

# Privacy through Noise: A Design Space for Private Identification

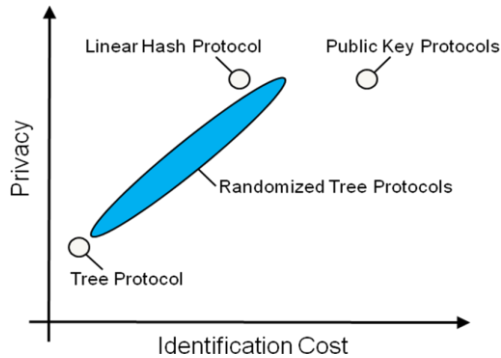
Karsten Nohl  
*University of Virginia*  
nohl@cs.virginia.edu

**Abstract.** To protect privacy in large systems, users must be able to authenticate against a central server without disclosing their identity to the network. Private identification protocols based on public key cryptography cannot be implemented on small devices like RFID tags and are computationally expensive for the backend server. Symmetric key protocols, on the other hand, provide only modest levels of privacy, but can cheaply be executed on servers and implemented on devices. The privacy of these symmetric-key privacy protocols derives from the fact that an attacker only ever knows a small fraction of the keys in a system while the legitimate reader knows all keys. This gap in knowledge can be widened by adding noise to user responses. The noise blurs the borders between groups of users that the attacker would otherwise be able to distinguish. We evaluate the effectiveness and cost of this randomization and find that the information leakage from the tree protocol can be decreased by 99.9% at a 150x increase in the number of hashing operations. Any degree of privacy up to that of public key protocols can be reached while staying well below the cost of public key cryptography. Our technique integrates particularly well with the HB family of protocols, which can be used as an already randomized hash function in our scheme.

## 1 Introduction

The need for an ever-growing number of small devices and tokens to identify or authenticate creates a permanent threat to privacy. To preserve privacy, users must be able to identify themselves to a legitimate server without disclosing their identity to unauthorized readers. Private identification can be achieved through public key cryptography. Public key encryption, however, cannot be implemented on small devices. Hardware implementations of public key ciphers are at least six orders of magnitude less efficient than symmetric primitives such as block ciphers, stream ciphers, and hash functions. Radio-readable credit cards are an example of a large-scale system whose current lack of privacy protection has recently attracted attention [5]. The resource constraints of touch-less credit cards and the increasingly large number of issued cards require a privacy solution that scales gracefully and comes at little extra cost per card. Privacy protocols specifically designed to support the area, power, and scalability constraints of large RFID systems have repeatedly be shown to disclose too much information [1, 12].

The tree protocol was proposed for low-security RFID applications such as retail logistics, where readings are frequent and data has to be available instantly. Credit card transactions, on the other hand, already involve extensive computation needed for processing and fraud detection. Privacy protocols for credit cards can hence be more expensive for the backend server.



**Figure 1.** Privacy-cost design space of existing protocols and new randomized tree protocols

We propose a new randomized protocol that fills the gap between RFID privacy protocols and public key protocols. Through parameterization, our protocol can reach any point on the trade-off curve between scalability and privacy that lies between current RFID protocols and public key protocols as illustrated in Figure 1. Our protocol improves upon a tree-based RFID privacy protocol in which each user is assigned several secrets, many of which are shared with some of the other users. An attacker who steals some of the secrets can begin to distinguish groups of users. We improve the privacy of the tree protocol by flipping a random set of bits in user messages. A legitimate reader that knows the secrets of all users may need to evaluate several tree branches, but always finds the correct match.

In this paper, we analyze the trade-off between added work for the legitimate reader and improved privacy. Using an entropy-based metric, we measure privacy as the amount of information leaked to the attacker for different degrees of randomizations. We evaluate the feasibility of our approach by analyzing the running time of our protocol on dedicated hardware and find that it performs better than public key cryptography even for very large systems up to one billion users. Our protocol can, therefore, provide privacy even for the largest, most ubiquitous computing and sensor network applications. Furthermore, unlike public key cryptosystems, it can be implemented on RFID tags, as it only requires a random number generator and a one-way function.

