Ad Hoc & Sensor Wireless Networks, Vol. 24, pp. 135–159
Reprints available directly from the publisher
Photocopying permitted by license only

ACTION: Breaking the Privacy Barrier for RFID Systems

LI LU[1], YUNHAO LIU[2] AND JINSONG HAN[3]

[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, China E-mail: lulirui@gmail.com
[2]School of Software, Tsinghua University, China
E-mail: yunhaoliu@gmail.com
[3]Department of Computer Science and Technology, Xi'an Jiaotong University, China
E-mail: hanjinsong@mail.xjtu.edu.cn

Received: April 11, 2013. Accepted: May 16, 2014.

In order to protect privacy, Radio Frequency Identification (RFID) systems employ Privacy-Preserving Authentication (PPA) to allow valid readers to explicitly authenticate their dominated tags without leaking private information. Typically, an RF tag sends an encrypted message to the RF reader, then the reader searches for the key that can decrypt the ciphertext to identify the tag. Due to the large-scale deployment of today's RFID systems, the key search scheme for any PPA requires a short response time. Previous designs construct balanced-tree based key management structures to accelerate the search speed to $O(\log N)$, where N is the number of tags. Being efficient, such approaches are vulnerable to compromising attacks. By capturing a small number of tags, an adversary can identify other tags that have not been corrupted. To address this issue, we propose an Anti-Compromising authenticaTION protocol, ACTION, which employs sparse tree architecture, such that the key of every tag is independent from one another. The advantages of this design include: 1) resilience to the compromising attack, 2) reduction of key storage for tags from $O(logN)$ to $O(1)$, which is significant for resource critical tag devices, and 3) high search efficiency, which is $O(\log N)$, as good as the best in the previous designs.

Keywords: Authentication Privacy RFID Security

1 INTRODUCTION

Due to the low cost and easy deployment, Radio-Frequency Identification (RFID) has been an important enabling technology for everyday applications, such as retailing, medical-patient management, access control [1], logistics and supply chain management [2, 3]. In RFID systems, RF tags emit their unique serial numbers to RF readers. Without privacy protection, however, any reader can identify a tag ID via the emitted serial number. Indeed, within the scanning range, a malicious reader can easily perform bogus authentication with detected tags to retrieve sensitive information. Today, many companies embed tags into items. As these tags contain unique information about the items, a customer carrying those tags is subject to silent tracking from unauthorized readers. Sensitive personal information might be exposed: details about an illness inferred by the purchase of certain pharmaceutical products; the malls she shops at; the types of items she prefers to buy, and so on. Clearly, a secure RFID system must meet two requirements. On the one hand, a valid reader must be able to identify the valid tags; on the other hand, misbehaving readers should not be able to retrieve private information from those tags.

In order to protect user privacy, Privacy-Preserving Authentication (PPA) is introduced into the interactive procedure between RFID readers and tags [4]. To achieve PPA, an RFID tag performs a cryptography enabled challenging-response procedure with a reader [5]. For example, we can let each tag share a distinct key with the reader. During authentication, the reader first probes a tag via a query message with a nonce. Instead of using plaintext to directly answer the query, the tag encrypts the nonce and sends the ciphertext back to the reader. The back-end database of the reader searches all the keys that it holds, and, if possible, finds a proper key to recover the authentication message, and thereby identifying the tag. (For simplicity, we use the term "reader" to denote the reader device as well as the back-end database in the following). If a tag is invalid, it cannot provide a proper ciphertext related to a key owned by the reader. In this procedure, the tag does not expose its identity to any third party. Meanwhile, the key used for encrypting messages is only known by valid readers. A malicious reader cannot identify a user via probing the valid tag.

As it is simple and secure, such a PPA based design suffers poor scalability. Upon receiving a nonce ciphertext, the reader needs a prompt lookup to locate a key in the database. Clearly, the search complexity is $O(N)$, where N is the number of all the possible tags, even only a small portion of them are in the reader's range. In today's large-scale RFID systems, N is often as large as hundreds of millions, and thousands of tags may respond to a reader simultaneously, demanding a fast key-search method as well as a carefully designed key-storage structure. Hence, balanced-tree based schemes [6-9] are proposed to accelerate the authentication procedure, in which the lookup complexity is $O(\log N)$.

The balanced-tree based approaches are efficient, nevertheless, not secure due to their key-sharing feature. As the key storage infrastructure of those approaches is static, each tag, more or less, shares some common keys with other tags (in this paper, we use *normal* tags to denote tags that are not tampered with). Consequently, compromising one tag might reveal information of other tags [6, 9]. L. Lu *et al.* evaluate the damage caused by compromising attacks to balanced-tree based approaches [9]. In an RFID system containing 2^{20} tags, and employing a binary tree as the key tree, an adversary, by compromising only 20 tags, has a probability of nearly 100% of being able to track normal tags [10].

To mitigate the impact of compromising attacks, L. Lu *et al.* propose a dynamic key-updating scheme [9], SPA, for balanced-tree base approaches. The key-updating of SPA reduces the number of keys shared among compromised and normal tags, and alleviates the damage caused by compromising attacks. SPA, however, does not completely eliminate the impact of compromising attacks. For instance, using SPA in an RFID system with 2^{20} tags, the probability of tracking normal tags is close to 60% after an adversary compromises 20 tags [9].

Another drawback for balanced-tree based PPAs is the large space needed to store keys in each tag. Balanced-tree based approaches require each tag to hold $O(\log_\delta N)$ keys, and the reader to store $\delta \times N$ keys, where δ is a branching factor of the key tree. Obviously, due to the limited memory capacity of RF tags, existing PPAs are difficult to apply in current RFID systems.

To address the above issues, we propose an Anti-Compromising authentica-TION protocol, called ACTION. By employing a sparse tree to organize keys, ACTION generates completely independent keys for tags, so that compromised tags have no keys that correlate with the normal ones. As a result, ACTION can effectively defend against compromising attacks. We show that if an adversary can track a normal tag with a probability larger than α, it must tamper with more than $N - 1/\alpha$ tags, while in previous balanced-tree based approaches, by compromising $O(\log N)$ tags, an adversary can track a normal tag with a probability more than 90% [10]. Another salient feature of this design is the low storage requirement for tags. ACTION only allows each tag to store two keys and the reader to store $O(N)$ keys, achieving high storage efficiency for both readers and tags, making this design practical for today's RF tags. We also show that ACTION retains high search efficiency in the sense that the lookup complexity is still $O(\log N)$, as good as the best of previous designs.

The rest of this paper is organized as follows. We discuss the related work in Section 2. We present the ACTION protocol in Section 3. In Section 4, we discuss the storage and search efficiency of ACTION. We present the security analysis in Section 5, and conclude the work in Section 6.

2 RELATED WORK

The fundamental principle of PPAs is based on HashLock [5], in which every tag shares a unique key with the reader. The tag and reader use a challenging-response scheme to conduct authentication. Recent studies [13] show that HashLock is a secure PPA. The main drawback of HashLock is that the key search is linear to the number of tags in the system, which limits the usage of HashLock in large-scale RFID systems. Subsequent approaches in the literature are mostly aimed at improving the efficiency of key search. Juels [14] classifies those approaches into three categories.

Synchronization approaches: Such approaches [15-18] use an incremental counter to record the state of authentication. When an authentication is successfully performed, the tag increases the counter by one. The reader compares the value of a tag's counter with the record in the database. If the difference of the two counter values is in a proper window, the tag is viewed as valid and the reader synchronizes the counter record of the tag. Synchronization schemes are subject to the *Desynchronization Attack* [14], in which a malicious reader interrogates a tag many times such that the counter of the tag exceeds the range of the window and the reader fails to recognize a valid tag.

Time-space tradeoff approaches: AO [19] employs Hellman tables to improve the key-efficiency. Hellman [20] studies the problem of breaking symmetric keys and shows that an adversary can pre-compute a Hellman table of storage size $O(N^{2/3})$, in which the adversary can search a key with the complexity of $O(N^{2/3})$. That means the key-searching efficiency of OSK or AO is also $O(N^{2/3})$. Those approaches are not sufficiently efficient for supporting large-scale RFID systems.

