Tree-based HB Protocols for Privacy-Preserving Authentication of RFID Tags

Tzipora Halevi　　Nitesh Saxena
Polytechnic Institute of New York University
thalev01@students.poly.edu, nsaxena@poly.edu
Tel/Fax: 718-260-3116/3609

Shai Halevi
IBM Research, NY, USA
shaih@alum.mit.edu
Tel/Fax: 914-784-7653/6205

Abstract

An RFID reader must authenticate its designated tags in order to prevent tag forgery and counterfeiting. At the same time, due to privacy requirements of many applications, a tag should remain anonymous and untraceable to an adversary during the authentication process. In this paper, we propose an "HB-like" protocol for *privacy-preserving authentication* of RFID tags. Previous protocols for privacy-preserving authentication were based on PRF computations. Our protocol can instead be used on low-cost tags that may be incapable of computing traditional PRFs. Moreover, since the underlying computations in HB protocols are very efficient, our protocol also reduces reader-side load compared to PRF-based protocols.

We suggest a tree-based approach that replaces the PRF-based authentication from prior work with a procedure such as HB+ or HB#. We optimize the tree-traversal stage through usage of a "light version" of the underlying protocol and shared random challenges across all levels of the tree. This provides significant reduction of the communication resources, resulting in a privacy-preserving protocol almost as efficient as the underlying HB+ or HB#. We also present analytical and simulation results comparing our method with prior proposals in terms of computation, communication and memory overheads.

Keywords: HB Family Protocols, Privacy-Preserving Authentication.

1 Introduction

Radio Frequency Identification (RFID) technology is increasingly used in many aspects of daily life. Low-cost RFID devices have numerous applications in military, commercial and medical domains. They are already being used widely in supply-chain management (such as retail check-out), libraries, access control systems, payment systems, medical records, animal tracking, automobile immobilizers, and most recently, in electronic passports and driver's licenses. In most of these applications, an RFID reader must authenticate its designated tags to prevent tag forgery and counterfeiting. At the same time, privacy concerns in many of these applications dictate that the tag's identity is not revealed to an attacker, even while the tag authenticates itself (and even if the attacker pretends to be a reader).

RFID tags are typically very low-cost devices and their computation and storage capabilities are severely constrained. Hence, traditional authentication protocols based on Pseudo-Random Functions (PRFs) may not be applicable on such tags. This calls for simple authentication protocols that can be accommodated within the limited resources present on low-cost tags. To this end, Juels and Weis [12] suggested using the HB protocol (of Hopper and Blum [11]) for tag authentication. Several enhancements – such as HB+ [12] and HB# [8] – were proposed later, and are currently the only viable solutions for very low-cost authentication.

Further complicating RFID authentication is the fact that a single reader must often handle a very large tag population, possibly in the millions or even billions. Clearly, to use the appropriate secret keys, the reader must learn the identity of a tag that is trying to authenticate itself. Although the tag can identify itself by sending a pre-shared unique identifier, such identifiers can easily be used by an eavesdropping adversary to compromise the privacy and anonymity of the tags (and thus also of their bearers). Therefore, a privacy-preserving solution is needed to allow the reader to identify the tag.

One possible approach to achieve *privacy-preserving authentication* is for the reader to perform an exhaustive search, trying to match the result of the tag being authenticated against all possible tags. However, this requires $O(N)$ computation for a population of N tags. To address this scalability problem, Molnar et al. [17] proposed using a "tree of keys", assigning to each tag the keys corresponding to a root-to-leaf path in the tree, and using these keys as seeds for a PRF; the PRF-based response of a tag is then linked to this path, and this link is used to identify the tag. This solution reduces the reader computational complexity $O(log(N))$. Recently, Cheon et al. [6] proposed a 2D-mesh scheme that enables the reader to identify the tag with complexity of $O(\sqrt{N}\log N)$: The idea is to create two sets of PRF keys, each with $\sqrt{N}$ keys, and give every tag one key from each of the two sets, such that the combination of these two keys uniquely determined that tag. In [4], Burmester et al. also utilized PRF functions for an anonymous RFID authentication scheme with constant lookup. In this approach, the reader keeps for every tag a constant number of pseudonyms. At the end of each successful authentication session the tag generates a new pseudonym, which the reader then uses to identify the tag from its tag lookup pseudonyms table.

All the above proposals are based on protocols that require the tag to compute PRF operations, and therefore are not applicable to very low-cost tags. To this end, we consider the HB family of protocols for privacy-preserving authentication. Adapting HB-like protocols to this setting takes some care, however. For example, HB protocols are typically designed with very tight parameters, which yield a non-negligible false accept rate (FAR), i.e., the probability of an illegitimate tag successfully authenticating to the reader. In an exhaustive search, the overall false accept rate grows roughly by a factor of N. For example, the HB+ protocol with 80 rounds and noise probability of 0.25 has FAR of 4×10^{-6}. When using this protocol, with an exhaustive search over a population of a million tags, the overall FAR would be close to one, which is clearly unacceptable for any practical use.

1.1 Our Contributions

In this work, we develop a tree-based protocol for privacy-preserving authentication. Specifically, we study the use of HB-like protocols (such as HB+ or HB#). This is useful for two reasons: First, for low-cost tags, not capable of efficiently performing PRF operations, HB protocols are currently the only viable solution for authentication. Second, underlying computations in HB protocols are extremely efficient which also helps naturally reduce reader load, even when compared to tree-based protocols for tags capable of performing PRFs.

Our tree-based protocol has two logical stages. First there is a *tree traversal stage*, in which the reader tries to identify the most likely tag to authenticate, and then there is an *authentication stage*, in which the identity of that "most likely tag" is verified. Unlike exhaustive search, the false-accept rate of this protocol does not grow with the number of tags in the population, since only one tag is authenticated in the last stage. We also show that the false-reject rate grows very slowly, despite the presence of a large number of tags. We propose several optimizations over naive use of HB+/HB# in a tree-based scheme. These optimizations bring the communication of the tree-based protocol down to not much more than the underlying HB+/HB# authentication protocol, and also improve the computation time.

Our work is based on the observation that the function $f_x(A) = Ax + noise$ for a random matrix A behaves like a (randomized) pseudo-random function (which was proven in [2, Section 5] under the assumption that learning parity with noise is hard). Therefore we replace the standard PRFs during the tree-traversal stage with this "HB-like" function. We observe that this function only needs tag-generated random challenges during the tree-traversal stage. (It is only in the authentication stage that a reader-generated challenge is needed.) Moreover, there is no need to generate a new challenge in every level, it is perfectly safe to use the same random challenge in all the levels of the tree, and we can use significantly smaller parameters during the tree-traversal stage. The combination of these observations reduces computation and communication by a factor of $2\times$ to $4\times$ (In fact our tree protocol is almost as efficient as the underlying HB+ or HB#). We present analytical and simulation results comparing our method with prior proposals in terms of computation, communication and memory overheads.

We note that just like all other HB-type protocols, ours is also vulnerable to man-in-the-middle attacks [9, 19]. Also, just like other tree-based protocols, its privacy guarantees degrade somewhat when the secret keys of many tags are exposed to the adversary.

