a recursive algorithm, Algorithm 1, as illustrated in Fig. 4 to identify T_i. For the example in Fig. 2, 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 $h(k_1^i, r_1)$, If $h(k_1^i, r_1)$ 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 extends the path of T_i originated from the root by invoking Algorithm 1. Suppose the path reaches an intermediate node j at level l $(1 \le l \le d)$ in the tree. At this point, R computes all hash values $h(k_{l+1}, r_1)$ and $h(tk_{l+1}, r_1)$ 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.

In short, 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.

3.5 Key-Updating

After successfully identifying T_i, R invokes the Key-updating algorithm in Step 2, as shown in Fig. 3.

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

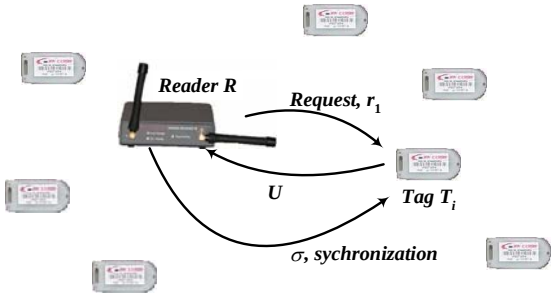


Figure 3. Authentication Procedure in SPA. After receiving *U*, Reader *R*'s operations are: Step 1, identifying T_i and Computes σ; Step 2, sending σ to T_i and key-updating. T_i also updates its keys after checking σ.

Algorithm 1: Identification (*U*, node *n*)

Fix $d \leftarrow \log N$;
SUCCEED $\leftarrow$ false;
$l \leftarrow \text{DepthofNode}(n)$;
if $(v_l = h(k_n, r_1) \vee v_l = h(tk_n, r_1))$
  if ($l \neq d$)
    if $v_l = h(tk_n, r_1)$
      Record *n* in Synchronization Message;
      for *i*=1 to δ
        $m \leftarrow \text{FindChildren}(n, i)$;
        Identification (*U, m*);
  else if *l*=*d*
    SUCCEED $\leftarrow$ true;
if (¬SUCCEED)
  fail and output 0;
Accept and output 1;

Figure 4. Tree-based identification.

Algorithm 2: Key-updating (node *n*)

if *n* is a non-leaf node
  Store the old key $tk_n \leftarrow k_n$;
Generate a new key $k_n \leftarrow h(k_n)$;
$m \leftarrow \text{FindParent}(n)$;
if *n* is the left child of *m*
  Set $s_m^l \leftarrow 1$;
else if *n* is the right child of *m*
  Set $s_m^r \leftarrow 1$;
if ($s_m^l = s_m^r = 1$)
  Reset s_m^l and s_m^r to 0, and record *m* in Synchronization message;
if *m* is not the root node
  $n \leftarrow m$;
Key-updating (*n*);

Figure 5. Tree-based key-updating.

$k_j' = h(k_j)$. The key-updating algorithm for the key tree is shown in Fig. 5. To remain consistent, the non-leaf 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 challenging 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 non-leaf nodes are shared by multiple tags. Those keys should be updated in a consistent manner. Second, each non-leaf 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 non-leaf node. The basic idea behind this mechanism is that each non-leaf 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 non-leaf 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 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 non-leaf 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.

When *R* finishes key-updating, it sends a message $\sigma = h(k_d^i, r_1, r_2)$, as shown in Fig. 3, 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. Having received 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 Fig. 2, suppose that *R* has updated keys $k_{1,1}$ and $k_{1,2}$ at level 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 non-leaf node to indicate the key-updating states of δ children.

3.6 System Maintenance

In practice, users might withdraw their tags. On the other hand, some tags of new users might be added. To deal with these maintenance issues, SPA provides the tag enrollment and withdrawal services.

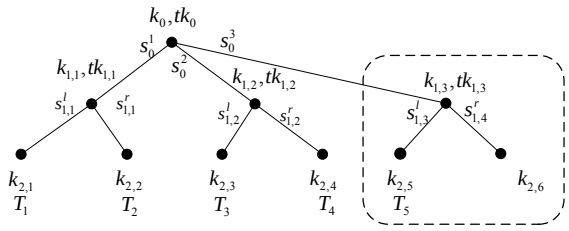


Figure 6. A new tag T_5 joins the RFID system.