## 2 Background

The problem of private identification has been extensively studied in the context of RFID tags. Solutions have been proposed that provide provable privacy, but cause extensive computational overhead on the backend server [16]. Other protocols that are more easily implemented than these sacrifice either availability or strong privacy. Those protocols with limited availability maintain and synchronously update some shared state on tag and server side [13]. Too many unauthorized read attempts bring this state out of sync and if the chain of pre-computed states is exceeded, the tag is effectively lost from the database. The other possible trade-off for more scalability sacrifices some privacy by sharing secrets among different users. Only by giving up some privacy, all required properties of identification systems such as scalability and availability can be provided. In this paper, we are extending one such protocol by Molnar and Wagner

that stores keys in a tree of secrets and improve its privacy substantially [9]. Their protocol along with some of its proposed modifications is described in Section 3.2.

The challenge of scalable cryptography has previously been addressed in several contexts. The problem of preventing piracy in multicast networks such as Pay-TV has a very different threat model, but is conceptually close to the question we are considering. One protocol proposed in this context that also uses a tree of secrets allows for counterfeit Pay-TV cards to be linked to the subscriber that leaked access information [14]. The best setup that assigns the smallest number of keys to each user while still being scalable is derived using an entropy-based metric similar to the one we are using.

Our approach uses randomization to enhance privacy. Randomization for the sake of privacy is also found in *privacy-preserving data mining*. In these schemes, data sets that contain sensitive information are perturbed in a way that preserve some statistical values of large subsets of the database but eliminate all private information of individual data items. Whether these techniques can successfully preserve enough information for more than just a few specific queries while providing sufficient privacy is an open research question. Our work is orthogonal to these works as we are providing a way to privately send an identifier that can be used to look up information from databases, but are not considering how to privately release these datasets.

Randomizing responses has previously been used to achieve privacy in RFID systems in the HB family of protocols that were originally developed by Nicholas Hopper and Manual Blum to support authentication by humans without computer assistance [6]. These protocols use only very basic mathematical operations to create a hash function. To achieve one-wayness in this type of hash function and prevent an attacker from recovering the secret key, some of the response bits have to be randomly flipped. The hardness of the HB hash functions relies on the *learning parity with noise* (LPN) problem which has not conclusively been shown to be hard. A first attempt to make the HB protocols secure against active attackers was proven secure in a limited attacker model [8], but later shown to be vulnerable against very practical attacks that are outside of the scope of the proofs [2]. In this man-in-the-middle attack, the attacker flips bits in challenges send by a legitimate reader to a tag and learns key bits from observing whether or not the protocol succeeds [2]. Improved variants of the function have not yet received sufficient scrutiny to assess their security [10]. If a secure hash function can be built based on LPN, this function would provide the perfect base for our randomization technique. Instead of adding noise to the hash output, the HB hash functions already add noise to be secure.

### **3 Private Identification Protocol**

In various applications, including radio-enabled credit cards, protocols are needed to protect the user's privacy, while making it hard to impersonate other users or to counterfeit identification tokens. At the same time, the protocols must be efficiently implementable on the tokens and must not lead to prohibitive computational overhead on the backend server.

### 3.1 Definition

A protocol is considered private if no reasonably powerful attacker is able to distinguish between different users of the system with more than very small probability. The attacker we are protecting against is able to compromise a large (yet limited) number of tokens.

Let  $P(k,r)$  be the output of an identification protocol with secret key  $k$  and random nonce  $r$ . For all polynomial time distinguishers,  $D(\cdot)$ , there exists a bound,  $\varepsilon$ , on the probability that two users with different keys can be distinguished, which is described by the following game:

$$\Pr\left[k \leftarrow U_n; r_1, r_2 \leftarrow U_{f(n)}; x_1 \leftarrow P(k, r_1); x_2 \leftarrow P(k, r_2): D(r_1, x_1, r_2, x_2) = 1\right] - \Pr\left[k_1, k_2 \leftarrow U_n; r_1, r_2 \leftarrow U_{f(n)}; x_1 \leftarrow P(k_1, r_1); x_2 \leftarrow P(k_2, r_2): D(r_1, x_1, r_2, x_2) = 1\right] \leq \varepsilon \quad (1)$$

In the game, the distinguisher is asked to label two different cases. In the first case, one secret key and two random nonces are chosen at random<sup>1</sup> and the protocol is run twice with the same key but different nonces. In the second case, two keys and nonces are randomly chosen and the protocol is run on different keys and nonces. In each case, the distinguisher only gets to see the random nonces and the protocol output but not the secret keys. The distinguisher then has to decide whether a given quadruple was generated from two different secrets (in which case it outputs ‘0’) or using one secret (in which case it outputs ‘1’). The advantage,  $\varepsilon$ , with which the best distinguisher can tell the two cases apart corresponds to the maximum probability with which an attacker can distinguish two different users on average.

