Randomizing RFID Private Authentication

Qingsong Yao[1], Yong Qi[1], Jinsong Han[2], Jizhong Zhao[1], Xiangyang Li[3], Yunhao Liu[2]

[1]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China

[2]Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

[3]Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, USA

***Abstract*—Privacy protection is increasingly important during authentications in Radio Frequency Identification (RFID) systems. In order to achieve high-speed authentication in large-scale RFID systems, researchers propose tree-based approaches, in which any pair of tags share a number of key components. Such designs, being efficient, often fail to achieve forward secrecy and resistance to attacks, such as compromising and desynchronization. Indeed, these attacks may still take effect even after a tag successfully finishes the authentication and key-updating procedure. To address the issue, we propose a lightweight RFID private authentication protocol, RWP, based on the random walk concept. RWP also provides the forward security and temporal resistance to the tracking attack. The analysis results show that RWP effectively enhances the security protection for RFID private authentication, and increases the authentication efficiency from $O(\log N)$ to $O(1)$.**

Keywords-RFID; private authentication; random walk.

I. INTRODUCTION

Radio Frequency Identification (RFID) is a popular technology that automatically identifies objects or people [1, 2]. For security consideration, authentication is required by most RFID applications. A core requirement of RFID systems is protecting user privacy in the authentication process [3]. Different from other pervasive or mobile devices [4-7], the RFID tag has limited computation and storage capacity. To enable the authentication, RFID systems usually adopt the challenge-response scheme. Typically, each tag shares a distinct symmetric key (or keys) with the reader and the reader stores the key in the backend database. When a tag approaches the reader, the reader issues a query containing a nonce. The tag encrypts the nonce and sends the cipher to the reader. The reader then searches the backend database to locate the key, and hence identify the tag. For the ease of discussion, we denote T as the tag and R as the combination of backend application and the RFID reader.

In RFID systems, private authentication protocols can be categorized into two groups, tree-based and non-tree-based approaches. Non-tree-based protocols usually employ a linear key storage structure in the backend database. Each pair of tags has no shared key components such that the keys of tags are not correlated. The search efficiency of non-tree-based approaches, however, is unacceptable for large-scale RFID systems due to the linear key structure. On the other hand, tree-based schemes leverage virtual trees for key assignment and management. In such a virtual tree, every virtual node holds a key component. Each leaf node is uniquely assigned to a tag. After the assignment, there exists a path from the root to each leaf node in the key tree. The keys on such a path are correspondingly allocated to the tag associated with the leaf node, as illustrated in Fig. 1. In the tree structure, any pair of tags share at least one key component. The tree-based key structure improves the authentication efficiency from $O(N)$ to $O(\log N)$, where N is the number of tags in the system [8].

Although tree-based designs significantly accelerate authentication speed, the key components shared among tags lead to security flaws. A number of attacks raise privacy challenges to the authentication. Among them, the compromising attack is widely known as the most critical threat to tree-based approaches. In the compromising attack, attackers can compromise a tag and then obtain all its key components, so that they are able to track other tags based on the shared key components. The compromising attack also incurs a detriment to the forward secrecy, which is a guarantee that even if a tag's key is leaked, all previously encrypted messages cannot be disclosed. Lu et al. propose SPA [9], an authentication protocol with key-updating, to mitigate the impact of compromise attacks and partially provides forward secrecy. The threat of compromising attack, however, has not been thoroughly eliminated since the shared key components among tags still exist. Even worse, the mapping relation between a tag and a leaf node is fixed in most tree-based approaches, which may facilitate the attacker's probing.

This work is supported in part by the NSFC grants No. 60873262 and No. 90612014, National High Technology R & D Program of China (863 Program) under grants No. 2007AA01Z180 and No. 2006AA01Z101, and Hong Kong ITF GHP/044/07LP.

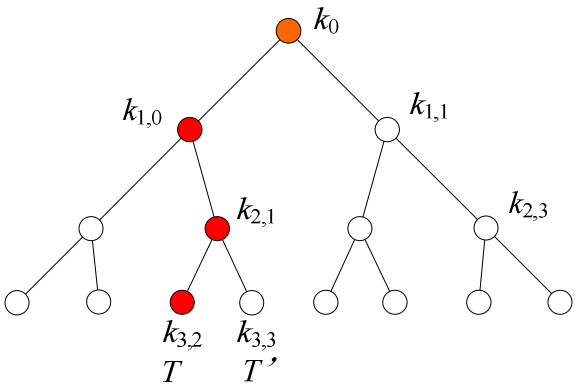

Figure 1. Tree-based architecture.

Besides the compromising attack, RFID private authentication also suffers from the mix of tracking attacks and desynchronization attacks. The desynchronization attack [10] interrupts the updating or synchronization procedure between the reader and tag, which causes failure in the subsequent authentications. Therefore, a temporally permanent protection on privacy is necessary when performing updating in RFID systems. In practice, the adversary is capable of conducting an attack that mixes the compromising and desynchronization attacks for tracking the tag. Unfortunately, all the existing works we are aware of cannot defend against such an attack.

In this paper, we propose a concept of *common-keys* effect which considers three security concerns altogether, the compromising attack, desynchronization attack, and forward secrecy. We comprehensively analyze the intrinsical reason of privacy degradation in the state of the art approaches. To eliminate the impact of *common-keys* effect, we propose a Random Walk based RFID private authentication Protocol, called RWP, which introduces randomness to both the authentication and updating procedures. In this design, authenticating a tag is similar to randomly walking in the key tree. As a result of updating, RWP also changes the tag's position in the key tree randomly and unpredictably. As such, RWP can eliminate the *common-keys* effect and close the gaps among the three security concerns. The contributions of this work are twofold.

First, we provide a new insight on the privacy issue of RFID systems, and show that existing authentication approaches still suffer from the multiple attacks even with updating mechanisms. We propose a new concept, *common-keys* effect, to integrate the main privacy concerns into the protocol design.

Second, we propose a lightweight authentication protocol, RWP, to enhance RFID authentication. RWP can effectively withstand to existing attacks, including replaying, compromising, and desynchronization attacks. It also reduces the storage and computing complexity of a tag to $O(1)$. For the backend application, RWP reduces the search and computing complexity of each authentication to $O(1)$.