Balanced-tree based approaches: Balanced-tree based approaches [6-9] improve the key search efficiency from linear complexity to logarithmic complexity. They employ a balanced-tree to organize and store keys for tags. In a balanced-tree, each node stores a unique key. Keys in the path from the root to a leaf node are distributed to a tag. Each tag uses these multiple keys to encrypt the identification message. Upon receiving an encrypted message, the reader performs a Depth-First Search on the key tree with a logarithmic complexity of the system size. The balanced-tree based approaches, however, are subject to *Compromising Attack* [6, 9]. In a balanced-tree, tags always share keys with others. Hence, hacking one tag may reveal several keys used by other tags. For example, in a binary balanced-tree based RFID system containing 2^{20} tags, an adversary can identify any tag with the probability of about 90% by tampering with only 20 tags [9, 10]. To address a compromising attack, L. Lu *et al.* propose a dynamic key-updating scheme, SPA [9], for enhancing balanced-tree based approaches. In the scheme, after successfully identifying a tag, the reader dynamically and recursively updates keys in the key tree and coordinates the keys with the tag. The key-updating scheme reduces the probability of locating a tag via compromising attacks. However, the threat from compromising attacks has not been completely relieved. For instance, in a SPA system containing 2^{20} tags, a compromising adversary still can recognize any normal tag with a high probability (about 60%) after it tampers with 20 tags [9].

3 ACTION DESIGN

In this section, we first discuss the motivation of this work, and then present the details of the ACTION protocol.

3.1 Motivation

In previous balanced-tree based approaches, initially, a reader organizes a hierarchical balanced-tree with a depth of $\log_\delta N$ (δ is branching factor), in which each node is assigned a unique key. The reader then monogamously maps N leaf nodes to N tags. Figure 1 plots a balanced-tree for 8 tags. For each tag, there is a unique shortest path from the root to the corresponding leaf node. For example, in Fig. 1, tag 3 obtains $k_{1,1}$, $k_{2,2}$, and $k_{3,3}$. During authentication, upon receiving a request with a nonce r from the reader, T_3 encrypts r in the way $\{k_{1,1}\{r\}, k_{2,2}\{r\}, k_{3,3}\{r\}\}$ and sends the ciphertexts to the reader. Upon receiving the response from T_3, the reader searches proper keys in the key tree to recover r. This is equal to exploring a path from the root to the leaf node of T_3 in the tree. At the end of identification, if such a path exists, R regards T_3 as a valid tag. Clearly, the search complexity is $O(\log N)$.

The fundamental nature of balanced-tree based PPAs is that a tag shares some non-leaf nodes, more or less, with other tags in the key tree. This is a fatal flaw when balanced-tree based PPAs are under compromising attacks. For example, in Fig. 1, we can see that a common key, $k_{1,1}$, is shared by tags T_1, T_2, T_3, and T_4, and $k_{2,2}$ is shared by T_3 and T_4. If an adversary compromises T_3 and reveals the keys stored in T_3, the keys $k_{1,1}$ and $k_{2,2}$ are also exposed. As a result, even though T_4 is not cracked, the adversary can easily distinguish T_4 via $k_{1,1}$ and $k_{2,2}$. Even worse, the adversary can actually distinguish each normal tag by only compromising a small fraction of all tags.

Based on the above analysis, it is clear that the only solution to compromising attacks is to eliminate the correlation among the keys of different tags. Therefore, in this design, we intend to remove all correlations among the keys. The difficulty is that we cannot sacrifice the search efficiency as well as the storage efficiency.

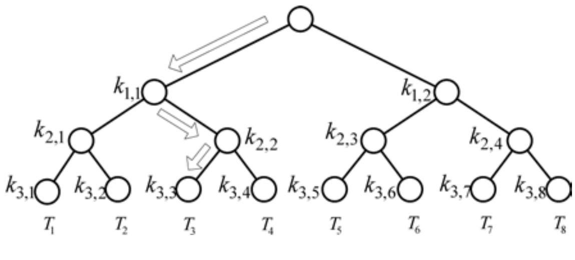


FIGURE 1
An example of key organization in balanced-tree based PPAs.

3.2 Overview

ACTION mainly has four components: *system initialization*, *tag identification*, *key-updating*, and *system maintenance*. In the first component, instead of using a balanced-tree, we employ a sparse tree to organize keys for tags. In the extreme case, the sparse tree can support 2^{128} tags. We generate two random keys (128 bits), denoted as path key k^p and leaf key k^l, to a tag and a corresponding path in the sparse tree according to k^p value. After the key initialization, each tag is associated with a leaf node in the tree. The leaf node thereby represents the key k^l assigned to the tag, and the path from the root to the leaf node indicates the key k^p. Since the two keys are randomly generated, keys among different tags are independent. In the second component, the reader performs a logarithmic search to identify a tag. In the third component, ACTION performs a key-updating procedure, in which ACTION employs a cryptographic hash function, such as MD5, SHA-1, to update the old key in a tag. Note that the new key is still random and independent of the keys used by other tags. ACTION also reduces the maintenance overheads in highly dynamic systems where tags join or leave frequently by using the fourth component.

3.3 System initialization

We assume that there are N tags T_i, $1 \leq i \leq N$, and a reader R in the RFID system. We denote the sparse tree used in ACTION as S. Let δ denote the branching factor of the key tree and d denote the depth of the tree. Each tag is associated with a leaf node in S. The secret keys shared by tag T_i and reader R are denoted as k_i^p and k_i^l. Let n be the length of k_i^p and k_i^l, i.e. $|k_i^p| = |k_i^l| = n$. We split k_i^p into d parts, that is, $k_i^p = k_i^p[0]|k_i^p[1]|\ldots|k_i^p[d\text{-}1]$, and the length of each $k_i^p[m]$ is n/d, $m = 0...d\text{-}1$. We set the branching factor, δ, of each non-leaf node in S as $2^{n/d}$, namely $d \times \log\delta = n$. For example, if we set the key length as 128 bits and $d = 32$, the branching factor of the S is $\delta = 2^{128/32} = 2^4 = 16$. In other words, each non-leaf node is able to accommodate 16 child positions in S. If the c-th child node exists in a child position of a non-leaf node j, we set c as the *index number* of this child and record c in j. Note that a non-leaf node *only* stores the index numbers for existing children.

For simplicity, we denote the set of j's index numbers as IS_j, and the element number of IS_j as IN_j, that is, $IN_j = |IS_j|$. We show an example in Fig. 2, in which the branching factor $\delta = 2^4$. Each non-leaf node has 16 child positions. For a non-leaf node a, as shown in Fig. 2, the reader maintains its' index number set as $IS_a = \{5, 7\}$, and the IN_a=2.

Initially, the tree is empty. Reader R generates two keys k_i^p and k_i^l uniformly at random for every tag T_i. Meanwhile, the reader divides each k_i^p into d parts, $k_i^p[0]|k_i^p[1]|\ldots|k_i^p[d\text{-}1]$, where d is the depth of key tree S. The reader distributes k_i^p and k_i^l to tag T_i and organizes k_i^p into S as follows. From the root, the reader generates a non-leaf node at each level m according to the corresponding $k_i^p[m]$. That is, after the reader generates a node a at the level m-1 according to the $k_i^p[m\text{-}1]$, it will generate the $k_i^p[m]$-th child of node a, and set

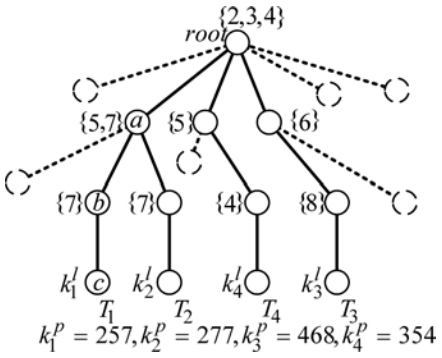


FIGURE 2
A key tree with four tags ($N = 4$).

an index number of a as $k_i^p[m]$. For example, as shown in Fig. 2, the branching factor δ of S is 16, and there are 4 tags in the system, denoted as T_1, T_2 , T_3, and T_4. Assume that the length of path key is 12 bits. Each path key is divided into 3 parts, and the length of each part is 4 bits (because $\delta = 16$, the length of each part of a key should be $\log_2 16 = 4$ bits). The reader generates four path keys as 257, 277, 468, and 354 for tags T_1-T_4, respectively. The reader also generates four leaf keys as k_1^l, k_2^l, k_3^l, and k_4^l for T_1-T_4, respectively. For T_1, $k_1^p = 257$ (0010|0101|0111), thus, $k_1^p[0] = 2$, $k_1^p[1] = 5$, and $k_1^p[2] = 7$. The reader first generates a child at the root, and sets an index number as 2 ($k_1^p[0] = 2$). Here the index number 2 means the root has a child marked as node a in its second position, as illustrated in Fig. 2. Then the reader generates a child b of node a, and sets an index number of a as 5 ($k_1^p[1] = 5$). Finally, the reader generates a child c of node b, which is a leaf node c, and sets an index number of b as 7 ($k_1^p[2] = 7$). Indeed, the key organization can be analogous to generate a path in tree S. In the above example, the path of T_1 is $root \rightarrow a \rightarrow b \rightarrow c$. After the same procedures on tags T_2, T_3, and T_4, we obtain a sparse tree as illustrated in Fig. 2. The procedure is described as Algorithm 1 TagJoin.