For a protocol to be considered private, we need  $\varepsilon$  to be small. Analogous to perfect secrecy [15], we define a protocol to provide *perfect privacy* if and only if the attacker advantage decreases faster than any polynomial in the key length,  $n$ ; that is:  $\varepsilon \leq 1/p(n)$ , over all polynomial functions  $p(n)$ . Even though the value  $\varepsilon$  represents the privacy of a system, there is no fixed threshold that would separate private from not private, but rather different values for different scenarios. Instead of arbitrarily creating such a bound, we provide a way of measuring privacy through our previously introduced metric for *information leakage* [12]. Based on this measure, the designer of a system will have to decide what level of privacy is required (and affordable). Our metric enables protocol designers to estimate the trade-off between scalability and information leakage. The conversion between information leakage and  $\varepsilon$ -privacy is as follows:

*Information leakage* measures how accurately an attacker can distinguish groups of users. It is calculated as the average entropy of the group sizes. This entropy is computed as:

$$L = \sum_i p_i \cdot \log_2 \left( \frac{1}{p_i} \right)$$

where  $p_i$  is the fraction of users in the  $i$ th group. If, for example, an attacker can distinguish 3 groups of 25, 25, and 50 users, the average information leakage is  $L = 2 \cdot \frac{1}{4} \cdot \log_2(4) + \frac{1}{2} \cdot \log_2(2) = 1.5$  bits. The attacker advantage,  $\varepsilon$ , can be expressed in terms of this

---

<sup>1</sup>  $U_n$  is a value randomly chosen from the uniform distribution over all strings of length  $n$ .

information leakage [12]. If on average  $x$  bits of information is learned from users of a system, the attacker advantage of distinguishing two users is the probability that these users do not share the same  $x$  bit identifier; that is:

$$\varepsilon \geq \frac{1}{2^x} \quad (2)$$

If the users are evenly distributed over  $2^x$  groups, each user is in a group of size  $2^{-x} \cdot N$  and  $\varepsilon$  is this bound. If the groups are distributed differently,  $\varepsilon$  is larger. If, for example, there exists groups of two sizes,  $(2^{-x} + \alpha) \cdot N$  and  $(2^{-x} - \alpha) \cdot N$ , then  $\varepsilon = 2^{-x} + 2 \cdot \alpha^2$ . As long as the deviation from the uniform distribution,  $\alpha$ , is small the attacker advantage is very close to the bound described by Equation 2. Given this conversion between information leakage and attacker advantage, we can estimate the  $\varepsilon$ -privacy of an information source by measuring its information leakage.

In addition to privacy, a private identification protocol must also provide correctness, which requires that given the nonces and outputs of the above game, and also the set of all keys in the system, the key or keys corresponding to the nonces and outputs can easily be found. For scalability, the search for the correct key should be fast (sub-linear in  $N$ ) on average.

For a protocol to also provide authentication, an attacker must not be able to spoof a user's response without knowing the user's secret keys, even after seeing many of the user's responses to observed or even chosen challenges. Authentication follows trivially from private identification in the tree protocol (explained in the next section) by adding a server-chosen nonce. In this paper, although we concentrate on improving private identification, all results apply equally to private authentication, because our approach only alters the higher levels of the tree, while authentication only happens at the lowest level of the tree.

### 3.2 Deterministic Tree Protocol

Several RFID privacy protocols have been proposed, all of which sacrifice scalability, availability, or strong privacy. In the basic hash protocol, each user has a unique key. When queried, the user responds with a random nonce and the keyed hash of that random number:

$$H(N, k), N$$

where  $H(\cdot, \cdot)$  is a one-way function and  $N$  is a random nonce. To identify the user, the server hashes the nonce under all keys in the database until it finds a match. The basic hash protocol provides perfect privacy (e.g.,  $\varepsilon < 1/p(n)$  over all polynomial functions,  $p(n)$ ) but does not scale well [16] and the computational overhead is prohibitive for large systems.