The rest of this paper is organized as follows. Section II discusses the related works. We present the security requirements of RFID authentication as well as the *common-keys* effect in Section III. We present the RWP design in Section IV, and analyze its security and efficiency in Section V. In Section VI, we conclude the work.

II. Related Work

Many approaches have been proposed to protect user privacy for RFID authentication [11–14]. They can be classified into two categories, non-tree-based approaches and tree-based approaches.

Non-tree-based protocols usually utilize linear search for locating a tag. The search efficiency is $O(N)$, where N is the number of tags. Obviously, the linear search is not efficient in large-scale RFID systems that may have millions of tags [15].

As a typical non-tree-based approach, Hash-lock [16] method uses the hash value of a key as a *metaID* to identify a tag. A tag thus can be tracked by the attackers from the *metaID*. A variant version of Hash-lock introduces a random number as another input to generate the hash value, which needs exhaustive search through all IDs to identify a tag.

Hash-chain [10, 17] enables key-updating using the hash function. After each authentication, the tag and the reader compute the old key's hash value as the new key. Due to the one-way property of hash function, the adversary cannot recover the old keys even if obtaining the current key. Based on Hash-chain, researchers further reduce the search complexity to $O(N^{2/3})$ [18]. Hash-chain based approaches, however, may suffer from desynchronization attacks, in which the adversary deliberately interrupts the authentication or updating procedure to desynchronize the keys used by the tag and reader. In [19], Henrici et al. introduce a counter for each tag to record the number of successful authentications between this tag and the reader. The reader also keeps such a counter for each tag. Only if the two counters are equal, the tag and reader can perform the authentication. For practical implementations, Juels [20] looses the synchronization limit by setting a small window for the counter: as long as the values of two counters are within the window, the tag and reader can still process the authentication. The improvements made on the Hash-chain scheme, however, make a tag more recognizable when an adversary interrogates the tag multiple times. Being iteratively interrogated, the tag's counter would have a distinctly large value, which is distinguishable and degrades the privacy level.

Dimitriou [21] proposes to defense against the desynchronization attack by updating keys only after successful authentications. The tracking attack, however, may still take effect between successful identifications since no updating operation takes place in these intervals.

Henrici et al. [22] propose the triggered hash chains approach that alters the tag ID after each successful authentication. The work in [22] utilizes three different hash functions to generate the response, authenticate the reader, and update the keys, respectively. The triggered hash chains approach allows a tag to update its key only after a successful mutual authentication with valid readers. To attack the

TABLE I. SUMMARY OF EXISTING WORKS

	Privacy	Tracking resistance	Forward secrecy	Authentication efficiency	Cloning resistance
[16][1]	No	No	No	$O(1)$	No
[16][2]	Yes	Yes	No	$O(N)$	No
[10]	Yes	Yes	Yes	$O(N)$	Yes
[18]	Yes	No	Yes	$O(N^{2/3})$	Yes
[19]	Yes	No	Yes	$O(1)$	No
[20]	Yes	No	Yes	$O(1)$	No
[21]	Yes	No	Yes	$O(\log N)$	Yes
[22]	Yes	No	Yes	$O(1)$	No
[23]	Yes	No	No	$O(\log N)$	Yes
[8]	Yes	No	No	$O(\log N)$	Yes
[9]	Yes	No	Yes	$O(\log N)$	Yes
[24]	Yes	No	Yes	$O(\log N)$	Yes
[26]	Yes	No	No	$O(\log N)$	No
[27]	Yes	Yes	No	$O(N)$	Yes

Remark: [1] Basic Hash-lock scheme; [2] Randomized Hash-lock scheme;

triggered hash chains approach, the adversary can keep sending semi-completed authentication queries to a tag. The tag will return identical responses that facilitate attackers' tracking.

The main drawback of the non-tree-based approaches is the low search efficiency. To address the issue, tree-based protocols are proposed. They construct a key tree structure to expedite key searching in large-scale systems. Although the search efficiency can be improved to $O(\log N)$, the tree-based designs suffer from compromising attacks due to the key-sharing structure in the key trees.

Molnar and Wagner introduce a tree-based method [23], in which, each tag is assigned to a leaf node in the key tree. Thus, a path from the root to the leaf node is related to the tag. Identifying a tag is equivalent to exploring the path related to the tag in the key tree. Because of the key-sharing structure, compromising some tags helps to locate other tags.

For defending against cloning or spoofing attack, Dimitriou proposes an enhanced tree-based approach [8]. To resist the compromising attack, Lu et al. propose a RFID private authentication protocol (SPA) [9], which enables a dynamic key-updating for tree-based authentication approaches. The dynamic key-updating [9] provides forward secrecy. Wang et al. also present a protocol (SAPA) to save the storage overhead [24]. The above tree-based approaches, however, cannot completely defend against compromising attacks as the key tree they adopt exhibits the *common-keys* effect, for which we will have more detailed discussions in Section III.

Other works focus on the physical protection or the easy implementation of RFID authentication. In [25], Lim et al. leverage the interference of RF waves to protect the authentication procedure. The tag owner carries powerful jamming equipments emitting random mask-codes during authentication. Interfered by the mask-codes, the adversary cannot correctly receive the authentication messages. The disadvantage of interference-based approaches is that a jamming device should be carried by each tag owner, which incurs extra cost and inconvenience to users. Molnar et al. propose a new method [26] that supports delegation in the tag authentication. The tag owner can transfer the ownership to another party for authenticating valid tags. Similarly, the authors in [27] propose a server-free authentication protocol. It does not need backend server or database. Each reader holds an identifier and a list of records of tags that the reader is authorized to access. The server-free approaches, however, do not provide an efficient key searching mechanism for the backend application.

Table I summarizes the capabilities of the existing works discussed in this section, where N is the number of tags in the system.

III. SECURITY REQUIREMENTS AND COMMON-KEYS EFFECT

We consider *Privacy*, *Cloning resistance*, *Forward secrecy, and Untraceability* as the fundamental security requirements of RFID privacy-preserving authentication [8]. In RFID systems, a private authentication protocol should meet the above security requirements.