Algorithm 1: TagJoin (Tag *T*, Key Tree *S*)

```
1:  $k^p$, $k^l$ ← KeyGeneration($T$);
2:  ($k^p$[0],..., $k^p$[d-1]) ← KeyDivision($k^p$);
3:  Node ← GetRoot($S$);
4:  for $i = 0$ to $d - 1$
5:      Add $k^p[i]$into $Node$'s Index Set $IS$;
6:      if the $k^p[i]$-th child does not exist
7:          Create the $k^p[i]$-th child;
8:          $Node$ ← the $k^p[i]$-th child;
9:  else $Node$ ← the $k^p[i]$-th child;
```

3.4 Tag identification

ACTION employs cryptographic hash functions to generate authentication messages and update keys. Let h denote a cryptographic hash function: $h:\{0,1\}^* \rightarrow \{0,1\}^n$, where n denotes the length of the hash value. Let N be the number of all tags in the system. The basic authentication procedure between the reader and a tag T_i ($1 \leq i \leq N$) includes three phases, as illustrated in Fig. 3. In the first phase, the reader R sends a "Request" with a random number r_1 (a nonce) to tag T_i. In the second phase, upon receiving "Request", tag T_i generates a random number r_2 (a nonce) and calculates a series of hash values, $h(r_1, r_2, k_i^p[0])$, $h(r_1, r_2, k_i^p[1])$, ..., $h(r_1, r_2, k_i^p[d\text{-}1])$, $h(r_1, r_2, k_i^l)$, where $h(r_1, r_2, k)$ denotes the output of the hash function on three inputs: a key k and two random numbers r_1 and r_2. T_i replies R with a message $U = (r_2, h(r_1, r_2, k_i^p[0]), h(r_1, r_2, k_i^p[1]), ..., h(r_1, r_2, k_i^p[d\text{-}1]), h(r_1, r_2, k_i^l))$. For simplicity, we denote the elements in U as u, v_0, v_1, …, $v_{d\text{-}1}$, v_d where $u = r_2$ and $v_j = h(r_1, r_2, k_i^j)$, $j = 0...d\text{-}1$, $v_d = h(r_1, r_2, k_i^l)$. In the third phase, R identifies T_i using the key tree S and the received U.

Algorithm 2: Identification (*U*, node *X*)

```
1:  SUCCEED ← $false$;
2:  $m$ ← DepthOfNode($X$);
3:  $IS$ ← GetIndexSet($X$);
4:  $IN$ ← $|IS|$;
5:  if $m \neq d$
6:     for $i = 1$ to $IN$
7:        if $v_m = h(r_1, r_2, i) \wedge i \in IS$
8:           $Y$ ← GetChild($X$,$i$);
9:           Identification ($U$, Y);
10:     else if $m = d \wedge h(r_1, r_2, k_1) = v_d$
11:              SUCCEED ← $true$;
12:   if (SUCCEED = $false$)
13:      Fail and output 0;
14:  Accept and output 1;
```

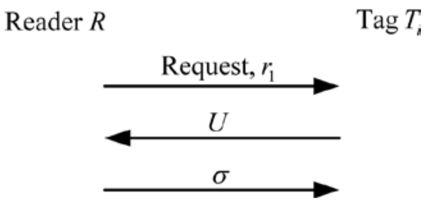


FIGURE 3
The authentication procedure of ACTION.

Reader R invokes a recursive algorithm to probe a path from the root to a leaf in S to identify T_i, as shown in Algorithm 2. Assume R reaches a non-leaf node a at level m-1. For all index numbers stored in a, R computes a hash value with inputs as r_1, r_2, as well as the index numbers, and then compares the hash value with the element v_m in the received U. If there is a match, the path of T_i should be extended to the child related to the index number. Note that here the child node is on the path assigned to T_i. Repeating such a procedure until arriving at a leaf node, R recognizes the tag T_i. For the example in Fig. 2, upon receiving a "Request" message with a random r_1, T_1 generates a random number r_2, and computes a series of hash values $h(r_1, r_2, 2)$, $h(r_1, r_2, 5)$, $h(r_1, r_2, 7)$, and $h(r_1, r_2, {k_1}^l)$, then replies R with the message $U = (u, v_0, v_1, v_2) = (r_2, h(r_1, r_2, 2), h(r_1, r_2, 5), h(r_1, r_2, 7), h(r_1, r_2, {k_1}^l))$. Upon receiving U, R first compute all $h(r_1, r_2, x)$ to compare with v_1. Here $x = 2$, 5, and 7, which are all the index numbers stored in the root. Clearly, R locates 2 as a match number and thereby moves to node a. Then R locates 5 and 7 in the nodes b and node c, respectively. R terminates its path probing when it reaches the leaf node c, thereby identifying T_1.

Algorithm 3: TagLeave (Tag *T*, Key Tree *S*)

```
1:  $k^p, k^l \leftarrow$ GetKey($T$);
2:  ($k^p[0],\ldots, k^p[d$-1]) $\leftarrow$ KeyDivision($k^p$);
3:  Node $\leftarrow$ GetLeaf($T$);\\ Get the corresponding leaf of $T$
4:  for $i = d - 1$ to 0
5:      if Node doesn't have brothers
6:          TempNode $\leftarrow$ Node;
7:          Node $\leftarrow$ FindParent(TempNode);
8:          Delete the $ki$ from the Index Set $IS$ of Node;
9:          Delete TempNode;
10:     else Node $\leftarrow$ FindParent(Node);
```

3.5 Key-updating

After successfully identifying T_i, R and T_i automatically update the key stored in T_i and coordinate the changes to the tree S as follows.

Reader R makes use of a cryptographic hash function h to generate new keys. Let k_i^p and k_i^l be the current path key and leaf key used by T_i. Reader R computes a new path key $\overline{k_i^p}$ from the old path key k_i^p and leaf key k_i^l by computing $\overline{k_i^p} = h(r_1, r_2, k_i^p, k_i^l)$. Similarly, R calculates the new leaf key as $\overline{k_i^l} = h(r_1, r_2, k_i^l)$. The challenging issue here is that we need to carefully modify the index numbers of non-leaf nodes according to the new key $\overline{k_i^p}$. Otherwise, some tag identifications can be interrupted, since the index number stored in non-leaf nodes might be shared among multiple tags.

To address the challenge, we design two algorithms for key-updating: 1) TagJoin, as shown in Algorithm 2; and 2) TagLeave, shown in Algorithm 3. The basic idea is that we first use the TagLeave to remove the path corresponding to old path key k_i^p of tag T_i, and then generate a new path corresponding to key $\overline{k_i^p}$ in S. It is possible that a non-leaf node in the path has multiple branches so that some keys are used by other tags, for example, node a in Fig. 2. In this case, the TagLeave algorithm terminates.