In the more scalable tree protocol, several secrets are assigned to each user [9]. The secrets are structured in a tree with the users as the tree leaves. A user  $t_i$  is assigned the secrets  $s_{i,1}, s_{i,2}, \dots, s_{i,d}$  where  $d$  is the depth of the tree (all secrets but the last are shared with some of the other users). When queried, the user responds with

$$H(s_{i,1}, N_1), N_1, H(s_{i,2}, N_2), N_2, \dots, H(s_{i,d}, N_d), N_d$$

The database executes the basic hash protocol for each tree level to find the secret used on each level. Once a leaf is reached, the path from the root to the leaf uniquely identifies the user. This tree-based hash protocol scales well beyond billions of users. The drawback of the protocol, however, is that secrets are shared among several users and extracting the secrets from some users potentially allows tracking others. An attacker can uniquely identify a user with higher probability when more secrets of that user are known. In the standard tree protocol, a tree with a constant branching factor at each level is used. In previous work, we showed that varying the branching factor for the different levels improves privacy [11]. We also showed that a rational attacker would be mostly interested in the high information disclosure from the few users in small groups, while the value of data collected from all other users is relatively low. We consequently proposed to structure the tree in a way that avoids small groups. In this paper, we assume this already optimized version of the tree protocol and further improve its privacy through randomization.

### 3.3 Randomized Tree Protocol

To improve the tree protocol we add noise to the hashes generated by the user to decrease information leakage from the user. Information leakage is caused by the fact that the attacker can learn some of the secrets in the tree. In a tree with two levels where the secrets on the first level are shared and the secrets on the second level are unique to each user, an attacker can steal as many as all of the shared secrets (e.g.,  $\sqrt{N}$  secrets in a two-level tree with constant spreading factor). To improve privacy, we need to increase the size of the groups that the users are hiding in without increasing the workload for the users and with only reasonably increasing the workload of the backend server. A somewhat higher server workload is acceptable in systems where transactions already involve extensive computation. One such application is credit card transactions where book-keeping and fraud-protection are already computationally costly operations. While we argue that somewhat higher workloads are reasonable, we still believe that the computational overhead of public key cryptography would be prohibitive for both device and server. We therefore propose a design space that provides options for these scenarios. Randomization provides cost and design choices anywhere in between the deterministic tree-protocol and public key cryptography.

Our technique is simple: some bits of the user-generated hashes are randomly flipped before being sent to the server. To enable legitimate readers to still uniquely identify (and strictly authenticate if a reader nonce is incorporated) each user, the last level of the tree is not randomized. Where before an attacker could deterministically identify a user as being a member of a single group, the randomization probabilistically places the user in several groups. Randomization amplified the information gap between the legitimate reader that knows all the secrets and the attacker who only knows a small fraction of the secrets.

A simple instantiation of our randomization technique would proceed as follows: The user generates a nonce, then hashes that nonce under its secret key, and finally flips every bit of the result with some probability  $p$ . Any response could hence have been generated by any user with some probability. If  $p$  is chosen too close to  $\frac{1}{2}$  even the legitimate reader has no advantage over trying random keys until the right one is found. If  $p$  is chosen properly, however, the legitimate reader only has to try a few groups to find the right user while an attacker is left

with a large uncertainty as to which group the user is in. The next section provides one possible search strategy for the reader and analyzes its cost overhead as a function of the degree of randomization.

If an attacker can learn several responses that are known to come from the same user, the effect of the randomization is diminished, potentially to the point where the randomized tree protocol does not provide better privacy than its deterministic version. For the randomization to be effective, the user, therefore, needs to respond with the same message when queried multiple times in the same location, but should respond with different messages when queried in different locations. This behavior can be achieved for RFID tags through storing the once-computed and randomized hash in capacitor backed RAM. Once the tag has been out of the reader field for some time, the capacitor is depleted and the stored value is lost. A new hash is then generated on the next query. This will mostly prevent an attacker from learning several responses that are known to be from the same user.

## 4 Analysis

Randomizing responses lowers the information leakage, decreases the chance of a successful identification through a rogue reader, and improves privacy. At the same time, the potential ambiguity of the response can cause extra work for a legitimate verifier. While the legitimate reader will always be able to correctly identify a card, some wrong tree branches might be evaluated before the correct branch. The randomization never leads to false identifications, because the last round that used the unique secret of the card is not randomized.