Privacy. Any user's private information should not be leaked to any third party during authentication.

Cloning resistance. All the valid tags should not be faked or impersonated. Replay attacks, in which adversaries may repeat the messages sent before to victims tag or readers, should also be infeasible to the authentication procedure.

Forward secrecy. Achieving forward secrecy is that keys stored in a compromised tag cannot reveal the previous outputs of this tag.

Untraceability. A tag should have no correlation with its authentication messages for avoiding tracking.

Besides the above requirements, RFID systems should be able to defend against the compromising and desynchronization attacks. As we mentioned in Section I, compromising a tag may threat other tags if they share a number of keys. In normal tree-based protocols, the compromising attack can help tacking the tags. A mix of tracking and desynchronization attacks remains to be a serious threat to RFID systems. The key reason that RFID systems are

vulnerable to the above attacks is that the keys for encrypting messages are shared during each authentication process. We define the *common-keys* effect to reflect this property.

Definition 1. Given a tag T and its j-th response, the ciphertext in the response message is $M_j = (r_1, r_2, \ldots, r_s)_j$, where the '$s$' denotes the number of ciphers in each message from the tag for responding to a reader's authentication query, and the 'j' denotes the number of such messages when the tag is continuously interrogated by malicious readers. Suppose the attacker can obtain enough response messages M_o, $(1 \le o \le j)$. The tag's output has a *common-keys* effect if there exists an integer i, $(1 \le i \le s)$, such that the attacker can deduce the r_i in the $(j+1)$-th response M_{j+1}.

A tag can easily be identified or be tracked, if its output has the *common-keys* effect. There are two kinds of *common-keys* effect. One is the temporal *common-keys* effect that the keys used by a tag in different authentication procedures can be deducible; the other is the spatial *common-keys* effect that different tags share common key components.

Both the non-tree and tree-based methods may have the temporal *common-keys* effect. In most RFID authentication protocols, if not all, the updating process occurs when the reader R successfully authenticates a tag T. We denote A as the adversary. As the tag is designed to automatically answer the reader's query, A can launch malicious interrogations to a victim tag and observe its responses. Without knowing the keys shared by the valid tag and reader, A cannot complete a successful authentication and updating with T. Hence, the keys and ID of T will not be changed after those incomplete authentications. In both the non-tree-based and tree-based approaches, those response messages are temporally deducible, as in the static tree approaches [8, 26], or predictable, as in the triggered hash chains approach [22], because the ID and keys used by T to generate the responses is identical upon each interrogation from A. According to Definition 1, those responses from T have the *common-keys* effect. Therefore, A can leverage those responses to track T.

On the other hand, the spatial *common-keys* effect is more severe in tree-based approaches. By compromising a number of tags, A can obtain a number of keys stored. Since non-leaf keys are shared in the key tree, A can compute pseudo-random function (PRF) values using the captured keys, so as to distinguish different branches in the key tree as well as the most of the tags [18].

To eliminate the *common-keys* effect, existing works attempt to update keys periodically to avoid tracking. The updating process, unfortunately, fails to change the structure of the key tree, as the mapping relation between a tag and its corresponding leaf node is fixed. After the updating process, the attackers are still able to recognize the tag without knowing those updated keys, because the unchanged key sharing structure helps to track the tag if the adversary is aware of the tree structure. Even after performing updating, for example using SPA [9], a tag does not directly update those keys shared with other tags. Otherwise, the other tags will fail in the following authentications because they are still using the old keys. A, however, can leverage this weakness to gain the knowledge of key tree and the updating status of other tags. Even worse, A may deliberately compromise a small number of tags to distinguish the rest of the tags with high probability.

We illustrate the above impact of *common-keys* using an example. As illustrated in Fig.1, a tag T performs a updating process like in [9], and R updates a key component $k_{3,2}$ at level 3. Note that tree-based approaches usually update a key k as $h(k)$, where $h(\cdot)$ is the hash function. R cannot directly update the key components $k_{2,1}$ in the key tree since the tag T' is still using $k_{2,1}$ at that time. R thereby only updates $k_{3,2}$ and keeps the $k_{2,1}$ unchanged. Later, when T' performs the updating, R will update $k_{2,1}$. If T is compromised, all keys assigned to T are disclosed. In this case, A is aware of the old key $k_{2,1}$, and A can compute the $h(k_{2,1})$ and $h(k_{3,2})$. Thus, the response from T, if T updates its keys, should be encrypted using k_0, $k_{1,0}$, $h(k_{2,1})$, and $h(k_{3,2})$, while the response from T' must be encrypted using k_0, $k_{1,0}$, $k_{2,1}$, and $k_{3,3}$. Although the key component $k_{3,3}$ is still unexposed, the adversary can easily track T' because it can differentiate T' from T according to the difference between using $k_{2,1}$ and $h(k_{2,1})$ in the responses.

Note that each non-leaf node in the key tree can hold δ caches for storing old keys, where the δ is the branch factor of the tree. Due to the restriction of authentication latency, the value of δ cannot be too large. Otherwise R has to execute too many searching operations at each level in the key tree. This constraint also saves the adversary's effort by reducing the number of tags she/he has to compromise to capture sufficient key components.

IV. RWP PROTOCOL

To mitigate the effect of *common-keys*, we design a private RFID authentication protocol without any shared key components in the key structure. In this section, we first give an overview of our RWP protocol, and then present the detailed design.

A. Overview

RWP includes four components, *System Initialization*, *Authentication*, *Updating Process*, and *System Maintenance*. RWP is also tree-based as shown in Fig. 2. Different from existing tree-based approaches, the basic idea of RWP is to setup a key structure in which tags do not share any key. To this end, we remove all the key components from the non-leaf nodes. Indeed, except the leaf node, all the non-leaf nodes only serve for navigating the tree in RWP. A navigation method for identifying the leaf node related to a given tag is also designed. We virtually transfer the tree to a coordinate space. The location of each node in the tree can be mapped to a coordinate. In RWP, we employ a bitstring to represent the coordinate. Initially, each tag is randomly assigned to a leaf node, which is correlated with a '*formal location*'. During authentication, the tag T generates a random number and computes its hash value. The hash value can be mapped to a virtual node in the tree, termed as an *anchor node*. Then T computes the distance offset between its *formal location* and the location of the *anchor node*.