After deleting the old key, R re-generates a new path for tag T_i according to the new key $\overline{k_i^p}$ using the TagJoin algorithm. A potential problem of new path generation is that the path has existed in S, which means the key $\overline{k_i^p}$ has been generated in the system. The probability of this situation happening is quite small. First, the sparse tree is a virtual tree according to the initialization algorithm. Prior to the tag deployment, the tree is empty. When a path key is generated by a hash function, a path from a certain leaf to the root emerges accordingly in the sparse tree. Therefore, a path in the sparse tree corresponds to a hash value. This correspondence leads to two facts: 1) the capability of a sparse tree is as large as the size of the hash value space. In our work, a path key is a hash value with a length of 128 bits, which indicates the sparse tree can hold 2^{128} paths maximum, that is, the sparse tree can hold 2^{128} tags correspondingly. In any practice RFID system, however, the number of tags is much less than 2^{128}. The probability of the tree becoming dense is negligible. 2) A path in the sparse tree corresponds to a hash value. Therefore, if two tags have the same path in the sparse tree, this means a hash collision appears. According to the collision-resistance property of hash functions, the probability of a collision happening is also negligible. For example, an RFID system contains 2^{20} tags, and the length of a path key is 128 bits. The ratio of occupied paths in the sparse tree is 2^{-88} ($2^{20}/2^{128}$), and the path key is generated uniformly at random. Thus, the probability of generating an existing path is 2^{-88}. Summarizing the above analysis, it is safe to claim that the probability of two tags having a similar path is negligible.

If such a collision does happen, in this design, R first generates a new key $\overline{k_i^p}^{2} = h(r_1, r_2, \overline{k_i^p}, k_i^l)$, and then executes the TagJoin algorithm again to create a new path in S. R repeats such a procedure until a new path is successfully generated. R counts the number of TagJoin runs, denoted as s (due to the negligible probability of collisions, s usually equals to 1), and sends a synchronization message $\sigma = (s, h(r_1, r_2, \overline{k_i^p}^{s}), h(r_1, r_2, \overline{k_i^l}))$ to tag T_i, as shown in Fig. 3. Here $\overline{k_i^p}^{s}$ is computed from iterative equations by:

$$\begin{cases} \overline{k_i^p}^{1} = k_i^p \\ \overline{k_i^p}^{s} = h(r_1, r_2, \overline{k_i^p}^{s-1}, k_i^l) \end{cases} \tag{1}$$

Having σ, T_i first computes $\overline{k_i^p}^s$ using k_i^p and s with (1), then computes $\overline{k_i^l}$ = $h(r_1, r_2, k_i^l)$. Thus T_i gets σ' = (s, $h(r_1, r_2, \overline{k_i^p}^s)$, $h(r_1, r_2, \overline{k_i^l})$). After computing σ and σ', T_i verifies whether or not σ = σ'. If yes, T_i updates its keys as $\overline{k_i^p}^s$ and $\overline{k_i^l}$. Otherwise T_i returns an error to the user and will not update keys stored on it; by doing that, T_i can coordinate its key with the one generated by the reader.

3.6 System maintenance

This component is mainly for tag joining and leaving.

If a new tag T_i joins the system, R needs to find a new path in the key tree. R invokes the TagJoin algorithm, as shown in Algorithm 1. Specifically, R generates a new path key k_i^p and leaf key k_i^l independent of other keys, then splits k_i^p into d parts, $k_i^p[0]$, $k_i^p[1]$,…, $k_i^p[d\text{-}1]$. Starting at the root, if R arrives at a non-leaf node j at level m, R adds $k_i^p[m]$ into j's index number set IS_j, and walks to the $k_i^p[m]$-th child of j (if this child does not exist, R creates it). When a leaf node is reached, R associates T_i to the leaf node, and sets the key of the leaf node as k_i^l. A new path is generated for T_i.

To withdraw a tag T_i, R should erase the path from the root to T_i's associated leaf node, using the TagLeave algorithm. In this algorithm, R first deletes the leaf key k_i^l of T_i. Starting from the associated leaf node of T_i, if R reaches a node e at level m, R first finds e's parent f, and then deletes $k_i^p[m]$ from the index set IS_f. After arriving at node f, R deletes e. R repeats this procedure until a non-leaf node in the path has multiple branches, for example, node a in Fig. 2. Thus, R withdraws T_i.

4 EFFICIENCY

We first investigate the storage efficiency of ACTION, and then analyze the identification efficiency by estimating the necessary number of hash computations. We also discuss the lower and upper bounds of ACTION's identification efficiency.

4.1 Storage

An RFID tag normally has a very tiny memory for storing user information as well as the keys. Hence, storage efficiency must be taken into account in designing secure PPA protocols.

In balanced-tree based approaches, each tag is allocated multiple keys, which incur a relatively large storage overhead. ACTION is more efficient in the key storage on both the tag and reader sides. Specifically, ACTION allocates each tag only two keys, a path key and a leaf key, and requires the reader to store the keys for each tag. Each path key is divided into several fractions, which are stored in the non-leaf nodes' index sets, respectively. Thus, the storage at readers is 2*N*. In contrast, balanced-tree based approaches distribute *O*(logδ*N*) keys to each tag, and maintain δ × *N* keys on the reader side, where δ is the branching factor of the balance key tree. Clearly, ACTION is more practical for current RFID systems.

4.2 Identification efficiency

The basic operations in a PPA authentication are mainly hash computations and comparisons. The numbers of these two operations are equal, because each hash computation is followed by a comparison of hash values. Hence, we use the number of hash computations to estimate the time complexity. We present the best and worst cases in ACTION's authentication procedure, which are the computation's lower bound and upper bound, respectively.

In the best case, the reader always meets only one index number at the non-leaf node at each level of the key tree. After *d* steps probing, the reader successfully identifies a tag. With the same branching factor setting δ, the depth of sparse tree is larger than the balanced-tree, that is, $d > \log\delta N$. Therefore, the computational lower bound of ACTION's identification is logδ*N*.

As we assume the branching factor of the key tree is δ, each non-leaf node has at most δ children. In the worst case, at the root, the reader will compute δ hash values, and narrow the search scope to *N*/δ tags; at a child node of the root, the reader performs δ hash computations again. Then the reader narrows the search scope to N/δ^2 tags. At this time, the reader spends 2δ hash computations. The reader repeats the same process at each level. At a given level *l*, the reader narrows the search scope to N/δ^l tags, and performs $l \cdot \delta$ hash computations. We assume at level *l*, the reader finds $N/\delta^l = 1$, or $l = \log\delta N$. Since $d > \log\delta N$, the reader does not reach leaf nodes at level *l*. We assume that the reader reaches a non-leaf node *a* at level *l*. The node *a* must have only one child (if *a* has two children, the number of tags in the system must be *N*+1, not *N*). Similar to *a*, each node of *a*'s offspring has only one child, except leaf nodes that are always childless. Thus, the reader will perform $d - l$ hash computations after level *l*. We illustrate the worst case in Fig. 4.

We calculate hash computations in the worst case, $f(\delta) = \delta \times l + d - l$. Since $l = \log\delta N$, and $d = n/\log\delta$ (see Section 3.3).we have

$$\begin{aligned} f(\delta) &= \delta \cdot \log_\delta N + (\frac{n}{\log\delta} - \log_\delta N) \\ &= (\delta - 1) \cdot \log_\delta N + \frac{n}{\log\delta} \end{aligned} \qquad (2)$$

In (2), *n* is the bit length of keys in the system; in ACTION, *n* = 128. Let E_{ACTION} de note identification efficiency, $\log_\delta N < E_{ACTION} \leq (\delta - 1) \cdot \log_\delta$

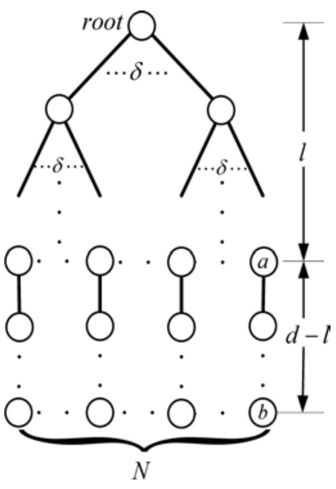


FIGURE 4
The worst case of ACTION.