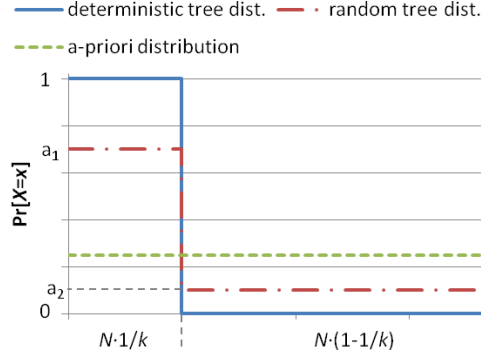
This section presents the trade-off between lower information leakage and increased workload. We also introduce a slight modification of the randomization that greatly extends the design space of possible trade-offs.

### 4.1 Information Leakage

The amount of information learned from the tree protocol depends on the tree setup, the set of secrets known to the attacker, and the position of the identifier in the tree [12]. If no secrets are known to an attacker, any read could have originated from any of  $N$  users and the attacker faces  $\log_2 N$  bits of uncertainty as to which user it might be. As secrets are disclosed to the attacker, the users can be placed in groups of varying sizes. A user in a group of size  $g$  can be identified with uncertainty  $\log_2 p$ . We define the loss in uncertainty  $\log_2 N - \log_2 p = \log_2(N/p)$  as *information leakage*. The average information leakage of the tree protocol for large numbers of broken secrets in large systems with up to billions of users is typically below 7 bits [12].

In the deterministic tree protocol, an attacker can test which of the stolen secrets any user possesses. The randomization blurs this border and the attacker can instead only decide with what probability a known secret was used to generate a given response. When randomizing the response, the probabilities that different keys have generated a given response become more equal, which lowers the information leakage.

We analyze the information leakage from the first level of a two-level tree. As the secrets on



**Figure 2.** Probability distribution of user location as seen by attacker compared for deterministic and random tree protocol. Tree with  $N$  users, spreading factor  $k$ .

the second level are unique, no information is leaked from this level. We will further assume the strongest possible attacker who knows all secrets on the first level but none of the second level.

Figure 2 depicts the probability distribution that reflects where in the tree a given user resides as seen by an attacker. Since the attacker knows all secrets on the first level of the tree, each user can be placed into one of the tree branches when using the deterministic tree protocol. This decreases the number of possible tree positions from  $N$  to  $N/k$  and leaks  $\log_2 k$  bits of information, where  $k$  is the spreading factor on the first tree level. For the randomized tree, on the other hand, the attacker only learns with what probability the user resides in the different branches.

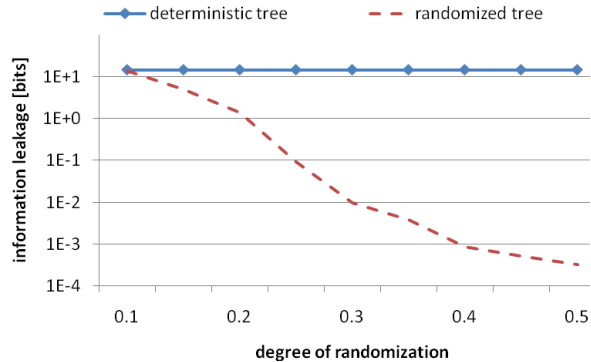
The value of entropy (and information leakage) depends on the probability that the correct secret was used,  $a_1$ , the probability that another secret was used,  $a_2$ , and the number of secrets,  $k$ . The entropy of the distribution is:

$$E = \log\left(\frac{N}{k}\right) + \log(k + r - 1) - \frac{r}{k + r - 1} \log(r). \quad (3)$$

where  $r$  is the ratio of the probabilities that the right or a wrong secret was used,  $r = a_1/a_2$ . The border cases  $r=1$ , and  $r=\infty$  correspond to the linear protocol (no information leaked) and the tree protocol (completely deterministic grouping of users). The derivation of Equation 3 is given in Appendix A.

Note that the probability distribution shown in the Figure 2 depicts the average probabilities over all possible user responses; for most responses the distribution will be very different. On average,  $a_1$  will always be larger than  $a_2$ , but an attacker cannot easily distinguish between the two groups given only a single response.