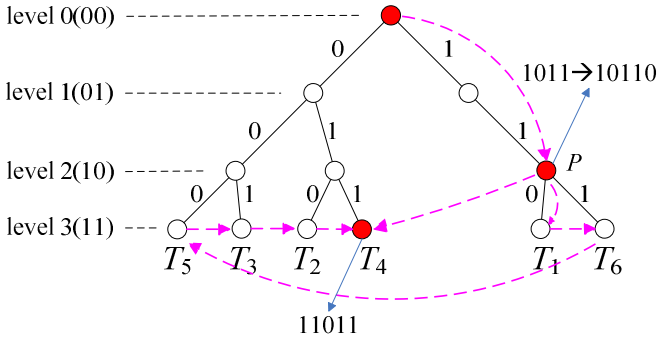

Figure 2. A RWP's key tree.

T then delivers the offset and location of *anchor node* to the reader *R*. Before responding to *R*, *T* employs a hash function to process the response messages for protecting the private information contained in the offset and location of *anchor node*. The information helps *R* to locate the leaf node assigned to *T*. After authentication, *R* randomly assigns *T* to another leaf node, updates the key for the tag, and synchronizes this updating. In this way, the *common-keys* effect is eliminated.

B. System Initialization

We use a sparse tree for facilitating the identification of a tag, as the example shown in Fig. 2. Let δ denote the branching factor of the key tree *S* and *d* denote the depth of *S*. For simplicity, we assume $\delta = 2$ in this paper. Note that the protocol can easily adopt any $\delta > 2$. We also assume that the system comprises of *N* tags $\{T_i\}$, $i \in [1, N]$, and a reader *R*. Each virtual node *j* in *S* has a location L_j. We define such a location as the *formal location* of a node. All the locations can be represented by binary strings with uniform length. A *formal location* contains three kinds of information, the depth of the node in the key tree, the sequence of the node at its level, and a direction to reach the *formal location*. RWP employs the bitstring to represent the location information. Specifically, the depth part needs at least $\lceil \log_2 d \rceil$ bits and the sequence part needs at least $\lceil \log_2 \delta^d \rceil$ bits.

There are six tags in the system depicted in Fig. 2. Node *P*'s *formal location* is 10 110, where the first 2 bits 10 indicates the depth of node *P* is 2. In RWP, the depth part of a *formal location* indicates how many bits are used in the sequence part. For example, the depth part 10 implies that the sequence part of *P*'s location constrains only two bits 11. Therefore, the latter substring 110 means that the node stays at the fourth position at the level 2.

Besides the depth and sequence parts, the direction part helps to reach a leaf node related to the tag. In this part, we use '0' to denote the left subtree and '1' to denote the right subtree. For the example in Fig.2, the tag T_1, has 11 110 as its *formal location* and the last bit '0' is the direction part. According to its *formal location*, we first locate the node *P*, then walk down to the most left node in the subtree of *P*, and finally reach the leaf node related to T_1.

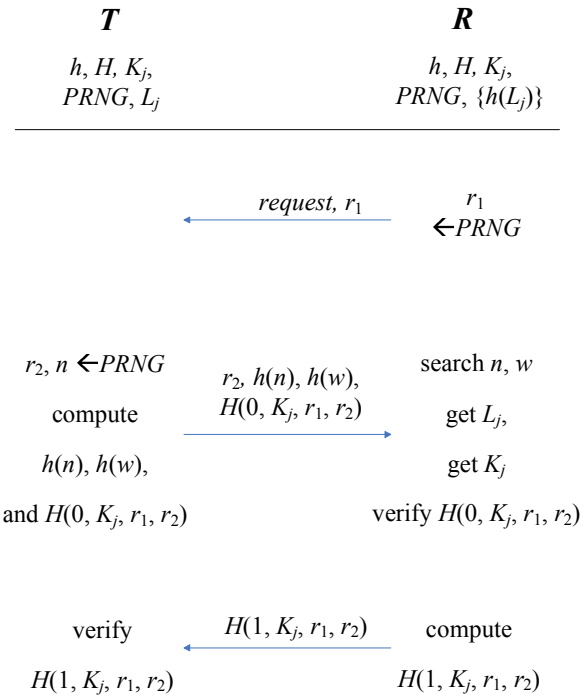

Figure 3. The authentication procedure of RWP.

Initially, the leaf nodes in tree *S* are empty. *R* first constructs the mainframe of the sparse tree, including the non-leaf nodes and leaf nodes. *R* then generates a random key for each leaf-node, and randomly assigns the *N* tags to *N* leaf nodes. Each T_j has a pair of (L_j, K_j), where L_j is its *formal location* in *S* and K_j is the key assigned to T_j. For example, the leaf node assigned to T_4 contains the pair of (11 011, K_4).

C. Authentication

The authentication process comprises of three steps, as illustrated in Fig. 3.

Step 1: When tag T_j enters the detecting range of *R*, *R* sends a query message as (*request,* r_1) to it, where r_1 is a nonce.

Step 2: Upon *R*'s authentication query, T_j sends a response message $I = (h(n), h(w), H(0, K_j, r_1, r_2))$ to *R*, where r_2 is a nonce, *n* represents the *anchor node* in the form of *formal location*, *w* is the offset between the location of the leaf node assigned to T_j and the location of the *anchor node*, K_j is the key assigned to T_j, and $H(0, K_j, r_1, r_2)$ is the hash value computed with the inputs of K_j, r_1, and r_2. When *R* receives *I*, it first searches $h(n)$ and $h(w)$ in the backend database to get the appropriate *n* and *w*. To save storage space while enabling $O(1)$ lookup, we employ CuckooHash [28] to manage the hash values of *formal locations*. Using *n* and *w, R* recovers the *formal location* of T_j, and then uses the K_j store at *R* to check whether the hash value $H(0, K_j, r_1, r_2)$ is equal to the hash value in *I*. If yes, T_j is legitimate. Note the use of K_j also helps to resolve the problem of multiple locations caused by hash collisions. When hash collision happens, the hash value $h(n)$ or $h(w)$ may be correlated to multiple *formal locations* in the

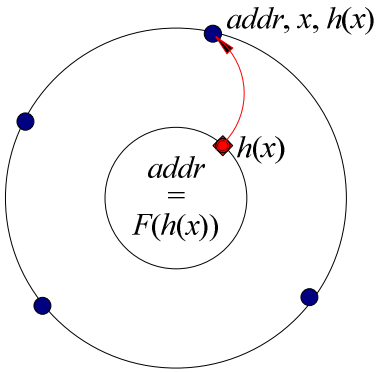

Figure 4. *Formal location* and offset recovery.