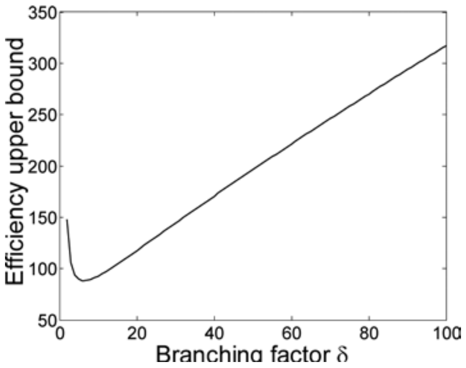


FIGURE 5
Efficiency upper bound vs. branching factor. (Assume $N = 2^{20}$, $n = 128$).

$N + \frac{n}{\log\delta}$. Hence, E_{ACTION} is $O(\log \delta N)$. We plot the curve of the efficiency upper bound $f(\delta)$ in Fig. 5.

To find the optimal δ, we set $f'(\delta) = 0$. We have

$$\log_\delta N = \frac{n \log e \ln \delta}{(\delta(\ln\delta - 1) + 1)\log^2 \delta} \tag{3}$$

By solving (3), we find that $\delta = 8$ is the optimal setup for identification efficiency. The upper bound is $f(8) = \frac{7}{3}\log N + \frac{n}{3}$. According to the relation between δ and d, if $\delta = 8$, then $d = 128/(\log 8) = 128/3$. By that setup d is not an integer. Hence, in ACTION, we set a sub-optimal $\delta = 16$, such that $d = 32$. The upper bound is $f(16) = \frac{15}{4}\log N + \frac{n}{4}$. Combined with the early discussion, we can see that the time complexity of ACTION authentication is $O(\log N)$.

5 PRIVACY AND SECURITY

The essential goal of ACTION is to protect the privacy and defend against both passive and active attacks. For RFID systems, passive attack often means eavesdropping on the communication between tag T and reader R, which are intensively discussed in previous designs [5-10, 13-19]. Active adversaries can forge, replay, or discard the messages exchanged between T and R, so the attacks include tracking, cloning, and tag-compromising [7]. Adversaries are even able to execute bogus authentication procedures. Up to now, the research on RFID is still short of appropriate formal models that can explicitly define the privacy and adversaries in a general way. Lacking such models, existing PPA schemes have to employ ad hoc notions of security and privacy [13], and then heuristically analyze the security and privacy via those notions. The heuristic analysis, however, only allows those PPA schemes examine the privacy under the known attacks. It is difficult to explore the potential vulnerabilities and flaws that are vulnerable to newly emerging attacks.

Juels proposes a “Strong privacy” [13] to meet the demands on privacy in RFIDs. This model employs *indistinguishability* to define the privacy. Loosely speaking, indistinguishability means that RFID tags should not be told from each other according to their output. Thus, tags need to randomize their output such that adversaries cannot distinguish a tag from others. Although the Strong privacy model presents a method to protect tags’ privacy completely, the major problem of it lies in the authentication efficiency. A PPA protocol that satisfies the Strong privacy, the legitimate reader cannot be directly aware of which tag it is interrogating due to the random output of the tag. The reader thereby has to search all tags in the system to identify the tag instead. Therefore, the authentication efficiency is linear to the number of tags in a given system, and PPAs that satisfy the Strong privacy model are not more efficient than linear search. As analyzed in [24], protocols with logarithmic efficiency cannot be proven private under the Strong privacy model.

The authentication efficiency is one of major concerns in RFID systems, so many PPAs focus on improving authentication efficiency. Although these PPAs are more efficient than linear search, their privacy cannot satisy the Strong privacy due to the tradeoff between privacy and authentication efficiency. These PPAs including ACTION thus cannot be proven private formally. L. Lu *et al* [24] propose a “Weak privacy” to address this issue that how much cost on the privacy degradation brings back how much improvement on the authentication efficiency. The Weak privacy loosens the strict constraints on the output of tags, such as randomization and unpredictability. For stating the tag's identity, it allows a tag's output to contain a temporally constant field, which will be refreshed at the successive authentication. By this means, RFID systems can achieve acceptable privacy protection as well as highly efficient authentication.

In this section, we use the weak privacy model [24] to explicitly define the privacy and adversaries. Based on this model, we generally prove that ACTION can preserve privacy of RFID systems, instead of using those attacks one by one to verify the capabilities against those attacks. We also briefly present other security guarantees of ACTION.

5.1 Privacy definition

The model consists of three components: RFID Scheme, Adversaries, and Privacy Game.

5.1.1 RFID scheme

In the model, an RFID scheme is defined as following:

Definition 1 (RFID Scheme): An RFID scheme has four components:

1. A polynomial-time algorithm *KeyGen*(1^s) which generates all key materials $k_1, \ldots, k_n$ for the system depending on a security parameter s.
2. A setup scheme *SetupTag*(*ID*) which allocates a specific secret key k and a distinct *ID* to a tag. Each legitimate tag should have a pair (*ID, k*) stored in the back-end database.
3. A setup scheme *SetupReader* which stores all pairs (*ID, k*) in the reader's back-end database for all legitimate tags ID in the system.
4. A polynomial-time interactive protocol *P* between the reader and a tag in which the reader owns the common inputs, the database and the secrets. If the reader fails, it outputs $\perp$. Otherwise, outputs some ID and may update the database.

An RFID scheme has a correct output if the reader executes the protocol *P* honestly and then infers the *ID* of a legitimate tag except with a negligible probability (A function in terms of a security parameter s is called negligible if there exists a constant $x > 0$ such that it is $O(x^{-s})$); otherwise, the reader outputs $\perp$ when the tag is not legitimate.

5.1.2 Adversaries

Adversaries in the model have three characteristics: the oracles they can query, the goal of their actions, and the rules of their actions. According to those characteristics, we define adversaries in RFID systems below.

Definition 2 (Adversaries): An adversary *A* in an RFID system is a polynomial-time algorithm which performs attacking behaviors by querying five oracles.

1. *Launch* → π*:* Execute a new protocol instance π between the reader and a tag.
2. *TagQuery* (*m,* π*, T*) → *m':* Send a message *m* to a given protocol session π on the tag *T.* The oracle returns a message *m'* as the output of *T.*
3. *ReaderSend*(*m,* π*, R*) → *m':* Send a message *m* to a given protocol session π on the reader *R*. The oracle returns a message *m'* as the output of *R*.
4. *Corrupt*(*T*)*:* Compromise the tag *T*, and obtain the secret stored in *T*. the tag *T* is no longer used after this oracle call. In this case, we say that the tag *T* is destroyed.
5. *Result*(π)*:* When π is complete, the oracle returns 1 if the scheme has the correct output; otherwise, it returns 0.

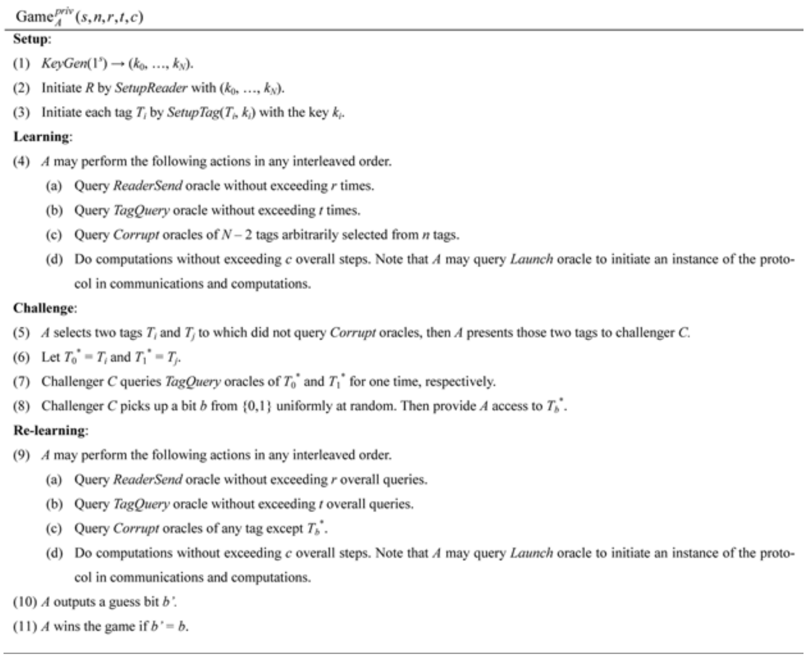
The adversary starts a game by setting up the RFID system and feeding the adversary with the common parameters. The adversary uses the oracles above following a privacy game, which will be described in next subsection, and produces the output. Depending on the output, the adversary wins or loses the game.