The different amounts of information leakage achieved by varying the degree of randomization are shown in Figure 3. The amount of disclosed information decreased roughly exponentially



**Figure 3.** Workload required for one identification in deterministic tree with no randomization vs. randomized tree. System with 1 billion users, tree with depth 2. Average values over 100k simulations each.

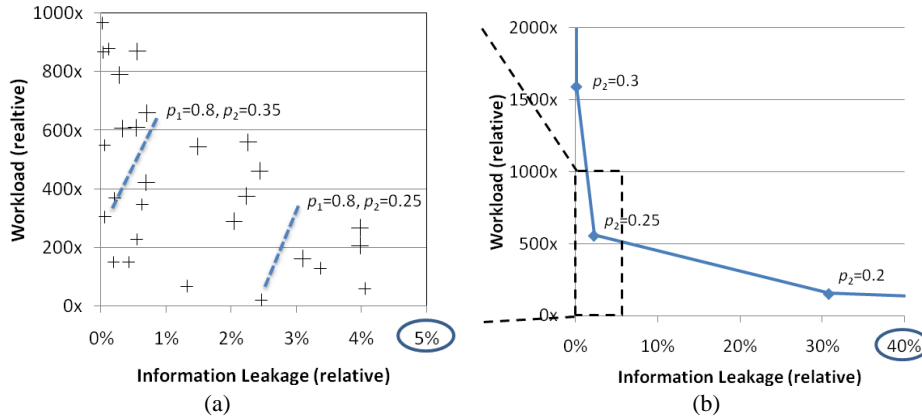
with the degree of randomization (and the attacker advantage,  $\epsilon$ , decreases roughly linearly). For high levels of randomization, the information leakage drops to virtually zero. Next, we analyze the cost associated with adding noise.

#### 4.2 Verification overhead

The verification overhead for the legitimate server grows as more bits of the user’s responses are randomized, as shown in Figure 5. In the unmodified tree protocol with a tree of height  $d$  and spreading factor  $k$ , an average number of  $\frac{1}{2}k \cdot d$  hashing operations are needed for every identification. In the very unlikely worst case, the entire tree is evaluated for a single identification (a sensible implementation would avoid needing to search the entire tree for a bad read by cutting off the search once the probability drops below some threshold). To calculate the expected additional cost of the randomization, we simulated the server workload for a large number of possible parameters. We choose a simple search strategy for the server in which the branches are evaluated based on their initial probability of containing the match. We choose this search strategy merely for its simplicity, while more adaptive strategies would perhaps lead to lower costs (in particular, when deeper trees are used).

The degree to which privacy is improved through randomization varies with the height of the tree. It is largest for shallow trees, mainly because a smaller fraction of the levels is randomized (the last level is never randomized). We have previously shown that duo-level trees have better privacy than deeper trees and are therefore the preferred setup for our randomization [11]. The opportunity to increase privacy by decreasing the height of the tree should hence be fully exhausted before adding randomization, as this usually provides a better trade-off between cost and privacy.

We find that the average cost grows roughly exponentially according to the degree of randomization as depicted in Figure 3. Since information leakage decreases exponentially with randomization



**Figure 4.** Design space for **(a)** selective randomization and **(b)** simple randomization ( $p_1=1$ ) and. System with one billion users, two-level tree. Values are averaged over 100k simulations each.

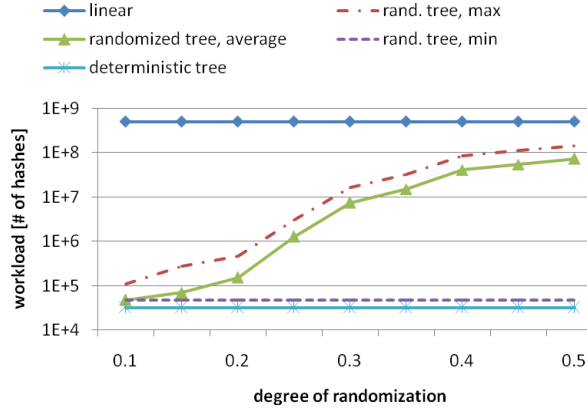
mization, the trade-off between information leakage and cost provided by the simple randomization is roughly linear.