This paper is an extension of the paper presented at the Fifth Workshop on RFID Security (RFID-Sec'09) [10]

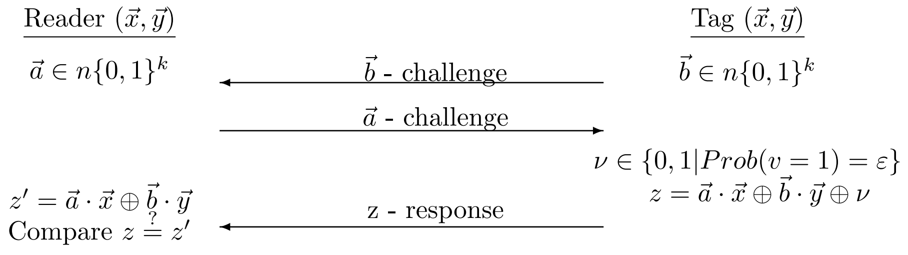
1.2 Paper Organization

The rest of this paper is organized as follows. We give an overview of HB, HB+ and HB# protocols in Section 2. In Section 3, we propose the use of HB family of protocols for privacy-preserving authentication and discuss their security in Section 2.1. We show, in Section 5, that the resulting error rates are comparable to that of the underlying protocols. In Section 5.6, we discuss the choice of parameters to be chosen for practical use of our proposal, followed by our simulation results in Section 6.

2 Background: HB Series Protocols

In the original HB protocol by Hopper and Blum [11], the reader and the tag share a secret $\vec{x}$ which is a k-bit vector (e.g. $k = 256, 512$, etc.). The protocol consists of many rounds. In each round the reader sends a random challenge vector $\vec{a}$ and the tag replies with one bit $z = \vec{a} \cdot \vec{x} \oplus \nu$, where ν is 1 with probability ε (for some parameter $\varepsilon < 0.5$). After r such rounds, the tag is accepted if z matches $z' = \vec{a} \cdot \vec{x}$ at least $r - \tau$ times, where τ is some threshold parameter.

The HB protocol is vulnerable to attacks by an active adversary, who can send to the tag the same challenge $\vec{a}$ many times, thereby eliminating the noise and recovering the secret $\vec{x}$. To counter this attack, the HB+ protocol by Juels and Weis [12] added a second secret $\vec{y}$ that is shared by the reader and the tag. In each round of the protocol, a random challenge $\vec{b}$ is generated by the tag, a second challenge $\vec{a}$ is generated by the reader, and the tag sends to the reader $z = \vec{a} \cdot \vec{x} \oplus \vec{b} \cdot \vec{y} \oplus \nu$. Namely, a single round of HB+ proceeds as follows:


Reader $(\vec{x}, \vec{y})$
Tag $(\vec{x}, \vec{y})$
$\vec{a} \in n\{0,1\}^k$
$\vec{b}$ - challenge
$\vec{b} \in n\{0,1\}^k$
$\vec{a}$ - challenge
$\nu \in \{0, 1 | Prob(v = 1) = \varepsilon\}$
$z = \vec{a} \cdot \vec{x} \oplus \vec{b} \cdot \vec{y} \oplus \nu$
$z' = \vec{a} \cdot \vec{x} \oplus \vec{b} \cdot \vec{y}$
Compare $z \overset{?}{=} z'$
z - response

As before, the tag is accepted after r rounds if z matches $z' = \vec{a} \cdot \vec{x} \oplus \vec{b} \cdot \vec{y}$ at least $r - \tau$ times. It was later shown in [13, 14] that all the rounds can be carried out in parallel: The tag chooses a matrix B, the reader responds with a matrix A, the tag chooses a noise vector $\vec{\nu}$ and replies $\vec{z} = A \cdot \vec{x} \oplus B \cdot \vec{y} \oplus \vec{\nu}$, and the reader checked that $\vec{z}$ is close enough to $\vec{z'} = A \cdot \vec{x} \oplus B \cdot \vec{y}$. This protocol can withstand an active attacker that interacts with the tags, but it is still vulnerable to man-in-the-middle attacks [9, 19].

The HB# protocol [8] is a variant where the "types" of the secrets and the challenges are swapped: Namely, the tag and reader share secrets X and Y which are $r \times k$ matrices, the challenges are r-vectors $\vec{a}$ and $\vec{b}$, and the reply that the tag computes is $\vec{z} = \vec{a} \cdot X \oplus \vec{b} \cdot Y \oplus \vec{\nu}$. To alleviate the additional storage requirement, HB# proposes using Toeplitz matrices for X, Y. Another claimed benefit for swapping the types is that HB# can resist some limited form of man-in-the-middle attacks, however, not all [19].

2.1 Security

The security of the HB series of protocols relies on the computational hardness of the problem of *Learning from Parity with Noise* (LPN) [12]. Roughly, the LPN problem is to find a secret k-vector

$\vec{x}$ given a random binary $r \times k$ matrix A and a vector $\vec{z} = A \cdot \vec{x} \oplus \vec{\nu}$, where ν is a noise vector in which each entry is set to 1 with probability ε (for a parameter $0 < \varepsilon < \frac{1}{2}$).

The hardness of this problem (and thus the security level of the protocols) depends on the choice of the parameters ε and k. The best algorithm for learning parity with noise is due to Blum, Kalai and Wasserman [5], with some later optimizations, notably by Levieil and Fouque [15]. Based on the performance of the BKW algorithm, Juels and Weis suggested in [12] that setting $k = 224$ and $\varepsilon \in [\frac{1}{8}, \frac{1}{4}]$ yields an 80-bit security level. In [15], Levieil and Fouque claimed, however, that these parameters yield only a 55-bit security level, and that with $\varepsilon = \frac{1}{4}$ one needs to use $k = 512$. (They also noted that only one of the secrets $\vec{x}, \vec{y}$ in HB+ needs to be this long, while the other can safely be only 80-bit long.)

The other parameters of interest in HB-type protocols are the number of rounds[1] and the acceptance threshold. Juels and Weis suggested in [12] to use number of rounds $r \in [40, 80]$ and set the threshold at $\tau = \varepsilon r$. This gives a relatively high false reject rate (FRR) of 0.44, and when used with error probability $\varepsilon \in [\frac{1}{8}, \frac{1}{4}]$ it gives false-accept rates (FAR) between 10^{-3} and 10^{-9}. (The HB# proposal in [8] suggested other parameters with a much lower FRR and FAR, e.g., FAR $\leq 2^{-80}$ and FRR $\approx 2^{-45}$.)

3 Private Authentication with an HB-like protocol

Recall that in our setting, we have a system with many tags under one administrative domain. Similarly to [17], we arrange all the tags in a tree structure and use this structure to find the right tag to authenticate. However, we use an HB-like procedure to implement both the tree search and the tag authentication as opposed to the PRF-based solution in [17].

A naive approach would simply perform the underlying authentication protocol at every level of the tree. Namely, at every level one can run HB+ (or HB#) and the reader then performs an exhaustive search over the children of the current node, selecting the one for which the authentication was successful. A closer inspection reveals that this naive approach can be significantly improved:

- First, there is no reason to use challenges from the reader while traversing the tree. Indeed, the challenge from the reader is needed for secure authentication, but *not* for the identification of "the right tag to authenticate".

[1]The name "number of rounds" refers to a sequential protocol where a one-bit z is returned in every round. For the parallel version, this parameter is the reply-size.

We therefore suggest to logically split the protocol into a *tree-traversal stage*, where the reader finds the right tag to authenticate, and an *authentication stage*, where the identity of that tag is verified. In the authentication stage, we just use the underlying HB+ protocol unchanged, but in the tree-traversal stage we only use random challenges from the tag. This optimization cuts the computation time and storage requirements for both the tag and the reader.