5.1.3 Privacy game

The game experiences three phases: *Learning, Challenging* and *Re-learning*.

As shown in Fig. 6, in the Learning phase, *A* is able to issue any message and perform any polynomial-time computation (i.e., query oracles in polynomial times). After the Learning phase, *A* selects two uncorrupted tags as challenge candidates in the challenge phase. One of those challenge candidates is then randomly chosen by the system (the *challenger C*) and presented to the adversary (the oracles of the selected tag can be queried by the adversary except the *Corrupt* oracle). After that, similar to the Learning phase, *A* is offered the oracles of all tags in the RFID system by *C* except the two challenge candidates. This phase is named Re-learning. At the end of Re-learning, *A* outputs a guess about which candidate tag is selected by *C*. If the guess is correct, *A* wins the game; otherwise, *A* loses.

In addition, there are two requirements for our privacy game to work properly. First, at the step (7) in the Fig. 6, the challenger *C* refreshes the private information of the two challenge candidates T_0^* and T_1^*. Thus, the adversary cannot correlate the output of T_b^* at the Re-learning phase with the output of T_0^* and T_1^* at the Learning phase. Second, if an adversary can corrupt $N - 1$ tags and get the keys of those tags, then the adversary can retrieve the output of those corrupted tags. Therefore, any tag in the system can be definitely



$\mathrm{Game}_A^{priv}(s,n,r,t,c)$

Setup:

(1) $KeyGen(1^s) \rightarrow (k_0, \ldots, k_N)$.

(2) Initiate R by *SetupReader* with $(k_0, \ldots, k_N)$.

(3) Initiate each tag T_i by $SetupTag(T_i, k_i)$ with the key k_i.

Learning:

(4) A may perform the following actions in any interleaved order.

- (a) Query *ReaderSend* oracle without exceeding r times.
- (b) Query *TagQuery* oracle without exceeding t times.
- (c) Query *Corrupt* oracles of $N - 2$ tags arbitrarily selected from n tags.
- (d) Do computations without exceeding c overall steps. Note that A may query *Launch* oracle to initiate an instance of the protocol in communications and computations.

Challenge:

(5) A selects two tags T_i and T_j to which did not query *Corrupt* oracles, then A presents those two tags to challenger C.

(6) Let $T_0^* = T_i$ and $T_1^* = T_j$.

(7) Challenger C queries *TagQuery* oracles of T_0^* and T_1^* for one time, respectively.

(8) Challenger C picks up a bit b from $\{0,1\}$ uniformly at random. Then provide A access to T_b^*.

Re-learning:

(9) A may perform the following actions in any interleaved order.

- (a) Query *ReaderSend* oracle without exceeding r overall queries.
- (b) Query *TagQuery* oracle without exceeding t overall queries.
- (c) Query *Corrupt* oracles of any tag except T_b^*.
- (d) Do computations without exceeding c overall steps. Note that A may query *Launch* oracle to initiate an instance of the protocol in communications and computations.

(10) A outputs a guess bit b'.

(11) A wins the game if $b' = b$.

FIGURE 6
Privacy game.

distinguished from others with the output of the tag. That is why at least two tags need to be uncorrupted.

We denote such a privacy game for an RFID system as $\mathrm{Game}_A^{priv}(s,N,r,t,c)$. . Here s is a security parameter, for example, the length of keys, and N, r, t, and c are respective parameters for number of tags, number of *ReaderSend* queries, number of *TagQuery* queries, and computation steps. An adversary A with parameters r, t, and c is denoted by $A[r, t, c]$.

Based on the privacy game in Fig. 6, we define the privacy of an RFID scheme in Def. 3.

Definition 3 (RFID (r, t, c) – privacy*)*: *A protocol P* of an RFID system achieves (r, t, c) – privacy with parameter s, if for any polynomial-time adversary A, the probability of A wining under $\mathrm{Game}_A^{priv}(s,N,r,t,c)$ satisfies:

$$\forall A(r,t,c), \Pr[A\ wins] \leq \frac{1}{2} + {}^{1}\!\big/_{poly(s)}$$

where $poly(s)$ denotes any polynomial function of parameter s.

For a given protocol P in an RFID system, we define the *advantage* of an adversary by:

$$\mathrm{Adv}_P(A) = \Pr[A\ wins] - \frac{1}{2}$$

In $\mathrm{Game}_A^{priv}(s,N,r,t,c)$, the adversary *A* can win the game in a trivial way. That is *A* picks up a bit *b'* from {0, 1} uniformly at random, i.e., $\Pr[b' = b] = \frac{1}{2}$. In this case, *A* attacks the system without any knowledge about the tags in the system, the successful attacking probability is the lower bound of all attacking activities. Therefore, we define the advantage of any polynomial-time adversary by $\Pr[b' = b] = \frac{1}{2}$.

5.2 Privacy proof

Based on the model given in Section 5.1, we formally prove that ACTION protocol satisfies (*r, t, c*) – privacy, which means an adversary has a negligible advantage when it conducts attacks on the ACTION.

Theorem 1. ACTION achieves (*r, t, c*) – privacy under random oracle model [23], for any polynomial-time adversary *A*, i.e., for any *r, t* and *c* polynomial in the security parameter *s*, and the advantage of an adversary *A* is bounded by

$$\mathrm{Adv}_{\mathrm{ACTION}}(A) \le \frac{(r+t)^2 + c^2 + 4c}{2^{s+1}}$$

Proof: In this proof, we use the *random oracle* (RO) model [23], in which hash functions are treated as arbitrary random functions. Since all keys in ACTION are generated independently, the keys of a tag are not related to those in other tags. We denote the game between the challenger *C* and the adversary *A* as G_0.

We introduce another challenger *C'*, (who plays a simulated game G_1 with the adversary *A*), to simulate the real challenger *C*, and make them indistinguishable to *A*. Thus, from the viewpoint of *A*, the game G_1 between *A* and *C'* exactly simulates the real game G_0 between *A* and the real challenger *C*. On the other hand, we construct *C'* without the knowledge of T_0 and T_1's secret keys, k_0^l, k_0^p, k_1^l and k_1^p. That said, there is no information about T_0 and T_1's keys is leaked to adversary *A*, so that *A* must randomly guess which tag T_0 or T_1 is, that is, guessing the bit *b* (see the step 10 of the attack model in Fig. 6) at random. In this case, the probability of a correct guess is 1/2. According to the Def. 3, *A*'s advantage in G_1 is 0. Obviously, G_1 almost perfectly simulates the real game G_0, so the activities of the challenger *C'* would also perfectly simulate the real challenger *C*. However, without the knowledge of T_0 and T_1's secret keys, there are some differences, called *Exceptions*, between the activities of *C'* and *C* in some situations. If we can estimate the probability of *Exceptions* happening, we can compute the upper bound of *A*'s advantage.

In G_1, the challenger *C'* simulates the hash function *h* in ACTION as a RO *h'*. *h'* is constructed as a hash value list, *H_list*, maintained by *C'*. *H_list* is initialized as empty. The format of each item in *H_list* is (r_1, r_2, k, v), where v is the hash value of r_1, r_2, and k, i.e. $v = h(r_1, r_2, k)$.

For a query (r_1, r_2, k): If it exists in *H_list*, *C'* returns the corresponding $v = h(r_1, r_2, k)$; Otherwise *C'* picks up a v uniformly at random, returns the v as the answer of $h(r_1, r_2, k)$, and adds (r_1, r_2, k, v) into the *H_list*.

In the real game G_0, each message is computed with the hash function; the outputs of oracles *TagQuery* and *ReaderSend* are also computed with the hash function. Thus, we use the *h'* given above to construct the *TagQuery* and *ReaderSend* oracles in the G_1.