The randomized tree protocol provides design choices that span the whole range of privacy and scalability options between the linear protocol and the deterministic tree protocol. The design space is described by the trade-off between randomization and information leakage as depicted in Figure 3 and the trade-off between randomization and cost in Figure 5. The resulting trade-off curve is shown in Figure 4b. In all the figures, we assume a large system with one billion users and a tree with two levels. The next section provides an enhanced randomization that skews this trade-off towards more privacy.

### 4.3 Selective Randomization

To reach more points in the design space, many of which are superior to what the simple randomization can achieve, we introduce an extension to the randomization scheme. In this selective randomization, we first select a fixed size subset of bits and then flip each bit in this set with a certain probability. On each read, we select a new set of  $p_1 \cdot n$  of the  $n$  bits for randomization and then flip each of these bits with probability  $p_2$ . The simple randomization analyzed previously corresponds to the case where  $p_1=1$ . The selective randomization leads to a distribution with the same expected value as the simple randomization when flipping bits with probability  $p_1 \cdot p_2$ , but the actual distribution is more concentrated around the center. In particular, no value with more than  $p_1 \cdot n$  flipped bits can be assumed. This constraint helps the attacker in that some users are known to not have generated some responses. For well-chosen  $p_1$  and  $p_2$ , however, the probability that at least a few of the wrong secrets could have generated any given response is very high. We calculated the amounts of entropy that the selective randomization preserves and simulated the expected cost for many choices of  $p_1$  and  $p_2$ .

The resulting design space of additional cost versus decreased information leakage is depicted in Figure 4b. Possible points of the design space include one where information leakage is 98%



**Figure 5.** Workload required by different protocols for each identification. System with 1 billion users, tree with depth 2. Values are averaged over 100k simulations each.

lower and cost increases 22 times ( $p_1=0.8, p_2=0.25$ ) and another point where 99.96% less information is disclosed and cost is up 304 times ( $p_1=0.8, p_2=0.35$ ).

## 5 Feasibility

The randomized tree-based hash protocol provides good scalability while leaking very little information. In this section, we quantify the cost and privacy of the protocol for a popular scenario and estimate the resources needed to operate it.

In our scenario, we assume a system with one billion tokens (e.g., RFID tags in credit cards), each of which knows two secrets: one from the first level of a tree that is shared with some of the other tags and a unique secret on the second level. For simplicity, we assume that the spreading factor is the same on both levels at about 31,000. The attacker is assumed to know all the secrets on the first level.

The cost overhead of the privacy protection offered by the randomized tree protocol has so far been measured normalized to the cost of the deterministic tree protocol. We translate this abstract measure into concrete running time on dedicated hardware and compare this cost to that of public key cryptography. To operate the reader-side of the tree protocol, only a hash function is needed. This hash neither needs to provide collision-resistance nor needs to have particularly long keys as long as finding keys is significantly more expensive than other ways to compromise privacy.

In the deterministic key protocol, the reader computes 31,000 hashes on average for each read. The randomization increases this workload by a factor by 304 to 10 million hashing operations and lowers the information leakage by 99.96%.

Hardware implementations of AES achieve about 30 million operations per second per core, about 20 of which fit on a single high-end FPGA chip [3]. As some level of protection is already sacrificed in symmetric protocols to make them scalable, ciphers less strong than AES might well provide sufficient protection. Smaller algorithm such as Tiny Encryption Algorithm (TEA) and its relatives XTEA and XXTEA can be implemented more efficient in the same hardware and perhaps provide a better fit for RFID tags [7]. Efficient ciphers such as A5/1 achieve up to 2 billion operations per second on a single FPGA<sup>2</sup>.

Yet another alternative are HB protocols that can potentially build a one-way function [8]. Since these hash functions only require very basic arithmetic operations, the implementation overhead on an RFID tag is virtually zero. An HB hash would, however, require a significant number of rounds for each hashing operation and hence have a high communication overhead. This overhead is acceptable in applications such as building access control where identification may take up to a second.

In comparison to symmetric one-way functions, public key cryptography such as Elliptic Curve Cryptography or RSA is much more expensive in hardware. One high-end FPGA fits about 10 instances of RSA that each compute about 30 operations per second [4]. RSA is hence about two million times less efficient than AES and seven million times less efficient than A5/1. On the other hand, only a single RSA operation is required for each authentication. One identification using public key cryptography, therefore, is as efficient in hardware as the tree protocol with two million AES hashing operations, which also is the cost for decreasing the information leakage of the tree protocol by 99%. If this level of privacy is sufficient, then all symmetric ciphers that are more efficient than AES are clearly superior to RSA in terms of reader workload. Furthermore, in all application in which larger amounts of information leakage is tolerable, the randomized tree protocol constitutes the only available design option to trade some privacy for less cost.