(We note, however, that in the asymmetric setting suggested in [15], where one secret is longer than the other, we need to store a long secret in every internal node in the tree. For example, using the 512/80 asymmetric setting, this optimization only saves 80 of the combined $512+80 = 592$ bits, which is $\approx 13\%$.)

- More importantly, we observe that we can share the same challenges across all the levels. Namely, instead of the tag sending a different challenge for every level, we send only one challenge and use it at all the levels. This means that we only have communication of two challenges throughout the tree-based protocol (just as in the underlying HB+), and the only additional communication is an r-bit response vector for every level.

In more details, the reader in our system maintains a tree structure with the tags at the leaves, and each internal node n in the tree is associated with a secret key $\vec{y_n}$, which is a binary vector of length k_y. A tag t, associated with a leaf in the tree, is given all the secret keys that are associated with nodes on the path from the root to that leaf. In addition the tag is also given two unique secret keys $\vec{x}_t$ and $\vec{y}_t$ (of length k_x and k_y, respectively) that are used for authentication. The notations that we use in the description below are summarized in Table 1(a).

3.1 System Setup

The system is setup with some upper bound N on the number of tags that it can support, and the parameters r, d, β are derived from this upper bound. (See Section 5.6 for a discussion of how to determine these parameters.) Once determined, the parameters d, β define a tree with $\beta^d \geq N$ leaves, which will be associated with the tags in the system. Also, a "master secret" is chosen, which will be used to determine all the other secrets in the system as described next.

3.2 Tag Registration

When a tag is registered into the system, it is associated with a random available leaf in the tree. This can be done either by the system administrator remembering the positions of all the tags in the tree and choosing at random one leaf that is not yet used, or by choosing a random permutation π over the domain $[1, \beta^d]$ and assigning the i'th tag in the system to leaf $\pi(i)$. [2]

Let $n_0, n_1, \ldots, n_d$ be the path in the tree from the root (denoted n_0) to the leaf that is associated with the new tag (denoted n_d). For each node n_i (except the root), the tag is given the k_y-bit secret $\vec{y}_{n_i}$ that is associated with that node. (This secret can be derived from the master secret, say by setting $\vec{y}_{n_i} = PRF_{ms}(n_i)$ for some pseudo-random function PRF that is keyed by the master secret ms.) The tag also gets two additional keys $\vec{x}_t$ and $\vec{y}_t$ of size k (that can similarly be derived from the master secret). After registration, the tag will hold the keys $\vec{y}_{n_1}, \ldots, \vec{y}_{n_d}, \vec{x}_t, \vec{y}_t$.

3.3 Tree-Traversal Stage

At the start of the authentication process, the tag needs to be identified so the correct keys can be used for the authentication. To this end, the tag chooses an $r \times k_y$ random challenge matrix B and noise vectors ν_i for every level i in the tree. The tag computes $\vec{z_i} = B \cdot \vec{y}_{n_i} \oplus \vec{\nu_i}$ and sends to the reader the matrix B and all the $\vec{z_i}$'s.

Starting from the root, the reader then goes down node by node, using the $\vec{z_i}$'s to decide what child of the current node it needs to use next. Specifically, for every child c of the root it computes $\vec{z}_c = B \cdot \vec{y}_c$, where $\vec{y}_c$ is the secret associated with that child. Then, the reader descends into the child c for which $\vec{z}_c$ is closest to the response vector $\vec{z}_1$ received from the tag. Similarly, after descending into an internal node n_i at level i, the reader computes $\vec{z}_c = B \cdot \vec{y}_c$ for every child c of n_i, and descends into the child c for which $\vec{z}_c$ is closest to $\vec{z}_i$ that was received from the tag. (Throughout this process, if two children are equally close to $\vec{z}_i$, one is chosen arbitrarily as the next node).

In the following description, we denote by Hwt the Hamming weight of a binary vector:

Algorithm 1, Tree Traversal Stage, single level

Input: β nodes with tag secrets $\vec{y}_i$ for each node, B matrix sent from Tag

Output: C_i Next node in the tree

Processing (Tag):

[2] An efficient method for implementing such a random permutation was recently proposed by Morris et al. [18].

```
1. Tag chooses $\nu$
2. Tag calculates $\vec{z_i} = B \cdot \vec{y}_{n_i} \oplus \vec{\nu_i}$
3. Tag sends reader $\vec{(}z)_i$
```

Processing (Reader):

```
1. $HW_{min} = r$
2. for $c = 1..\beta$ do
3.     $\vec{z}_c = B \cdot \vec{y}_c$
4.     $HW_c = \mathsf{Hwt}(\vec{z}_i \oplus \vec{z}_c)$
5.     if $(Hw_c < HW_{min})$
6.         $HW_{min} = Hw_c$
7.         $C_i = c$
8.     end if
9. end for
```

Algorithm 2, Tree Traversal Stage, multi-level

Input: (d, β) tree with secrets $\vec{y}_i$ for each node i in the tree

Output: l leaf ID

Processing:

```
1. Tag chooses $B$ and sends it to the reader,
2. for $i = 1..d$ do
3.     Call Algorithm 1 on input $B$, $\beta$ children with node secrets $\vec{y}_i$
4.     $\beta_i$ = output (algorithm 1)
5. end for (each level in tree)
6. $l = \beta_1, \beta_2 ... \beta_d$ (path to leaf)
```

The tree traversal stage is shown in Figure 1(a) (single-level) and Figure 1(b) (multi-level).

3.4 Authentication Stage

At the end of the Tree-traversal stage, the reader arrives at a leaf that it considers to be the most likely to be the one associated with the correct tag. At this stage, we need to run the authentication protocol to confirm that the tag is valid. Here we just use the parallel HB+ protocol. Specifically, the reader sends a random challenge matrix A, the tag chooses a noise vector $\vec{\nu}$, and reply with $\vec{z} = A \cdot \vec{x}_t \oplus B \cdot \vec{y}_t \oplus \nu$ (using the two last secret keys $\vec{x}_t, \vec{y}_t$ that it holds). The reader recovers the keys $\vec{x}, \vec{y}$ that are associated with the tag in this most likely leaf, computes $\vec{z'} = A \cdot \vec{x} \oplus B \cdot \vec{y}$, and checks that the Hamming distance between $\vec{z}$ and $\vec{z'}$ is below the threshold τ. It accepts the tag if $\vec{z}$ and $\vec{z'}$ are close enough, and rejects it otherwise.

3.5 Tree-HB+

The algorithm includes both the tree-traversal stage, in which one potential tag is chosen from all the tags managed by the reader and the authentication stage, in which the chosen tag secrets are used to authenticate the current tag using the HB+ algorithm. The overall algorithm is shown in Figure 1(c).

Algorithm 3, Tree HB+

```
Input: $(d, \beta)$ tree with secrets $\vec{y}_i$ for each node $i$ in the tree, $\vec{y}_l, \vec{l}$ for each leaf, $\tau$ = HB+ threshold
Output: Tag authentication/rejected by reader
Processing:
  1. Call Algorithm 2, $l$ = output (algorithm 2) (leaf ID)
  2. Perform one HB+ authentication round:
        Reader generates $A$ and sends it to tag
        Tag generates $B$, $\nu$
        Tag calculates $\vec{z} = A \cdot \vec{x}_t \oplus B \cdot \vec{y}_t \oplus \nu$
        Tag sends $\vec{z}$ to reader
        Reader calculates $\vec{z'} = A \cdot \vec{x} \oplus B \cdot \vec{y}$ for leaf $l$
        Hwt = Hwt($\vec{z'} \oplus \vec{z}$)
        if Hwt $< \tau$ then
           Output=1 (tag confirmed)
        else
          Output=0 (tag rejected)
        end if
```

3. Authenticate or Reject item

3.6 Tree-HB#: A Variant Based on HB#

The algorithm can be modified to use the HB# algorithm instead of HB/HB+. Namely, the secrets are Toeplitz matrices and the challenges are vectors. In this case, upon registration in the system the tag gets a set of secret keys $Y_{n_1}, \ldots Y_{n_d}$ corresponding to the nodes $n_1, \ldots, n_d$ on the path to the leaf of that tag, and additional two secrets X_t, Y_t, but this time each of these secret is an $k \times r$ Toeplitz matrix.