CuckooHash table. Utilizing K_j, the tag correspond to the correct *formal location* can be eventually authenticated.

The recovery process of n and w is as follows. Typically, a hash function maps a key to a hash value. Based on CuckooHash, we use two hash functions instead of one. Assume x is a key in the key space of $h(\cdot)$. We compute all hash values of the key spaces. Meanwhile, each key x, as well as its hash value $h(x)$, will gain a record. Suppose the address of this record is *addr*. $F(x)$ is a carefully selected hash function to correlate each $h(x)$ to the address of the record {*addr*, x, $h(x)$}. All the records are stored at R, as illustrated in Fig. 4.

During authentication, R receives a hash value $h(n)$ from the tag's response. R then computes the address of the record {*addr*, n, $h(n)$} using $F(h(n))$. For w, the processing is similar. The computational complexity of such an operation is $O(1)$.

Step 3: T_j checks the validation of R. R sends a message C with the value of $H(1, K_j, r_1, r_2)$ for T_j's checking. The entire authentication procedure is shown in Algorithm 1.

We use the example shown in Fig. 2 to illustrate the authentication procedure. After receiving a query message, tag T_4 generates a random number n = 10 110 and the corresponding w. The *formal location* of T_4 is 11 011. The depth of this *formal location* is 11. The first part of w is (11 – 10) mod 100 = 01. The second part of w is (011 – 110) mod 1000 = 101. R acquires $h(n)$ and $h(w)$ from the response sent by T_4. Based on two hash values, R is able to find the *formal location* of T_4, that is, identify T_4.

Using Algorithm 1, R can easily recover n from $h(n)$ and w from $h(w)$. To find the location where the n should be mapped, R locates the *anchor node* P from the n. In this case, n = 10 11 implies that the fourth node at the level 2 in the tree is the *anchor node*. The next step is to find the way from the *anchor node* P to T_4 according to w. Note that w is divided into two parts. We denote them as the vertical offset and horizontal offset components. In this example, the two offset components are 01 and 101, respectively.

With the direction of two offset components, R navigates from P to T_4 in the tree S as follows. R locates the *anchor node* P based on the n, 10 110. R starts from the *anchor node* P, and then goes to the leaf node of T_4. Note that the path is

Algorithm 1: *Authentication*

$T_j \rightarrow R$: *request*, r_1

T_j : generate r_2, n

compute $w = (L_j.depth - n.depth) \bmod (d)$

$\| (L_j.tail - n.tail) \bmod (\delta^d)$

$R \leftarrow T_j$: $h(n)$, $h(w)$, $H(0, K_j, r_1, r_2)$

R : Check if $h(n)$ and $h(w)$ are in database

and $n.depth + w.depth = d$

If exist compute $L_j = (n.depth + w.depth)$

$\| (n.tail + w.tail) \bmod (\delta^d)$

Find the tag relative to L_j, get its key K_j'

Check if $H(0, K_j, r_1, r_2) = H(0, K_j', r_1, r_2)$

If match, accept T_j

$R \rightarrow T_j$: $H(1, K_j, r_1, r_2)$

T_j : Check $H(1, K_j, r_1, r_2)$

If match, accept R

Algorithm 2: *Updating*

T_j : $L_j' \leftarrow G(2, K_j, r_1, r_2)$

$K_j' \leftarrow H(3, K_j, r_1, r_2)$

delete L_j, K_j

R : $L_j' \leftarrow G(2, K_j, r_1, r_2)$

$K_j' \leftarrow H(3, K_j, r_1, r_2)$

Insert (L_j', K_j')

Clear the old (L_j, K_j)

determined by the vertical and horizontal offset components of w. In this case, these two values are 01 and 101, which indicate R has one vertical movement and five horizontal movements. At this point, the selection on the left or right string should be decided by the direction part of n. Here the direction part of n is '0'. Therefore, we need to move to left subtree of P in the vertical movement.

In RWP, all movements are top-down and from left to right. R first travels to T_1, and then walks to right '*circularly*' according to the horizontal offset component of w. Note that in the second move, R reaches the rightmost leaf. Then R goes to the leftmost leaf node at the same level, and continues the remaining moves. That is the meaning of '*circularly*' moving. Thus, the moving route is $P \rightarrow T_1 \rightarrow T_6 \rightarrow T_5 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$.

Correspondingly, R can calculate T_4's *formal location* l_4 = (10 + 01) || (110 + 101) mod 1000 = 11011 by Algorithm 1.

In short, identifying a tag is similar to walking in a key tree from a randomly chosen node to the leaf node assigned to the tag. The *formal location* and offset act as the index to navigate the movements. Then tag T_j is authenticated by checking K_j.

Note that the identification procedure does not always need the above step by step movements. The *formal location* can be computed using Algorithm 1 and the computational complexity is $O(1)$. Combining the lookup operation, the total computational cost of RWP's authentication is still $O(1)$.