If a new tag T_i joins the system, R starts the tag enrollment service. R first finds an empty leaf node in the key tree S and associates T_i with this node. T_i is accordingly assigned with the keys of nodes which are on the path from the root to the leaf node in S. If there is no an empty leaf node in 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, as we described in Section 3.3. After initialization, R grafts S' onto the root of S and S' becomes a sub-tree of S. T_i is then assigned to an empty leaf node in S' and T_i's keys are distributed according to the path from root of S to the leaf node. For example, in Fig. 6, R has 4 tags and all leaf nodes in S are occupied. If a new tag T_5 joins the system, R creates a new sub-tree marked with a dashed square. A leaf node in this sub-tree is associated with T_5. T_5's keys are k_0, $k_{1,3}$ and $k_{2,5}$. Indeed, increasing branching factor δ of the root of S incurs extra processes to the RFID system. For the example in Fig. 6, increasing δ of the root in a binary tree by one results in $N/2$ empty leaf nodes, while the added computation overhead is only one hash operation for node (1,3).

For any empty leaf node i in the key tree, i's parent node j will lock the corresponding state bit s_j as 1 until node i is assigned to a new tag T_i. The purpose of such constraint is to protect key-updating of other tags from being interrupted. Otherwise, if s_j is 0, it will never change such that node j will never update the keys.

If a tag is withdrawn, R empties the leaf node associated to this tag and sets the corresponding bit of the parent node to 1.

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 [5].

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 uncompromised 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; and (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 [2] provides an attack model for RFID systems. The model reflects the impacts of different attacks on the authentication protocols. Our discussions are mainly based on this model.

In the model, the attackers and the RFID system are abstracted into two participants: the *Adversary A* (the attackers) and the *Challenger C* (the RFID system). Attacking-defending between the attackers and the RFID system 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 one tag from another based on their outputs, we claim that A successfully compromises the privacy of the system. For simplicity, we let P denote the SPA authentication procedure.

We define four oracles, *Query, Send, Executive,* and *Reveal*, to model the attacks on each tag T and the reader R. Thus, each T or R has four such oracles in our model. Any attack on a given R or T can be represented as A's calling on one of its oracles as follows:

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 in the first and third round of SPA authentication procedure, respectively.

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

Execution(T, R): A acts as "a man in the middle" and executes an instance of P with T and R, respectively. A then modifies the received response message from one side and relays it to the other side.

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.

Based on these oracles, the detailed procedure of a game between A and C is demonstrated by the following steps.

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

2. 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.

3. C selects a bit $b \in \{0,1\}$ uniformly at random, and then provides the oracles of the corresponding tag T_b (if $b = 0$, $T_b = T_0$; otherwise, $T_b = T_1$) to A. For simplicity, we denote T_b as T. A then accesses T's oracles. Let O_T denote the set of accessed oracles of T.

4. 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 or T_1 from each other, hereby we say A succeeds and the protocol is broken; otherwise, A loses the game, which means the protocol is secure under A's attacks. In the model, we assume that A can access the oracles of O_{T_0}, O_{T_1} and O_T in polynomial times. Since T_0 and T_1 are randomly chosen from uncompromised tags, if A can distinguish T_0 from T_1 (or vice versa), it means that A can track all tags in an RFID system.

4.3 Security Analysis

In this subsection, we show how SPA meets the security requirements.

Privacy: The privacy is guaranteed by the security of the hash function used in SPA. Due to the pseudo-randomness and one-way properties of cryptographic hash functions, it is safe to claim that the output of the hash function can be seen as a random bit string. Therefore, the messages sent by the reader and tags will not reveal private information to any passive adversary. It is difficult, if not impossible, for passive adversaries to deduce the original messages based on the output of hash functions, unless they can break the hash function. It is well known that the probability of breaking a hash function is negligible.

Untraceability: SPA provides untraceability for tags. Since keys are dynamically updated in SPA, the encrypted messages of each tag are also changed accordingly. Thus, any passive adversary cannot track a tag by identifying the encrypted messages.