For the tree-traversal stage, the tag generates a random vector $\vec{b}$ of length k and noise vectors $\vec{\nu}_i$ of length r for all the levels of the tree $i = 1, \ldots, d$. Then the tag sends to the reader the vector $\vec{b}$ and also the "response vectors" $\vec{z}_i = \vec{b} \cdot Y_{n_i} \oplus \vec{\nu}_i$. The reader uses the exact same procedure from above to descend in the tree from the root to the "most likely" leaf, and then uses the HB# protocol to authenticate that tag. Namely, the reader generates a random k-vector $\vec{a}$ and sends it to the tag, the tag generates a noise vector ν and sends $\vec{z} = \vec{a} \cdot X_t \oplus \vec{b} \cdot Y_t \oplus \vec{\nu}$ to the reader, and the reader computes $\vec{z'} = \vec{a} \cdot X \oplus \vec{b} \cdot Y$ (using the keys X, Y of that "most likely" leaf) and accepts if $\vec{z}$ and $\vec{z'}$ are close enough.

4 Security Analysis

Recall that we have two security goals in our model, namely authentication and privacy. Our attack model allows the adversary to eavesdrop on the communication between tags and the reader, and also to communicate directly with the tag and the reader, but not to modify messages that are sent between them. In other words, we consider an active adversary, but explicitly disregard man-in-the-middle attacks (since all HB-type protocols are insecure against them [9, 19]).

4.1 Authentication

Our model for authentication is essentially the usual DET (Detection) model [12, 13, 8], where in our case we have one target tag that the attacker is trying to impersonate, and all the other tags in the system are considered to be adversarial. In more details, the attacker first gets the secrets of all the tags in the system but one, then it can interact with the remaining tag a finite number of times, and finally it interacts with the reader and tries to impersonate that remaining tag.

Since we just run the original HB+ protocol at the authentication stage, then trivially our Tree-HB+ protocol is as resilient against impersonation as HB+.

Theorem 1. *If HB+ protocol is a secure protocol in the DET model, then Tree-HB+ protocol is also secure against impersonation in the DET model.*

Proof (Sketch): Intuitively, the security of Tree-HB+ protocol (against impersonation) relies solely upon the security of the underlying HB+ protocol run at the authentication stage. Note that the tree-traversal stage only involves tag identification and impersonation of tag is not possible during that stage. More formally, we can show that if there exists an attacker $\mathcal{A}$ who can impersonate a tag in the Tree-HB+ protocol, then we can construct another attacker $\mathcal{B}$ who can impersonate a tag in the HB+ protocol. Construction of $\mathcal{B}$ is fairly straight-forward. Whenever $\mathcal{A}$ issues a query (e.g., to launch a session with the uncorrupted tag that it intends to impersonate), $\mathcal{B}$ simply simulates the responses corresponding to the tree-traversal stage as if it were itself an honest tag with randomly chosen secret keys. During the authentication stage, $\mathcal{B}$ simply forwards the queries from $\mathcal{A}$ to the oracles corresponding to the HB+ protocol that it has access to, and replies back to $\mathcal{A}$ with the responses that it receives from the oracles. This simulates a perfect view of the tree-traversal and authentication stages to $\mathcal{A}$. Eventually, $\mathcal{A}$ comes up with a session (i.e., B and $\vec{z}$ values) where it impersonates the tag; $\mathcal{B}$ simply forwards these values to the oracles corresponding to the HB+ attacked session. It is easy to see that the advantage of $\mathcal{B}$ is the same as the advantage of $\mathcal{A}$. $\square$

In the DET model [12, 13, 8], under the conjecture that the Toeplitz-MHB puzzle is hard [8], the HB# protocol is also secure. Since in our Tree-HB# variant protocol we just run the HB# protocol at the authentication stage, then again our protocol is as resilient against impersonation as the underlying HB#.

Theorem 2. *If HB# protocol is a secure protocol in the DET model, then Tree-HB# protocol is also secure against impersonation in the DET model.*

Proof (Sketch): Intuitively, the security of Tree-HB# protocol (against impersonation) relies solely upon the security of the underlying HB# protocol run at the authentication stage. Note that the tree-traversal stage only involves tag identification and impersonation of tag is not possible during that stage. More formally, we can show that if there exists an attacker $\mathcal{A}$ who can impersonate a tag in the Tree-HB+ protocol, then we can construct another attacker $\mathcal{B}$ who can impersonate a tag in the HB+ protocol. Construction of $\mathcal{B}$ is fairly straight-forward, and follows in a very similar manner as discussed in the proof of Theorem 1. $\square$

4.2 Privacy

Our privacy model meant to capture an attacker who can interact with many tags, and who tries to link different authentication sessions. For example, consider an attacker that roams through a library talking to tags and recording the books that these tags are embedded in. Later the attacker detects some tags in the book-bag of a random person on the street, and it tries to recognize these tags (thereby recognizing the books that this person checked out).

A (somewhat simplistic) formalization of this concern has the adversary talking to two tags in trying to determine if they are in fact the same tag: First the adversary chooses two arbitrary distinct tags in the system, and then a fair coin is tossed. If the outcome is head, then the two tags are the ones chosen by the adversary, and if the outcome is tail, then the two tags that the adversary talks to are actually just the first tag that it chose. The adversary now interacts with the "two tags" for a number of authentication sessions, at the end of which it needs to determine the outcome of the coin toss.

We note that this is not the only possible definition for privacy in this setting (and maybe also not the most natural one). For example, this definition does not consider the possibility of the adversary compromising some tags and learning their secrets. Still, we think that this model is a reasonably meaningful one.

The security of our protocol in this model follows easily from the fact that the function $f_{\vec{y}}(B) = B \cdot \vec{y} + \text{noise}$ is a pseudorandom function when B is random, and this fact was proven in [2, section 5] to follow from the hardness of LPN.

Theorem 3. *If the function $f_{\vec{y}}(B) = B \cdot \vec{y} + \text{noise}$, for randomly chosen B, is a randomized PRF, then Tree-HB+ (and Tree-HB#) protocols are privacy-preserving in the model described above (i.e., tag sessions are indistinguishable).*

Proof (Sketch): Intuitively, it is easy to see that if we replace the function $f_{\vec{y}}(.)$ in our protocols with a random function, then the adversary would have zero advantage in determining the outcome of the coin-toss in the experiment described above. More formally, since $f_{\vec{y}}(.)$ is a PRF, telling whether two authentication sessions correspond to the same tag or to two different tags, i.e., distinguishing between $(f_{\vec{y_1}}(B_1),\ f_{\vec{y_1}}(B_2))$ and $(f_{\vec{y_1}}(B_1),\ f_{\vec{y_2}}(B_2))$, where y_1 and y_2 are keys corresponding to two tags, would not be possible, except with a negligible probability. The fact that we can share the same challenge matrix B for all levels of the tree follow from an easy hybrid argument. $\square$

We finally note that our protocol, like all tree-based protocols, is somewhat vulnerable to tag-compromise. If an attacker manages to get the secrets of a specific tag, it can use them to recognize the tree-traversal replies of neighbor tags in the tree (and in particular it can distinguish neighbors of the compromised tags from non-neighbors). Devising mechanisms to mitigate the effect of tag compromise is an interesting open problem. The effects of tag compromise was studied by Avoine et. al [1].