D. Updating process

The updating process of RWP has two goals. First, RWP employs key-updating to protect the forward secrecy for tags. Without key-updating, if a tag is compromised, the keys stored in it would be exposed to adversary. The adversary is then able to use the keys to recover all messages sent by the tag before, and the forward secrecy is compromised. Second, the *formal location* of the tag should also be updated with the concern of compromising attacks. The updating process enables an unfixed key structure for RWP. Each tag has a new *formal location* after the updating process, and this location has no correlation with those of other tags. Therefore, there is no key component shared among tags to facilitate compromising attacks. RWP also protects the forward secrecy using a one-way hash function to update keys. Due to the one-way property, the probability is negligible for attackers to break the hash function and hence deduce the old keys or *formal locations* from current ones. We will show the effectiveness of RWP in

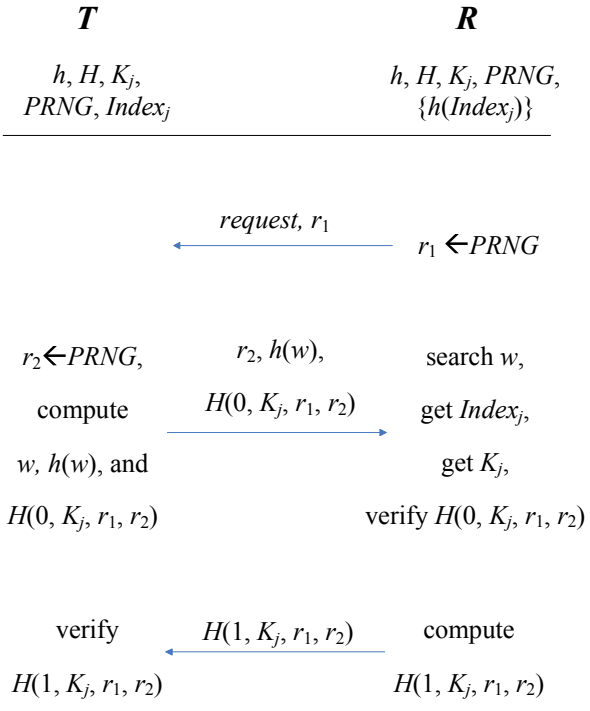

Figure 5. Random walk in a dataset.

providing forward secrecy and defending against the compromising attacks in Section V.

After a successful authentication, the reader and the tag start the updating process, as shown in Algorithm 2.

In RWP, the K_j generated in one updating round will be used as the input to compute the new location in next updating round. For any tag T_j, its new *formal location* is the hash value $G(2, K_j, r_1, r_2)$, and the new key is $H(3, K_j, r_1, r_2)$. RWP first uses the hash function $H(\cdot)$ to compute the hash value $H(2, K_j, r_1, r_2)$. This value, however, cannot be directly used as the *formal location* L_j. $G(\cdot)$ is a modified $H(\cdot)$. The output of $H(\cdot)$ can be divided into three fragments $dep\|seq\|else$, where the fragments *dep* and *seq* are in the same length as the depth and sequence parts in a *formal location*, respectively. The $G(\cdot)$ then further processes these two fragments as (*dep* mod d) and (*seq* mod δ^{dep}), which meets the requirements of the *formal location* format.

In RWP, some tags may have identical *formal locations*. Because the keys of those tags are independent from each other, the overlapped *formal locations* will not degrade the privacy of users. The above updating process only involves two hash operations and hence the computational cost is $O(1)$.

E. System maintenance

When a tag T joins the system, R randomly assigns T a leaf node and a key. Then R stores the *formal location* of the leaf node and the key assigned to T. In RWP, the location of a newly joined tag is randomly chosen in the key tree.

For the safety reason, any tag leaving the system should be deleted from the key tree together with its key.

F. Improvements

We further reduce the storage space required by RWP. The improvements aim at fully utilizing the entire key space. We propose the improvement based on the non-tree architecture shown in Fig. 5.

Given a key space S', we assign each tag T_j a key K_j as well as the tag's index $Index_j$. During authentication, the tag T generates a random w according to the index and the random number r_2 received from R. RWP performs a XOR operation on the r_2 and the index of the tag to generate w.

T sends the response message $I' = (r_2, h(w), H(0, K_j, r_1, r_2))$ to R. Having the response, R searches a w whose hash value is a match with the one in I'. With the pair of w and r_2, R computes the index, and locates the K_j. R further verifies $H(0, K_j, r_1, r_2)$. If the verification succeeds, R sends a hash value $H(1, K_j, r_1, r_2)$ to the tag for T's verifying. If both verifications succeed, the mutual authentication is successfully completed, and the updating process can follow. Indeed, the second improvement of RWP only uses w instead of (n, w) to identify the tag. The benefit is that authenticating a tag only needs one hash computation, and the search cost is reduced 50% at the reader side. Meanwhile, the storage space can also be reduced. For the example in Fig.2, the original RWP requires 5 bits to represent the tag's *formal location*. Using this advanced

version, representing a *formal location* needs only 3 bits. This shorter coding method thereby saves storage space.

V. ANALYSIS

In this section, we discuss the security guarantee of RWP. We first show RWP achieves the fundamental security requirements proposed in Section III. We then demonstrate the ability of RWP in defending against mainstream attacks.

A. Fundamental Defense Measurements

Privacy protection: In RWP, all messages related to the private information are encrypted. Furthermore, eavesdroppers cannot find out any dissimilarity among the messages because they follow the same type of probabilistic distribution statistically [29]. The brute force attack is also difficult to take effect due to the one-way property of hash function.

Cloning attack: In a cloning or spoofing attack, an adversary first queries a tag and records the response. Later, the adversary sends the response to a reader for some specific benefit. For example, the adversary may record the information of a cheap item from a tag, and then make a spoof of a much expensive one.

In RWP, we make use of nonces to defend against cloning attacks. In each authentication procedure, the tag and reader exchange a nonce with each other, and the nonces are used as inputs to generate the hash values $H(0, K_j, r_1, r_2)$ and $H(1, K_j, r_1, r_2)$. The attackers cannot directly utilize previous messages to perform cloning attack, because they are unable to know the nonces used in the current authentication procedure in advance. Thus, it is impossible for attackers to make spoofs.

Forward secrecy: The adversary might obtain the tag's key after successfully compromising it. If the key has never been updated, the adversary can use the key to recover the messages sent before. Some existing works suffer from such attacks due to the lack of updating [8, 26]. In RWP, a tag updates its key after each successful authentication. The updating process of RWP computes the hash values of old keys as the new keys. In this process, the probability is negligible for attackers to deduce the previous keys from current ones. Therefore, RWP guarantees the forward secrecy for tags.