According to the ACTION protocol, the inputs of the *TagQuery* oracle are "Request" and a nonce r_1, and the outputs are the authentication messages $U = (r_2, h(r_1, r_2, k_i^p[0]), h(r_1, r_2, k_i^p[1]),\ldots, h(r_1, r_2, k_i^p[d\text{-}1]))$. In G_1, the challenger *C'* simulates the *TagQuery* oracle as follows:

Upon receiving the "Request" and r_1, the challenger *C* generates a nonce r_2 and two n-bits long keys k^p and k^l uniformly at random respectively, and then divides k^p into d parts, $k^p[0]$, $k^p[1],\ldots$, $k^p[d\text{-}1]$. Then *C'* accesses the random oracle *h'* for d times to get the hash value sequence $h(r_1, r_2, k^p[0])$, $h(r_1, r_2, k^p[1]),\ldots, h(r_1, r_2, k^p[d\text{-}1])$; *C'* computes the hash value $h(r_1, r_2, k^l)$ by accesses *h'*; Return $U = (r_2, h(r_1, r_2, k^p[0]), h(r_1, r_2, k^p[1]),\ldots, h(r_1, r_2, k^p[d\text{-}1]), h(r_1, r_2, k^l))$.

Similarly, *C'* simulates the *ReaderSend* oracle in G_1 as follows:

Upon $U = (r_2, h(r_1, r_2, k_i^p[0]), h(r_1, r_2, k_i^p[1]),\ldots, h(r_1, r_2, k_i^p[d\text{-}1]), h(r_1, r_2, k_i^l))$: Generates a nonce r_1, a path key k^p, and a leaf key k^l uniformly at random; Accesses the random oracle *h'* to get the hash value $h(r_1, r_2, k^p, k^l)$ and $h(r_1, r_2, k^l)$; Accesses the random oracle *h'* to get the hash value $h(r_1, r_2, h(r_1, r_2, k^p))$ and $h(r_1, r_2, h(r_1, r_2, k^l))$. Returns $\sigma = (1, h(r_1, r_2, h(r_1, r_2, k)), h(r_1, r_2, h(r_1, r_2, k^l)))$ (where 1 is the value of s, the number of TagJoin algorithm running; see Section 3.5).

For *Corrupt* oracle, *C'* transfers oracle queries to the challenger *C* in G_0 and then returns the results from *C* directly.

G_1 is similar to G_0 except the constructions of the random oracle, *TagQuery* and *ReaderSend* oracles. In G_1, from the viewpoint of *A*, *C'* simulates *C* perfectly except following events happens:

1. Collisions in the input of the hash function h. For example, in G_0, the real hash function h will treat $(r_1, r_2, \cdot)$ and $(r_2, r_1, \cdot)$ as same input, therefore the output of h will be identical. In G_1 however, according to the definition of random oracle, *h'* considers that $(r_1, r_2, \cdot)$ and $(r_2, r_1, \cdot)$ are different, the output of *h'* of course is different. Thus, *C'* cannot answer *A*'s query correctly. We denote this event as $Event_1$.
2. Collisions in the output of *h'*. Since *A* performs at most c computations, the number of *h'* is not more than c. We denote this event as $Event_2$.

3. *A* guesses the correct keys of tags T_i and T_j. Thus, *A* can find that the output from *C'* is not correct. We denote this event as $Event_3$.

The probabilities of $Event_1$ and $Event_2$ are bounded by the birthday paradox:

$$\Pr[Event_1 \vee Event_2] \le \frac{(r+t)^2 + c^2}{2 \cdot 2^s}$$

For $Event_3$, given that *h'* is a random oracle, its output reveals no information about the secret keys. Hence, the probability that an adversary *A* can successfully guess keys of T_i and T_j is at most $\frac{2c}{2^s}$.

As discussed at the beginning of the proof, the advantage of *A* in G_1 is zero. Considering the probabilities of any event happening, the advantage of *A* in G_0 is bounded by:

$$\begin{aligned} \mathrm{Adv}_{\mathrm{ACTION}}(A) &= \Pr[Event_1 \vee Event_2 \vee Event_3] \\ &\le \frac{(r+t)^2 + c^2}{2 \cdot 2^s} + \frac{2c}{2^s} \\ &= \frac{(r+t)^2 + c^2 + 4c}{2^{s+1}} \end{aligned}$$

From above inequation, we get that the advantage of any adversary *A* is negligible. ACTION hereby is (*r, t, c*) – *privacy* according to Def. 3.

Theorem 1 states that, under the privacy model defined in Section 5.1, the advantage of adversaries is negligible. That is, in the extreme case, even if an adversary has captured $N - 2$ tags, the probability of distinguishing a normal tag from another one is still 1/2. In general, assume the adversary has tampered with *t* tags. To distinguish a normal tag, the adversary has to perform random guessing on $N - t$ normal tags, and the probability of correctly guessing, that is, the probability of a successful attack, $\alpha = 1/(N - t)$.

We compare the successful probabilities of attacks in balanced-tree based approaches, SPA, and ACTION. In this comparison, we assume SPA and balanced-tree based approaches use binary trees. The RFID system contains 2^{20} tags. As shown in Fig. 7, in SPA and other balanced-tree based approaches, adversaries have an overwhelming probability of distinguishing any normal tag after they tamper with 10 tags in the system, while ACTION perfectly eliminates the impact of those attacks.

5.3 Security

In this subsection, we show how ACTION achieves other security objectives. Specifically, a PPA must achieve the following security objectives as well [6, 9].

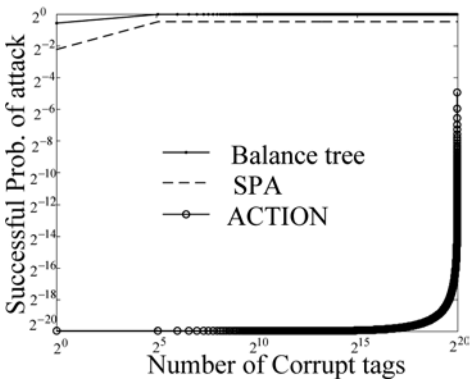


FIGURE 7
Comparisons on defending against the compromising attack (Assume $N = 2^{20}$).

5.3.1 Key extraction attack

We note that the path key of a tag may suffer from an extracting attack. In ACTION, we set the branching factor at 16 when the length of a path key is 128-bits long. The path key will be divided into 32 4-bit parts with this setting. In this case, for any identification message $h(r_1, r_2, k_i^p[j])$, an adversary can easily extract $k_i^p[j]$ by enumerating all 16 4-bit strings, like a brute-force search. Repeating the enumeration, the adversary can crack the entire path key.

Actually, the effect of extraction attack is very limited. First, adversaries can only get path key temporally. In Action, even if adversaries can extract the path key of the tag which is been accessed, they can only track a tag within the interval between two consecutive interrogations from the legitimate reader. After a successful authentication with the legitimate reader, the path key of tag will be refreshed or updated for next authentication procedure. Second, path key is protected with leaf key. The length of each leaf key is similar to that of the path key, that is, 128 bits in our protocol. After identification, the path key is updated by $h(r_1, r_2, k_i^p, k_i^l)$ and the key k_i^l is also updated accordingly in each key updating procedure. Without knowing the leaf key, the adversary cannot predict the updated path key by guessing or performing a brute-force-like search on its sub-parts. Thus, ACTION can be resilient to an extracting attack.

In a normal RFID system without privacy-preserving technologies, an adversary is able to track a target tag by continuous scanning, while in ACTION, the adversary can also record the trace of the target tag within the time interval between two successive authentications. In the worst case, the impact on privacy of ACTION is identical to the normal RFID system if the time interval is infinite. The adversary against ACTION, however, cannot predict the future output of the target tag even in the case of long time interval since the keys stored in the tag will be refreshed. That is, the adversary can track the target tag temporally in ACTION, but permanently in normal RFID systems. Secondly, although ACTION is subject to privacy degradation (i.e. temporally tracking), it achieves logarithmic complexity, which is more efficient than Hashlock-like approaches that protect tag privacy completely. Thirdly, balanced-tree based approaches have same authentication complexity as ACTION, but cannot be formally proved private under current formal privacy models including the Strong as well as the Weak privacy model. Moreover, balanced-tree based approaches cannot defend against the compromising attack, while ACTION can do. Therefore, the privacy of ACTION holds a midway between Hashlock-like and balanced-tree based approaches.