5 Parameters, Optimizations and Performance

Below we analyze the false reject and false accept rates for a fixed set of parameters β, d, r, and then use this analysis to set these parameters (as a function of the total number of tags N).

5.1 False-Accept Rate

We begin by observing that the false-accept rate of our protocol is exactly the same as that of the underlying HB+ that is used in the authentication stage. To see why, observe that the tree-traversal stage of the protocol always results in the reader identifying one leaf of the tree as the most likely to correspond to the right tag, and then the underlying HB+ is run against the keys of the tag in this most likely leaf.

This is in sharp contrast to the situation for the trivial linear search routine that was discussed in the introduction. In that procedure, the reader would try to check the authentication against the keys of all the tags in the system, so the false-accept rate would roughly be multiplied by the number of tags. This does not happen in our algorithm, since the reader only tries to authenticate against the keys of just one tag.

In other words, even giving the adversary "for free" the ability to steer the tree-traversal stage to any tag of its choice, it would still have to be able to authenticate a tag without knowing its keys in order to induce a false accept event.

5.2 False-Reject Rate

We note, however, that the false reject rate of our algorithm will be higher than the standard HB+. This is because the tree-traversal stage may identify a wrong leaf, in which case the authentication stage will almost surely reject. We therefore need to analyze the probability that the tree-traversal stage identifies the wrong leaf.

5.2.1 Taking a false branch, the binary case.

We start by considering a binary tree (i.e., $\beta = 2$) and analyzing the probability that the wrong branch is chosen at one step of the tree-traversal stage. For the binary tree, if the algorithm is currently in a node at level $\ell - 1$ which is on the path to the true tag, then the likelihood of choosing the wrong child for level ℓ is exactly the probability that the non-matching key of that "false child" generates a vector $\vec{z}_f$ that is closer to the response vector $\vec{z}_\ell$ than the matching key of the "true child".

We denote the "true child" of the current node by t (i.e., the child on the path to the true tag that is trying to authenticate itself), and denote the other child of that node by f (for a "false child"). With the keys of these children denoted $\vec{y}_t$ and $\vec{y}_f$, respectively, and the tag sending a challenge matrix B, recall that the reader computes $\vec{z}_f = B \cdot \vec{y}_f$ and $\vec{z}_t = B \cdot \vec{y}_t$, and compares these two vectors against the vector $\vec{z}_\ell$ that was sent by the tag.

The probability of descending into the wrong child f is bounded by the probability that $\vec{z}_f$ is closer to $\vec{z}_\ell$ than $\vec{z}_t$, namely $\Pr[\mathsf{Hwt}(\vec{z}_f \oplus \vec{z}_\ell) < \mathsf{Hwt}(\vec{z}_t \oplus \vec{z}_\ell)]$. We denote by $P_t(i)$ the probability that $\vec{z}_t$, $\vec{z}_\ell$ differ by exactly i positions, and similarly by $P_f(i)$ we denote the probability that $\vec{z}_f$, $\vec{z}_\ell$ differ by exactly i positions. Namely,

$$\begin{aligned} P_t(i) &\overset{\text{def}}{=} \Pr[\mathsf{Hwt}(\vec{z}_t \oplus \vec{z}_\ell) = i] = \binom{r}{i} \varepsilon^i (1-\varepsilon)^{r-i} \quad \text{(where we use } \varepsilon = 1/4\text{)} \\ P_f(i) &\overset{\text{def}}{=} \Pr[\mathsf{Hwt}(\vec{z}_f \oplus \vec{z}_\ell) = i] = \binom{r}{i} / 2^r \end{aligned}$$

Then we bound the probability of taking a false branch by

$$\Pr\left[\mathsf{Hwt}(\vec{z}_f \oplus \vec{z}_\ell) < \mathsf{Hwt}(\vec{z}_t \oplus \vec{z}_\ell)\right] \;=\; \sum_{i=1}^{r} P_t(i) \cdot \left(\sum_{j=0}^{i-1} P_f(j)\right) \tag{1}$$

We can approximate the above expression as follows: Let Δ_{ft} be the difference between the Hamming weight of $\vec{z}_t \oplus \vec{z}_\ell$ and $\vec{z}_f \oplus \vec{z}_\ell$,

$$\Delta_{ft} \overset{\text{def}}{=} \mathsf{Hwt}(\vec{z}_f \oplus \vec{z}_\ell) - \mathsf{Hwt}(\vec{z}_t \oplus \vec{z}_\ell)$$

The random variable Δ_{ft} is the sum of r independent and identically distribution random variables, one for every position in the response vector (and each a random variable over $\{-1, 0, 1\}$). The probability of taking a false branch is then bounded by $\Pr[\Delta_{ft} < 0]$, and by the law of large numbers we can approximate Δ_{ft} by a Normal random variable with the same mean and variance. Recalling that $\mathsf{Hwt}(\vec{z}_t \oplus \vec{z}_\ell)$ has mean $\mu_t = \frac{r}{4}$ and variance $\sigma_t^2 = \frac{3r}{16}$ and that $\mathsf{Hwt}(\vec{z}_f \oplus \vec{z}_\ell)$ has mean $\mu_f = \frac{r}{2}$ and variance $\sigma_f^2 = \frac{r}{4}$, we get that Δ_{ft} has mean $\mu = \mu_f - \mu_t = \frac{r}{4}$ and variance $\sigma^2 = \sigma_t^2 + \sigma_f^2 = \frac{7r}{16}$. Hence we have

$$\Pr[\text{false branch}] \;\le\; \Pr[\Delta_{ft} < 0] \;\approx\; \mathsf{erfc}\left(\frac{r/4}{\sqrt{7r/8}})\right) \;=\; \mathsf{erfc}\left(\sqrt{r/14}\right)$$

For example, if we choose $r = 80$, then we estimate the probability of choosing a false branch as $\Pr[\text{false branch}] \approx 6.97 \cdot 10^{-4}$, while a choice of $r = 144$ gives $\Pr[\text{false branch}] \approx 5.38 \cdot 10^{-6}$.

5.2.2 Taking a false branch, the general case.

For a larger branching factor ($\beta > 2$), we can still use an equation similar to Equation (1) for the probability of choosing a false branch, but estimating it becomes harder. Specifically, we replace the probability that one false tag has less errors than the true tag by the probability that at least one of the false tags has less errors:

$$\Pr[\text{false branch}](\beta) \;=\; \sum_{i=1}^{r} P_t(i) \cdot \left[1 - \left(1 - \sum_{j=0}^{i-1} P_f(j)\right)^{\beta}\right] \tag{2}$$

However, estimating the last expression is harder, since we need to bound the probability that the minimum of several random variables is below zero (and moreover these random variables are highly correlated).

In lieu of an analytical bound, we therefore evaluated the expression from Equation (2) explicitly. Specifically, fixing the target false-branch probability to some constant (either 0.1 or 0.01), we calculate for every branching factor from $\beta = 2$ to $\beta = 10^4$ the smallest response-size r for which the false-branch probability is below that target (For example, with branching factor of $\beta = 1000$ we need $r \approx 80$ to get false-branch probability of 0.1.) As expected, for a constant probability of false branch, the response-size r grew linearly with the log of the branching factor.