Tracking: As the tree-based structure is used in RWP, a security concern about the tree structure is whether RWP has the shared key components like normal tree-based protocols, which exposes a flaw to attackers' tracking. In the key tree of RWP, the non-leaf nodes only serve for navigation. There is no shared key existing in the key tree. Also, the specific index for a tag is hidden in the response message. Therefore, there is no shared information among tags that can be utilized by the adversary. If a new tag is assigned to a position where a leaving tag just stayed, this may lead to a potential flaw. Attackers can make use of this assignment pattern to track a newly joined tag in normal tree-based methods. This flaw, however, does not exist in RWP, as the newly coming tag is randomly assigned to a *formal location* that the attackers cannot predicate.

B. Compromising Attack

As in illustrated in [18], tree-based approaches suffer from compromising attacks because of the key components shared among tags. As we mentioned in Section III, a compromising attack mainly depends on the differences between the victim tag T and other tags. We eliminate the impact of compromising attacks from two aspects.

First, the number n in the tag's response is generated randomly. Correspondingly, the value of w varies at each authentication. The hash values of these two parameters thereby vary at each authentication procedure. The probability that the adversary generates a number that is equal to n or w is $1/2^L$, where L is the number of bits in the *formal location*. If the adversary knows n and w at a certain round, she/he knows the current *formal location* of the tag. Following RWP, the tag gains an updated *formal location* after an authentication procedure, and the n and w are changed at the next authentication procedure. Therefore, a tag's messages in different authentication procedures are not related. This property impedes tracking.

Second, since each tag has a distinct key that is not shared with other tags, compromising some tags is useless for an adversary to deduce the keys of other tags. Furthermore, the reader employs two nonce numbers as inputs to update the key for each tag. The adversary can only break those keys by using brute force attacks. If the length of the keys is long enough, the system is sufficiently secure.

C. Desynchronization Attack

The *desynchronization attack* usually focuses on synchronization based RFID approaches, but it also has a serious impact on any RFID authentication protocols when combined with the tracking attacks.

When adopting the updating scheme, the RFID system faces a dilemma, in which the tag cannot simultaneously resist to the desynchronization and tracking attacks. The normal way for defending against the desynchronization attack is to force the tag to update its keys if and only if it finishes a successful authentication. Otherwise, the keys used by the tag and the reader are desynchronized when the tag receives incomplete authentication queries. On the other hand, updating the tag's keys after a successful authentication cannot resist to tracking attacks. A malicious reader can iteratively interrogate the tag with incomplete authentication queries. The response messages from the tag are similar since the tag does not update its keys. Based on those recognizable messages, the attacker can track the tag.

For example, the triggered hash chains proposed in [22] employs the authentication procedure shown in Fig. 6. In this procedure, the tag updates the keys only after it successfully finishes an authentication with the reader. The reader holds each tag's ID. During the authentication, the reader sends a request to the tag first. The tag sends a hash value as g(ID) to the reader as the response. The reader thereby obtains the ID based on the g(ID). Then the reader uses another hash function to compute a hash value h(ID) and delivers it to the tag. The

tag checks the h(ID). If the result is correct, the tag updates the ID as ID← f(ID), where $f(\cdot)$ is the hash chain function.

We assume the ID of a given tag T_i is id. The attacker can send the tag with n incomplete authentication queries, denoted as r_1, r_2, …, r_n. Since the attacker is unable to obtain the original id from $g(id)$, it cannot send a correct *updauth* message. Thus, the ID of T_i does not change after receiving the n queries. The $g(id)$ in the response messages is also unchanged. The attacker can easily recognize and track the tag based on those messages.

In contrast, RWP allows a tag to change the response upon each authentication query. A tag in RWP introduces a nonce r_2 to generate the response message, so the value of $h(w)$ cannot be recognized. Meanwhile, the tag updates the keys only after a successful authentication. In addition, we have shown that RWP is resilient to compromising attack in subsection V.B. These methods effectively protect RWP from the compromising and desynchronization attack simultaneously.

If the adversary interrupts the response from the reader to the tag, a desynchronization may incur. To deal with this problem, RWP keeps a temporary cache for each tag in the backend database to store the last *formal location* and key successfully passed the verification. If a tag can provide this information, the reader will also accept the tag.

D. Performance

To evaluate the performance of RWP, we compare the computing complexity, storage cost, and search efficiency of RWP with those of Hash-chain [10] and Dimitriou's protocol [8] approaches. Hash-chain is a typical non-tree-based approach which adopts the linear storage and search. Dimitriou's protocol [8] is the representative of tree-based approaches.

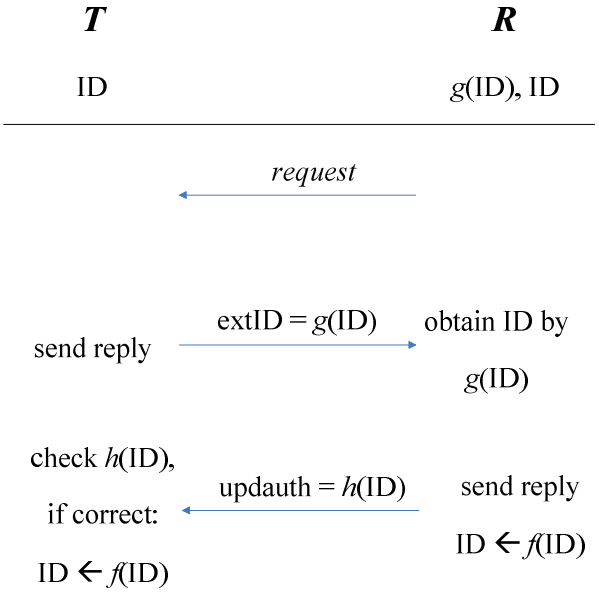

Figure 6. Triggered hash chains protocol.

Table II shows that RWP reduces the computing complexity and storage cost to $O(1)$ for tags. This improvement significantly reduces the hardware demand on tags. RWP also reduces the computing complexity to $O(1)$ on the reader side, and achieves a $O(1)$ search.