Cloning resistance: In a cloning attack, an adversary captures the messages from a tag and sends them to the reader repeatedly [5]. Similar to previous protocols, SPA employs 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 for each authentication procedure, it is extremely difficult for attackers to pre-determine them. In addition, the length of r_1 (r_2) in SPA is sufficiently long (more than 64 bits) such that the probability of successfully guessing a random number is negligible. Thus, SPA is not subject to the cloning attack.

Forward secrecy: If a tag is compromised, the adversary might obtain the tag's current keys. Since the keys stored in the tag are updated after each authentication procedure, the adversary cannot recover the past outputs of the compromised tag. Therefore, we can consider that the past-exposing probability of SPA approaches 0 and the forward secrecy of tags can be guaranteed. On the contrary, tags in the static tree protocols [5, 9, 10] never update their keys. Adversaries can easily recover the past outputs of compromised tags by using the obtained keys. Thus, the past-exposing probability of the static tree based 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 non-leaf nodes, the adversary can further identify a sub-tree to which T_i might belong.

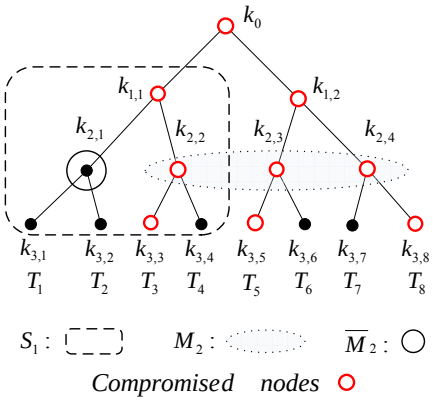


Figure 7. The compromising attack.

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 [3]. The game procedure comprises six phases.

Phase 1. *Adversary A* has compromised a number of tags and obtained their secret keys. Suppose the number of compromised tags is *t*. *A* will utilize the keys obtained from compromised tags in the attacks.

Phase 2. *Challenger C* chooses two tags T_0 and T_1. Note that T_0 and T_1 have not been compromised.

Phase 3. *A* calls oracles in O_{T_0} and O_{T_1} (except *Reveal* oracle), and then obtains 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 oracles in O_T (denote T_b as *T*) to *A* for accessing (except *Reveal* oracle).

Phase 5. *A* calls oracles in O_T (except *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 the other; otherwise, *A* loses.

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 non-leaf nodes in those paths. Let *M* denote the set of the compromised non-leaf nodes in the key tree. Let M_i denote the subset of *M* which includes the compromised nodes at the same level *i* in the 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 the key tree.

In Phase 5, *A* impersonates the reader and queries *T*, T_0 and T_1 with the keys obtained from compromised tags. As a result, there are three possible scenarios.

1) If neither T_0 nor T_1 has a non-leaf in *M*, *A* completely fails.

2) If either T_0 or T_1 (but not both) has a non-leaf node in *M*, the keys stored in this node as well as all the keys on the path from the root to this node have been compromised. The adversary can determine *T* in Phase 6. In this case, *A* succeeds.

3) If both T_0 and T_1 have an identical non-leaf node in *M*, *A* cannot directly distinguish T_0 or T_1 from the other. However, *A* can move down to the next lower level from the current non-leaf node in the key tree. We assume that the keys of *T*, T_0 and T_1 are $[k_0,\ldots,k_d]$, $[k_0^0,\ldots,k_d^0]$, and $[k_0^1,\ldots,k_d^1]$, respectively, where *d* is the depth of the tree. Suppose T_0 and T_1 share an identical node $n_{i-1,0}$ at lever $i-1$. At level *i*, T_0 has a node $n_{i,0}$ and T_1 has a node $n_{i,1}$. The keys of $n_{i,0}$ and $n_{i,1}$ are k_i^0 and k_i^1, respectively. Let S_{i-1} denote the sub-tree of the 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 $S_{i-1} \bigcap M_i$. Let U_i denote the set of the nodes in the interaction of $S_{i-1} \bigcap \overline{M}_i$. For example, suppose that *R* maintains a key tree with eight leaf nodes in Fig. 7. *A* has compromised tags T_3, T_5, and T_8. In this case, for sub-tree S_1, $\mathrm{K}_2 = \{k_{2,2}, k_{2,3}, k_{2,4}\}$ and $\mathrm{U}_2 = \{k_{2,1}\}$. Let t_i be the number of keys in K_i, and δ be the branching factor of the key tree. Let *a* denote the number of keys belonging to a non-leaf node (in SPA, any non-leaf node stores two keys *k* and *tk*, therefore $a = 2$). We consider the following five cases:

Case 1. If $C_i^1 = ((k_i^0 \in \mathrm{K}_i) \wedge (k_i^1 \in \mathrm{U}_i))$, *A* succeeds.