5.2.3 The overall false reject rate.

Recall that a valid tag can be falsely rejected either due to the algorithm choosing a false branch at some point during the tree-traversal stages, or due to a false reject in the authentication stage. The probability of failing the authentication stage is equal to the probability that the error vector has more than τ ones, namely the false-reject rate of the authentication stage is

$$FRR(\text{auth}) \overset{\text{def}}{=} \sum_{i=\tau+1}^{r} \varepsilon^i (1-\varepsilon)^{r-i} \binom{r}{i} \tag{3}$$

and the combined false-reject rate of the whole protocol is

$$FRR(\text{Tree-}HB+) \;\approx\; d \cdot \Pr[\text{false branch}] \;+\; FRR(\text{auth}) \tag{4}$$

As mentioned in [13], the false reject rate can be reduced if the tag checks the noise vector ν and only use it if it has at most τ one-bits.

5.3 Computation, Storage and Communication

The computation of the reader during the tree-traversal stage consists of d levels, wherein each level the reader computes β vectors $\vec{z}_c$ (one for every child c of the current node) and compares them to the response-vector $\vec{z}_i$ that the tag sent. Then, the computation during the authentication stage consists of a single execution of parallel-HB+. Recalling that every vector $\vec{z}_c$ during the tree-traversal stage takes one matrix-vector product to compute and the authentication stage takes two more matrix-vector products (one of which with a smaller $k_x \times r$ matrix), we get that the total computation on the reader side is between $d\beta + 1$ and $d\beta + 2$ matrix-vector multiplications.

On the tag side, the tag computes one matrix-vector product for every level of the tree in the tree-traversal stage, and two more in the authentication stage (again, one of which with a smaller matrix), for a total of between $d+1$ and $d+2$ products.

As for communication, the tag sends the challenge matrix B for parallel-HB+ and the response vectors $\vec{z}_i$, it receives the challenge matrix A and then sends $\vec{z}$. Hence the total communication (in both directions) is $r(k_x + k_y + d)$.[3] Finally, the storage requirement on the tag is $k_y(d+1) + k_x$ bits.

An important optimization for our protocol is to choose different values for the response-length r in the two stages, so as to get $d \cdot \Pr[\text{false-branch}] \approx FRR(\text{auth})$. If we denote by r_{tr} the response-length in the tree-traversal stage and by r the response-length in the authentication stage, we get total communication of $r(k_x + k_y) + r_{\mathsf{tr}} d$. The effect on computation is even larger: since we typically have $r_{\mathsf{tr}} < r/2$, the total computation is decreased by about a factor of two.

5.4 Iterating the Protocol

We recall that an easy (and cheap) way of reducing the false-reject rate is to iterate the protocol several times. Specifically, given a protocol Π with FRR of γ, FAR of δ, and complexity C, we can get from it a protocol Π' with smaller FRR as follows: we first run Π once, and if the authentication fails then we run it again. Clearly, Π' has FRR of γ^2 and FAR of 2δ, and we note that its expected complexity is only $C(1+\gamma)$ (since with probability $1-\gamma$ we will only run it once).

Similarly, by repeating the original protocol Π upto s times, we obtain a protocol Π^s with FRR of γ^s, FAR of $s\delta$, and for large s the expected complexity will approach $C/(1-\gamma)$. This means that even protocols with very large false-reject rate (such as the original HB+ that has FRR≈ 0.44) are meaningful, in that we can transform them cheaply to protocols with adequately low FRR.

5.5 Protocol Comparison

In Table 1(b) we compare the trivial exhaustive search with HB+ (ES), our tree-based privacy-preserving HB+ protocol, and the tree-based PRF protocol developed in [17].

Obviously, the calculation on the reader side takes $O(N) \cdot C_{HB+}$ for exhaustive search (where C_{HB+} is the number of calculations for the HB+ authentication protocol), while for the tree-based protocols, this is reduced to logarithmic in N. We also list in that table the storage requirements, the communication, and the corresponding FRR and FAR. (We note that in the exhaustive search case, if the FAR is too high then the FRR is not meaningful as the probability that a valid tag will

[3]We note that just like for HB#, it is likely that here too we can reduce the communication to $r(d+2) + k_x + k_y$ by using Toeplitz matrices for the challenges A, B.

be identified and accepted as a different tag is very high.)

For this table we used the values of $\beta = 1000$ for the branching factor of the tree and $d = 2$ for its depth (for a total population of $N = 10^6$ tags). For HB+ we used $r = 80$ for the response length, $\varepsilon = \frac{1}{4}$ for the error rate and $\tau = \varepsilon r = 20$ for the acceptance threshold, and for the size of the keys we used the "low security" values pf $k_x = 80$ and $ky = 256$ (corresponding to the requirement of 2^{55} memory to break LPN). For the PRF we used key size and output size of 128 bits (e.g., for AES-128).

5.6 Choosing the Parameters

With the analysis from above, we now illustrate how to set the parameters of our scheme for a given tag population and security goals. The results of this sections are summarized in Table 1(c). Specifically, suppose that we want to setup the scheme to achieve the following parameters:

- Tag population of upto $N = 10^6$ tags,
- False-reject rate of 10^{-4} and False-accept rate of 10^{-8},
- Security against attacks that work in space of upto 2^{65} bytes.

For these settings, we investigate the parameters that we get by working with noise rates of $\varepsilon = 0.125$ or $\varepsilon = 0.25$, using tree of depth either $d = 2$ or $d = 3$ (and thus $\beta = 1000$ or $\beta = 100$, respectively).

5.6.1 The key-length k_x, k_y.

Extrapolating from the parameters in [15, Sec 5.2], for this level of security we need $k_y \approx 440$ for noise level $\varepsilon = 0.125$ and $k_y \approx 330$ for noise level $\varepsilon = 0.25$ (and in either case we can get by with $k_x = 80$).

5.6.2 The response-length r and threshold τ.

As mentioned above, we can always get good false-reject rates by repeating the protocol a few times, as long as we start from a protocol that has low enough false-accept rates. Stipulating that repeating the protocol upto four times is reasonable, we thus begin by concentrating on the authentication phase, setting our target for a single run at (say) FAR$\approx 10^{-9}$ and FRR≈ 0.05.

To get these parameters for the case $\varepsilon = 0.25$ we can use response-length $r = 212$ and threshold $\tau = 64(= 212/4 \ + 11)$, which yield FAR$= 1.62E - 9$ and FRR$= 0.034$. Similarly, for the $\varepsilon = 0125$ case we can use $r = 86$ and $\tau = 16(= 86/8 \ + 5.25)$, which yield FAR$= 1.6E - 9$ and FRR$= 0.036$.

To optimize performance, it is beneficial to set a different response-length for the tree-traversal phase, so as to get $d \cdot \Pr[\text{false branch}] \approx FRR(\text{auth})$. Below we denote the response-length for the tree-traversal by r_{tr}. For $\varepsilon = 0.25, \beta = 1000$ we need $r_{\mathsf{tr}} = 102$ (yielding $\Pr[\text{false branch}] = 0.025$), for $\varepsilon = 0.25, \beta = 100$ we need $r_{\mathsf{tr}} = 83$ ($\Pr[\text{false branch}] = 0.0167$), for $\varepsilon = 0.125, \beta = 1000$ we need $r_{\mathsf{tr}} = 40$ ($\Pr[\text{false branch}] = 0.0215$), and for $\varepsilon = 0.25, \beta = 100$ we need $r_{\mathsf{tr}} = 32$ ($\Pr[\text{false branch}] = 0.0146$).