RWP introduces extra storage cost into the backend application. The coding of *formal location* requires adding a head to each tag's record. RWP needs at least $\lceil \log_2 \log_\delta N \rceil + \lceil \log_2 N \rceil$ bits for coding N tags. The increased storage, however, is affordable for backend application. Considering a RFID system with 2^{32} tags, RWP requires 18.5GB in the backend server, which is acceptable for current mainstream computers.

TABLE II. PERFORMANCE COMPARISON

Cost		**(1)**	**(2)**	**(3)**
Average computing cost	***Tag***	$H+g$	$H \cdot d$	$2h+4H$
	Reader	$N \cdot H/2$	$\delta \cdot d \cdot H/2$	$4H+2f$
Storage cost	***Tag***	l_k	$d \cdot l_k$	$l_k + L$
	Reader	$N \cdot l_k$	$N \cdot d \cdot l_k$	$d \cdot \delta^d \cdot (L + l_k)$
Search cost	***Reader***	$N \cdot a/2$	$\delta \cdot d \cdot a/2$	$3a$
Number of communications (considering updating)	***Tag & Reader***	2	3	3
Length of authentication message	***Tag***	l_k	$l_n + d \cdot l_k$	$l_n + 3l_k$

Remark: (1): Hash-chain [10]; (2): Dimitriou's protocol [8]; (3): RWP.

N: number of tags in the system

d: depth of the key tree;

δ: branching factor of the key tree;

h: cost of encrypt function for *formal location* $h(\cdot)$;

H: cost of encrypt function $H(\cdot)$,

f: cost of localization function $F(\cdot)$;

g: cost of updating function $g(\cdot)$ in Hash-chain;

l_k: length of a key and the output of $h(\cdot)$, $H(\cdot)$;

L: length of the *formal location*;

l_n: length of a random nonce;

a: one time search cost.

VI. Conclusions and Future Work

We present a secure and efficient private authentication protocol, called RWP, based on random walks in the tree-based key structure. RWP enhances the privacy protection for RFID systems by eliminating the key components shared among tags. The comprehensive analysis demonstrates that RWP outperforms existing works in terms of both security and efficiency. Our future work includes further reducing the costs of storage, increasing scalability, and seeking an optimal trade-off between the authentication latency and storage required.

Acknowledgments

We would like to thank the shepherd, Marcel C. Rosu, for his constructive feedback and input. We also thank Dr. Li Lu for reading the paper and his valuable suggestions.

References

[1] P. De, K. Basu and S. K. Das, "An Ubiquitous Architectural Framework and Protocol for Object Tracking using RFID Tags", in Proceedings of ACM MobiQuitous Networking Conference, 2004.

[2] M. Mamei, F. Zambonelli, "Pervasive Pheromone-based Interactions with RFID Tags", *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 2 , Iss. 2, June 2007.

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

[4] N. Ravi, C. Narayanaswami, M. T. Raghunath, M. C. Rosu, "Securing Pocket Hard Drives", *IEEE Pervasive Computing*, Vol. 6 , Iss. 4, October 2007, pp. 18-23.

[5] W. Gu, X. Bai, S. Chellappan, D. Xuan and W. Jia, "Network Decoupling: A Methodology for Secure Communications in Wireless Sensor Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No. 12, December 2007, pp. 1784-1796.

[6] Q. Cao, T. Abdelzaher, T. He, and R. Kravets, "Cluster-Based Forwarding for Reliable End-to-End Delivery in Wireless Sensor Networks", in Proceedings of INFOCOM, 2007.

[7] Z. Jiang, J. Ma, W. Lou, and J. Wu, "An Information Model for Geographic Greedy Forwarding in Wireless Ad-Hoc Sensor Networks", in Proceedings of INFOCOM, 2008.

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

[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 IEEE PerCom, 2007.

[10] M. Ohkubo, K. Suzuki and S. Kinoshita, "Cryptographic Approach to Privacy-friendly Tags," in Proceedings of RFID Privacy Workshop, MIT, 2003.

[11] G. Avoine, "Bibliography on Security and Privacy in RFID Systems", LASEC Security and Cryptography Laboratory, available online at http://lasecwww.epfl.ch/~gavoine/rfid, 2007.

[12] M. Lehtonen, T. Staake, F. Michahelles, and E. Fleisch, "From Identification to Authentication – A Review of RFID Product Authentication Techniques", in Proceedings of Workshop on RFID Security (RFIDSec), 2006.

[13] G. Avoine, "Cryptography in Radio Frequency Identification and Fair Exchange Protocols", Ph.D. thesis, EPFL, Lausanne, Switzerland, 2005.

[14] A. Juels, "RFID Security and Privacy: a Research Survey", *Selected Areas in Communication*, *IEEE Journal on*, Vol. 24, No. 2, 2006, pp. 381-394.

[15] G. Roussos and V. Kostakos, "RFID in Pervasive Computing: State-of-the-art and Outlook", *Pervasive and Mobile Computing*, Elsevier, 2008.

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

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

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

[19] D. Henrici and P. Müller, "Hash-based Enhancement of Location Privacy for Radio-Frequency Identification Devices Using Varying Identifiers," in Proceedings of Pervasive Computing and Communications Workshops, 2004.

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

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

[22] D Henrici, P Müller, "Providing Security and Privacy in RFID Systems Using Triggered Hash Chains", in Proceedings of IEEE PerCom, 2008.

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

[24] W. Wang, Y. Li, L. Hu, L. Lu, "Storage-Awareness: RFID Private Authentication based on Sparse Tree", in Proceedings of Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2007.

[25] T. L. Lim, T. Li, and S. L. Yeo , "Randomized Bit Encoding for Stronger Backward Channel Protection", in Proceedings of IEEE PerCom, 2008.

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

[27] C. Tan, B. Sheng, and Q. Li, "Serverless Search and Authentication Protocols for RFID", in Proceedings of IEEE PerCom, 2007.

[28] U. Erlingsson, M. Manasse, and F. McSherry, "A Cool and Practical Alternative to Traditional Hash Tables", in Proceedings of WDAS, 2006.

[29] M. Shao, Y. Yang, S. Zhu, and G. Cao, "Towards Statistically Strong Source Anonymity for Sensor Networks," in Proceedings of INFOCOM, 2008..