Case 2. If $C_i^2 = ((k_i^0 \in \mathrm{U}_i) \wedge (k_i^1 \in \mathrm{K}_i))$, *A* succeeds.

Case 3. If $C_i^3 = ((k_i^0 \in \mathrm{K}_i) \wedge (k_i^1 \in \mathrm{K}_i) \wedge (k_i^0 \neq k_i^1))$, *A* succeeds.

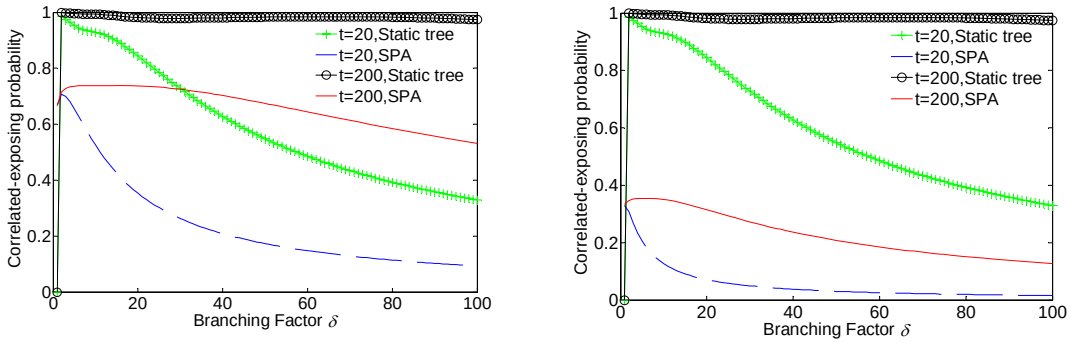
Case 4. If $C_i^4 = ((k_i^0 \in \mathrm{U}_i) \wedge (k_i^1 \in \mathrm{U}_i))$, *A* definitely fails.

Case 5. If $C_i^5 = ((k_i^0 \in \mathrm{K}_i) \wedge (k_i^1 \in \mathrm{K}_i) \wedge (k_i^0 = k_i^1))$, *A* fails at level *i* but it can move to level $i+1$ to continue its attack.

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

$$\Pr[C_i^1] = \Pr[C_i^2] = \frac{t_i}{a\delta}(1 - \frac{t_i}{a\delta}),$$

$$\Pr[C_i^3] = (\frac{t_i}{a\delta})^2 (1 - \frac{1}{t_i}), \text{ and } \Pr[C_i^5] = (\frac{t_i}{a\delta})^2 \cdot \frac{1}{t_i},$$



(a) $a = 2$ (b) $a = 5$

Figure 8. Defending against the compromising attack.

therefore, $\Pr[C_i^1 \vee C_i^2 \vee C_i^3] = \frac{t_i}{(a\delta)^2}(2a\delta - t_i - 1)$.

The correlated-exposing probability of A is given by:

$$\Pr[\text{Attack Succeeds}] = \Pr[C_1^1 \vee C_1^2 \vee C_1^3] + \sum_{i=2}^{d}(\Pr[C_i^1 \vee C_i^2 \vee C_i^3] \times \prod_{j=1}^{i-1}\Pr[C_j^5])$$

$$= \frac{t_1}{(a\delta)^2}(2a\delta - t_1 - 1) + \sum_{i=2}^{d}(\frac{t_i}{(a\delta)^2}(2a\delta - t_i - 1) \times \prod_{j=1}^{i-1}\frac{t_i}{(a\delta)^2}) \quad (1)$$

In Eq. (1), t_i, the number of keys known by the adversary at level i, is given by:

$$t_1 = \delta(1-(1-\frac{1}{a\delta})^t), t_i = \delta(1-(1-\frac{1}{a\delta})^{f(t_i)}),\ 1 < i \le d,$$

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

Equation (1) 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; and c) a, the number of keys belonging to each non-leaf node. Note that if $a = 1$, Equation (1) can also be used to evaluate the security of static tree based approaches. In Fig. 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 Eq. (1) to compute the correlated-exposing probability for A attacking SPA and static tree based approaches.