5.6.3 The resulting parameters.

The choices above result in four sets of parameters, depending on the values of the noise rate $\varepsilon \in \{0.125, 0.25\}$ and the depth of the tree $d \in \{2, 3\}$ (corresponding to branching factors $\beta \in \{1000, 100\}$, respectively). In either case, running the protocol once induces total communication $r(k_x + k_y + 1) + r_{\mathsf{tr}} d$, computation of $d k_y r_{\mathsf{tr}} + (k_x + k_y) r$ bit operations on the tag and $\beta d k_y r_{\mathsf{tr}} + (k_x + k_y) r$ bit operations on the reader, and storage requirements of $k_x + (d+1) k_y$ for the tags. Running the protocol once with these parameters results in a false-accept rate of roughly 0.1, which is too high for applications, so we repeat it upto four times. As we explained in Section 5.4, this only increases the expected complexity of the protocol by roughly 10%. The results (when running the protocol upto four times) are summarized in Table 1(c).

It can be seen from Table 1(c) that most of the relevant parameters are improved significantly when moving to a lower-error regime (i.e., from $\varepsilon = 0.25$ to $\varepsilon = 0.125$), and also when moving to deeper trees (i.e., from $d = 2$ to $d = 3$). The only limiting parameter is the memory requirement at the tag, which increases for deeper trees and smaller error rates. Therefore, for any particular application, one should use as deep a tree and as small an error as can be realized subject to the memory available at the tags.

We again note that the communication complexity of the protocol (which is rather large) can probably be significantly reduced by using Toeplitz matrices for the challenges A, B.[4] In this case, the communication will be reduced to $(3 \cdot r + k_x + k_y - 2) + r_{\mathsf{tr}} d$ (In the examples above, this optimization will reduce the communication complexity from the current range of 50000-100000 bits to only about 1000 bits.) The updated values for this version of the protocol (which we refer to as (Tree-HB+^{t}) are shown in the Table 1(d).

Another "big win" will be to be able to derive the keys $\vec{y}$ pseudo-randomly manner from shorter

[4]Another alternative is to replace $A\vec{x} + B\vec{y}$ with $\vec{a} \cdot \vec{x} + \vec{b} \cdot \vec{y}$ over a finite field.

secrets (e.g., using a cheap PRG such as the shrinking generator [7]).

6 Simulation

To estimate the computational overhead incurred at the reader-side when using different methods, we performed a simulation. Our simulation was done using Matlab. The computer used for the simulation is an IBM thinkpad, with Intel CPU, T2400, 1.83 GHz and 1.99 GB RAM.

The test was run for a million tags. We tested two cases for our Tree-HB+ protocol: A two-level tree with 1000 nodes in each level, and a three-level tree with 100 nodes at each level, all with the parameters that were proposed in the original HB+ protocol (i.e., error-rate of $\varepsilon = 0.25$, using $k_x = 80$ and $k_y = 330$). Our results show that for $r = 212$, the exhaustive search average run-time using HB+ was 425 seconds. For the tree-based search (2 levels), we get an average run time of 0.71 seconds, and for 3 levels we get run time of 0.087 seconds. This ratio is compatible with our theoretical estimates, as the exhaustive search will take $O(N)$ (e.g., $\approx 10^6$) while the tree based search only takes $O(d \log N)$ (e.g., $\approx 2 \cdot 10^3$ or $\approx 3 \cdot 10^2$, respectively). The AES tree-based search was performed with a 16-byte random key and uses Matlab code [3] to perform the actual AES calculations. for the tree-based search (2 levels) using AES was around 36 seconds and for 3 levels around 7 seconds, which confirms the fact that AES requires significantly more calculations than standard HB+ protocol. We can further estimate that since the Meet-in-the-Middle PRF scheme takes about $\frac{2}{\sqrt{\beta}} = 0.063$ of the 2-level tree-based PRF scheme, it would take about 2.2 seconds to run it on our computer. Therefore, our simulation shows that our privacy-preserving method is faster then the Tree-based PRF and the Meet-in-the Middle strategy. For $d = 3$, our method is faster by a factor of ≈ 80 relative to the Tree-based PRF and ≈ 16 comparing to the MITM method (for $d = 2$, we get the factors of roughly 10 and 3, respectively).

The results of our simulation are shown in Table 1(e). The mean and standard deviation of the run-time are provided for the different protocols.

7 Conclusions

In this paper, we developed tree-based HB-type protocols for privacy-preserving authentication. This is useful for two reasons. First, very low-cost tags may be incapable of computing standard PRFs and HB-type protocols are currently the only viable solution for such tags. Second, since the underlying computations in HB protocols are very efficient, it automatically reduces reader load compared to PRF-based protocols.

We proposed some significant improvements over the naive use of HB+ in a tree-based scheme. These improvements reduce the computation and communication by a factor of $2\times$ to $4\times$. In fact our tree protocol is almost as efficient as the underlying HB+. The error rates in our protocols are nearly as low as that underlying HB+ protocol. This makes our scheme suitable for a system with large number of tags (Unlike exhaustive search, which would produce unacceptable security for practical systems). We presented analytical and simulation results comparing our method with prior proposals in terms of computation, communication and memory overheads.

References

[1] Gildas Avoine, Xavier Carpent and Benjamin Martin. Strong Authentication and Strong Integrity (SASI) is not that Strong. RFIDSec'10, June 2010

[2] B. Applebaum, D. Cash, C. Peikert and Amit Sahai Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems, Crypto 2009

[3] j. Buchholz, AES Matlab Toolbox, 2001.
http://www.mathworks.com/matlabcentral/fileexchange/1190-aes-toolbox

[4] M. Burmester, B. Medeiros and R. Motta. Robust anonymous RFID authentication with constant key-lookup. ACM symposium on Information, Computer and Communications Security (ASIACCS '08). pp. 283-291. 2008.

[5] A. Blum, A. Kalai and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM 50(4): 506-519, 2003.

[6] J.H. Cheon, J. Hong and G. Tsudik. Reducing RFID Reader Load with the meet-in-the-Middle Strategy, IACR ePrint report 2009/271, 2009.

[7] D. Coppersmith, H. Krawczyk and Y. Mansour. The Shrinking Generator. CRYPTO '93, LNCS vol. 773, pages 22-39. Springer, 1993.

[8] H. Gilbert, M.J.B. Robshaw, and Y. Seurin. HB#: Increasing the Security and Efficiency of HB+, Eurocrypt '08. LNCS vol. 4965, pages 361-378. Springer, 2008.

[9] H. Gilbert, M.J.B. Robshaw, and H. Sibert. Active attack against HB+: a provably secure lightweight authentication protocol. IEE Electronic Letters 41(21), pages 1169-1170, 2005.

[10] T. Halevi, N. Saxena, S. Halevi. Using HB Family of Protocols for Privacy-Preserving Authentication of RFID Tags in a Population. Workshop on RFID Security (RFIDSec) , July 2009

[11] N.J. Hopper and M. Bulm. Secure human identification protocols. Asiacrypt '01, LNCS vol. 2248, pages 52-66. Springer, 2001.

[12] A. Juels and S. Weis. Authenticating Pervasive Devices with Human Protocols CRYPTO '05, LNCS vol. 3621, pages 293-308. Springer, 2005.