5.3.2 Cloning resistance

This property means that adversaries cannot impersonate a valid tag via bogus tags or repeatedly forwarding valid responses to the reader.

In a cloning attack, an adversary captures the messages from a tag and resends them to the reader [6]. In ACTION, the reader and the tag embed random numbers r_1 and r_2 in the authentication messages to defend against the cloning attack. Since the random numbers r_1 and r_2 are generated uniformly at random and are varied in each authentication procedure, it is infeasible for an adversary to predicate them. In addition, the length of r_1 or r_2 in ACTION is sufficiently long (more than 64 bits), which guarantees the probability of an adversary successfully guessing the random numbers as negligible. Thus, ACTION is not subject to cloning attacks.

5.3.3 Forward secrecy

Forward secrecy means that adversaries cannot reveal the previous messages sent from the captured tag if they compromise a tag and obtain the keys.

If a tag is captured, the adversary might obtain the tag's current keys. In ACTION, however, the adversary cannot trace back the tag's previous communications because the keys have been updated at the latest authentication procedure. That means the adversary, even if obtaining the keys from a tag, cannot retrieve any useful information from the past outputs of the tag, unless it can successfully invert the one-way cryptographic hash function. On the contrary, not many balanced tree based protocols [6, 7], can update the keys in practical systems. In those approaches, an adversary can easily reveal all past authentication messages of a tampered tag if revealing the stored keys.

5.3.3 Tag Impersonation

The aim of tag impersonation in the context of authentication is to make an honest reader accept a fake tag as valid. It should be noted that the keys in a tag are constantly refreshed in every request from readers and then past tag responses are uniformly distributed irrespective of the queries requested.

Therefore, a fake tag without knowledge of valid keys has no advantage than to reply with random responses. Let *N* and *L* be the number of total tags managed by a reader and the length of hash values in tag responses (see the message *U* in Fig. 3), respectively. Then the probability of the response being accepted by the reader is at most $N/2^{(d+1)L}$ for each query. Where *d* is the number of path keys stored in a tag. Thus, our protocol can defend against tag impersonation with a cheating probability of at most $N/2^{(d+1)L}$.

5.3.4 Reader Impersonation

The aim of reader impersonation is to make an honest tag accept the adversary as a legitimate reader. Obviously, a fake reader without knowledge of valid shared secret keys associated with an honest tag. Thus, the adversary has no advantage than to send a random *s* (see the last message in Fig. 3) in the final protocol round. The probability of such a response being accepted by the honest tag is negligible. Now assume that an adversary can tamper with a tag at time *t*. Suppose that the adversary obtains the tag's secret keys. The adversary obviously cannot get any advantage if the tag has been identified by a legitimate reader at time $t' > t$ which the adversary could not eavesdrop, since the secret keys stored in the tag would have been refreshed with the random numbers generated by the legitimate reader which is unknown to the adversary. Thus, we consider the case of the adversary attacking the tag immediately after compromising the tag secret keys. This is the only potential threat but inevitable in our protocol. This threat, however, is useless to the adversary in practice, since the adversary cannot impersonate the legitimate reader to other tags even he totally controls the compromised tag.

5.3.5 Denial of Service

Our protocol has strong resistance against Denial-of-Service (DoS) attacks on the last protocol message. Any block or alteration of this message may cause desynchronization of keys shared between the tag and the reader, but such a desynchronization problem can be detected by a legitimate reader in the next identification. Specifically, the tag would not update the shared keys if the last message sent from the reader (see *s* in Fig. 3) has been blocked or altered, since *s* will not pass the verification by the tag. The reader, however, can detect the inconsistence of shared keys stored in the tag and the reader, and then launch a new instance of ACTION to identify and update the tag.

6 CONCLUSIONS

We propose a privacy-preserving authentication protocol, ACTION, to support secure and efficient authentication in RFID applications. To the best of our knowledge, this is the first work that is able to defend against a compromising attack in tree-based approaches. The advantages of this design also include high efficiency in terms of storage and identification. We believe wide deployment of this design will make privacy preserving authentications more practical and effective for large scale RFID systems.

ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China under Grant No. 61173171, the Fundamental Research Funds for the Central University under Grant No. ZYGX2012J072 and Project funded by China Postdoctoral Science Foundation No. 2014M550466

REFRENCES

[1] T. Kriplean, E. Welbourne, N. Khoussainova, V. Rastogi, M. Balazinska, G. Borriello, T. Kohno, and D. Suciu, “Physical Access Control for Captured RFID Data,” IEEE Pervasive Computing, vol. 6, 2007.
[2] B. Sheng, C. C. Tan, Q. Li, and W. Mao, “Finding Popular Categories for RFID Tags”. in Proceedings of ACM Mobihoc, 2008.
[3] Y. Li and X. Ding, “Protecting RFID Communications in Supply Chains,” in Proceedings of ASIACCS, 2007.
[4] 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, 2003.
[5] S. Weis, S. Sarma, R. Rivest, and D. Engels, “Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems,” in Proceedings of International Conference on Security in Pervasive Computing, 2003.
[6] T. Dimitriou, “A Secure and Efficient RFID Protocol that Could make Big Brother (partially) Obsolete,” in Proceedings of PerCom, 2006.
[7] D. Molnar and D. Wagner, “Privacy and Security in Library RFID: Issues, Practices, and Architectures,” in Proceedings of CCS, 2004.
[8] D. Molnar, A. Soppera, and D. Wagner, “A Scalable, Delegatable Pseudonym Protocol Enabling Owner-ship Transfer of RFID Tags,” in Proceedings of Selected Areas in Cryptography - SAC, 2005.
[9] L. Lu, J. Han, L. Hu, Y. Liu, and L. M. Ni, “Dynamic Key-Updating: Privacy-Preserving Authentication for RFID Systems,” in Proceedings of PerCom, 2007.
[10] G. Avoine, E. Dysli, and P. Oechslin, “Reducing Time Complexity in RFID Systems,” in Proceedings of Selected Areas in Cryptography - SAC, 2005.
[11] S. Bono, M. Green, A. Stubblefield, A. Juels, A. Rubin, and M. Szydlo, “Security Analysis of a Cryptographically-Enabled RFID Device,” in Proceedings of USENIX Security, 2005.
[12] M. C. O’Connor, “Taking Advantage of Memory-Rich Tags”, http://www.rfidjournal.com/magazine/article/2925.
[13] Juels and S. Weis, “Defining Strong Privacy for RFID,” in Proceedings of PerCom, Workshop PerTec, 2007.
[14] Juels, “RFID Security and Privacy: a Research Survey,” Journal of Selected Areas in Communications, vol. 24, pp. 381-394, 2006.
[15] M. Ohkubo, K. Suzuki, and S. Kinoshita, “Efficient Hash-Chain based RFID Privacy Protection Scheme,” in Proceedings of UbiComp, Workshop Privacy, 2004.

[16] Juels, “Minimalist Cryptography for Low-Cost RFID Tags,” in Proceedings of International Conference on Security in Communication Networks - SCN, 2004.

[17] T. Dimitriou, “A Lightweight RFID Protocol to Protect Against Traceability and Cloning Attacks,” in Proceedings of SecureComm, 2005.

[18] G. Tsudik, “YA-TRAP: Yet Another Trivial RFID Authentication Protocol,” in Proceedings of PerCom Workshops, 2006.

[19] G. Avoine and P. Oechslin, “A Scalable and Provably Secure Hash Based RFID Protocol,” in Proceedings of PerCom, Workshop PerSec, 2005.

[20] M. E. Hellman, “A Cryptanalytic Time-Memory Trade-off,” IEEE Transactions on Information Theory, vol. 26, pp. 401-406, 1980.

[21] “Expands Tag-it ISO/IEC 15693 RFID Product Line”, News Releases from Texas Instruments, http://www.ti.com/rfid/shtml/news-releases-rel12-14-05.shtml.

[22] G. Avoine, “Adversarial Model for Radio Frequency Identification,” Technical Report 2005/049. http://eprint.iacr.org/2005/049, 2005.

[23] M. Bellare and P. Rogaway, “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols,” in Proceedings of CCS, 1993.

[24] L. Lu, Y. Liu and X. Li, “Refresh: Weak Privacy Model for RFIDs”, in Proceedings of INFOCOM, 2010.