As shown in Fig 8, SPA outperforms static tree based approaches in defending against compromising attacks. In SPA, although A captures a number of keys shared by some uncompromised tags, those tags are still secure if they update their keys. In contrast, uncompromised 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 attackers to capturing fewer keys shared by uncompromised tags.

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 Fig. 8 show that SPA is more secure under compromising attacks and flexible enough to meet different security concerns.

5. Prototype Implementation

We have implemented the SPA protocol on 40 Mantis™-series 303 MHz asset tags and a Mantis™ II reader manufactured by RF Code [1]. The back-end database is implemented on a desktop PC with the following configurations: Pentium M 3.2G dual core CPU, 1GBytes memory, and 40G hard disk. We use the 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

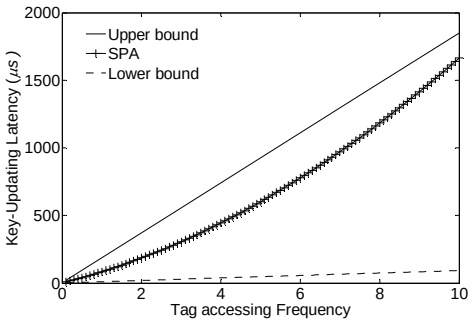


Figure 9. Key-updating latency of SPA.

tree. We perform 1000 independent runs and report the average. We employ a balanced binary tree as the key tree. Each non-leaf node is assigned with two keys, i.e., $a = 2$. 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.

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 1.7ms 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.

6. Conclusions

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.

Acknowledgements

This work is supported in part by the NSFC grant No. 60573053, the NSFC Key Project grant No. 60533110, the National Basic Research Program of China (973 Program) grant No. 2006CB303000, the Hong Kong RGC grants HKUST6152/06E and HKUST6183/06E, the Hong Kong RGC CAG grant HKBU 1/05C, and the HKUST Digital Life Research Center Grant.

References

[1] RF Code, Inc., http://www.rfcode.com/products.

[2] G. Avoine, "Adversarial Model for Radio Frequency Identification," Tech. Rep., 2005.

[3] G. Avoine, E. Dysli, and P. Oechslin, "Reducing Time Complexity in RFID Systems," in Proceedings SAC, 2005.

[4] T. Dimitriou, "A Lightweight RFID Protocol to Protect Against Traceability and Cloning Attacks," in Proceedings SecureComm, 2005.

[5] T. Dimitriou, "A Secure and Efficient RFID Protocol that Could make Big Brother (partially) Obsolete," in Proceedings IEEE PerCom, 2006.

[6] M. E. Hellman, "A Cryptanalytic Time-Memory Trade-off," IEEE Transactions on Information Theory, 1980.

[7] A. Juels, "Minimalist Cryptography for Low-Cost RFID Tags," in Proceedings SCN, 2004.

[8] A. Juels, "RFID Security and Privacy: a Research Survey," to appear in IEEE Journal of Selected Areas in Communication, 2006.

[9] D. Molnar, A. Soppera, and D. Wagner, "A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags," in Proceedings SAC, 2005.

[10] D. Molnar and D. Wagner, "Privacy and Security in Library RFID: Issues, Practices, and Architectures," in Proceedings ACM CCS, 2004.

[11] L. M. Ni, Y. Liu, Y. C. Lau, and A. Patil, "LANDMARC: Indoor Location Sensing Using Active RFID," in Proceedings IEEE PerCom, 2003.

[12] M. Ohkubo, K. Suzuki, and S. Kinoshita, "Efficient Hash-Chain based RFID Privacy Protection Scheme," in Proceedings UbiComp, Workshop Privacy, 2004.

[13] P. Robinson and M. Beigl, "Trust Context Spaces: an Infrastructure for Pervasive Security in Context-Aware Environments," in Proceedings SPC, 2003.

[14] S. Weis, S. Sarma, R. Rivest, and D. Engels, "Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems," in Proceedings SPC, 2003.