[13] J. Katz and J.S. Shin. Parallel and Concurrent Security of the HB and HB+ Protocols. EUROCRYPT '06, LNCS vol. 4004, pages 73-86. Springer, 2006.

[14] J. Katz, A. Smith. Analyzing the HB and HB+ Protocols in the "Large Error" case. IACR ePrint report 2006/326, 2006.

[15] E. Levieil and P.A. Fouque. An Improved LPN algorithm. SCN '06, LNCS vol. 4119, pages 348-359. Springer, 2006.

[16] D. Molnar, A. Soppera and D. Wagner. A Scalable Delegetable Pseudonym Protocol Enable Ownership Transfer of RFID Tags. The 2nd ACM conference on Wireless network security, 2009.

[17] D. Molnar and D. Wagner. Privacy and security in library RFID: issues, practices ACM-CCS '04, pages 210-219, ACM 2004.

[18] Ben Morris, Phillip Rogaway, Till Stegers. How to Encipher Messages on a Small Domain: Deterministic Encryption and the Thorp Shuffle. CRYPTO '09, LNCS vol. 5677, pages 286-302. Springer, 2009.

[19] K. Quafi, R. Overbock. S. Vaudenay. On the Security of HB# against a Man-in-the-Middle Attack. Asiacrypt '08, LNCS vol. 5350, pages 108-124. Springer, 2008.

(a) Notation table

d	depth of the tree
β	branching factor of the tree
k_x, k_y	length of the secrets in the system ($k_x = 80, k_y \in [224, 512]$)
r	size of the "response" messages (usually $r \in [80, 128]$)
ε	noise level ($\varepsilon = 0.25$)
τ	acceptance threshold (usually $\tau \in [20, 40]$)

(b) Protocol comparison for a population of $N = 10^6$ tags

Method	**Reader Computation**	**Communication**	**Tag Memory**	**FAR**	**FRR**
ES HB+	$10^6 \cdot C_{HB+}$	26960	336	0.98	0.44
Tree HB+	$2000 \cdot C_{HB+}$	27120	848	$4 \cdot 10^{-6}$	0.6
Tree HB#	$2000 \cdot C_{HB+}$	734	848	$4 \cdot 10^{-6}$	0.6
Tree PRF	$2000 \cdot C_{PRF}$	1024	256	0	0

(c) Concrete parameters, running the protocol upto four times to reduce the false-accept rates

ε	d	β	k_x	k_y	r	r_{tr}	C_{rdr}	C_{tag}	comm	mem	FRR	FAR
0.25	2	1000	80	330	212	102	$7.49E+7$	$1.71E+5$	97016	740	$6.0E-5$	$6.5E-9$
0.25	3	100	80	330	212	83	$9.23E+6$	$1.88E+5$	97066	1400	$6.0E-5$	$6.5E-9$
0.125	2	1000	80	440	86	40	$3.92E+7$	$8.88E+4$	49864	1400	$3.9E-5$	$6.4E-9$
0.125	3	100	80	400	86	32	$4.74E+6$	$9.66E+4$	49882	1840	$4.1E-5$	$6.4E-9$

Legend: ε- error-rate, d - depth, β -branching factor, k_x, k_y - key lengths, r - response length in auth. stage, r_{tr} - response length in tree stage, C_{rdr} - expected reader computation, C_{tag} - expected tag computation, comm - expected total communication, mem - tag memory requirements.

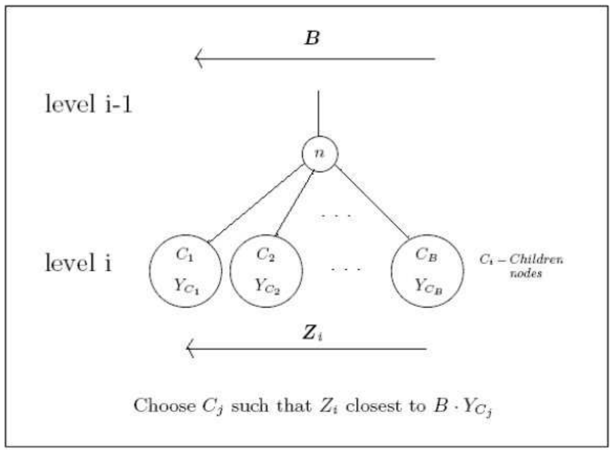
(d) Comparison of communication costs for Tree HB and Tree-HB+^{t} protocols

ε	d	β	k_x	k_y	r	r_{tr}	comm (Tree HB+)	comm (Tree-HB+^{t})
0.25	2	1000	80	330	212	102	97016	1248
0.25	3	100	80	330	212	83	97066	1111
0.125	2	1000	80	440	86	40	49864	856
0.125	3	100	80	400	86	32	49882	832

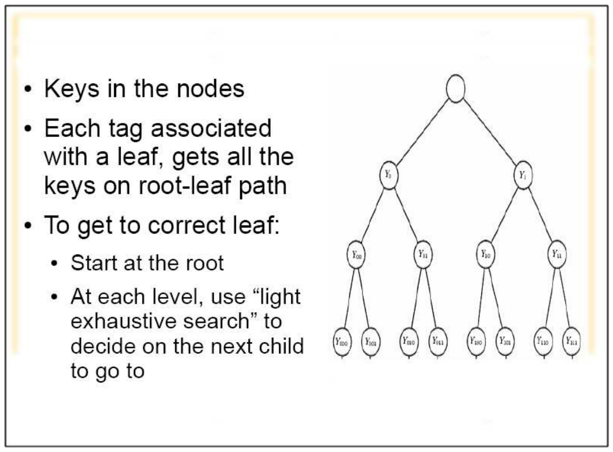
(e) Simulation Results, $r = 212$

Method	**Comp. Time (seconds) mean±stdev**
ES (HB+)	425.638 ± 6.452
Tree HB+ ($d = 2$)	0.71 ± 0.092
Tree HB+ ($d = 3$)	0.088 ± 0.0.009
Tree PRF(AES,$d = 2$)	36.137 ± 0.623
Tree PRF(AES,$d = 3$)	7.277 ± 0.209

Table 1: Results

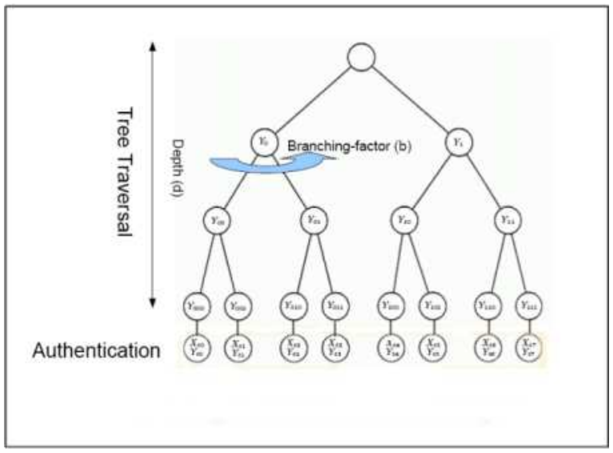


(a) Tree-Traversal Stage (single level)



- Keys in the nodes
- Each tag associated with a leaf, gets all the keys on root-leaf path
- To get to correct leaf:
  - Start at the root
  - At each level, use “light exhaustive search” to decide on the next child to go to

(b) Tree Traversal Stage (multi-level)



(c) TreeHB+ Algorithm

Figure 1: Tree-based HB protocols