## 6 Conclusions

The need for private identification can be satisfied within the constraints of small devices and large systems. The proposed protocols are based on symmetric cryptography, so they provide privacy only heuristically, not provably. To assess the effectiveness of these protocols, privacy must be measured in a metric that models the actual privacy threat. Information leakage, the measure of lost entropy in a system, is one such metric that captures how well an attacker can distinguish between different users of a system.

Randomization of responses is an effective design trade-off that increases the workload of the reader as much as it lowers the amount of information leakage. The trade-off provides a large number of design choices for different scenarios, all of which are cheaper than public key cryptography on the reader side and much cheaper on the tag side.

---

<sup>2</sup> Thanks to David Hulton from Pico Computing for this estimate.

## References

1. Avoine, G. and Oechslin, P. RFID Traceability: A Multilayer Problem *Financial Cryptography*, 2005.
2. Gilbert, H., Robshaw, M. and Sibert, H. An Active Attack Against HB<sup>+</sup> - A provably Secure Lightweight Authentication Protocol, 2005.
3. Helion Technology. High Performance AES (Rijndael) cores for Xilinx FPGA [www.heliontech.com/downloads/aes\\_xilinx\\_helioncore.pdf](http://www.heliontech.com/downloads/aes_xilinx_helioncore.pdf), 2007.
4. Helion Technology. RSA and Modular Exponentiation cores [www.heliontech.com/modexp.htm](http://www.heliontech.com/modexp.htm), 2007.
5. Holcomb, D., Burleson, W. and Fu, K. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags *Conference on RFID Security*, 2007.
6. Hopper, N. and Blum, M. A Secure Human-Computer Authentication Scheme *Asiacrypt*, 2001.
7. Israsena, P. Securing ubiquitous and low-cost RFID using tiny encryption algorithm *International Symposium on Wireless Pervasive Computing*, 2006.
8. Juels, A. and Weis, S. Authenticating Pervasive Devices with Human Protocols. *CRYPTO*.
9. Molnar, D. and Wagner, D. Privacy and Security in Library RFID: Issues, Practices, and Architectures *ACM CCS*, 2004.
10. Munilla, J. and Peinado, A. HB-MP: A further step in the HB-family of lightweight authentication protocols *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 2007.
11. Nohl, K. and Evans, D. Hiding in Groups: On the Expressiveness of Privacy Distributions, In Submission.
12. Nohl, K. and Evans, D. Quantifying Information Leakage in Tree-Based Hash Protocols *Conference on Information and Communications Security*, 2006.
13. Ohkubo, M., Suzuki, K. and Kinoshita, S. Cryptographic Approach to "Privacy-Friendly" Tags *RFID Privacy Workshop*, 2003.
14. Poovendran, R. An information-theoretic approach for design and analysis of rooted-tree-based multicast key management schemes *IEEE Transactions on Information Theory*, 2001.
15. Shannon, C.E. Communication Theory of Secret Systems *Bell Systems Technical Journal*, 1949.
16. Weis, S., Sarma, S., Rivest, R. and Engels, D. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems *International Conference on Security in Pervasive Computing*, 2003.

## Appendix A

In this appendix, we derive the amount of information leakage through the randomized tree protocol. Information leakage is defined as the average amount of lost entropy in the distribution of probabilities with which different users could have generated a given response. In the linear hash protocol and in public key protocols, this entropy is  $\log N$  and the information leakage is virtually zero because all users could have generated a response with probability very

close to  $1/N$ , where  $N$  is the number of users in the system. For the deterministic tree protocol with two levels of secrets, the first of which is completely disclosed to an attacker, the entropy is  $\log N - \log k$  and the information leakage is  $\log k$ , where  $k$  is the number of branches of the first tree level.

In the randomized tree protocol, an attacker never learns the exact branch a user resides in but rather a probability distribution over the different branches as was illustrated in Figure 2. On average, the correct branch will have a higher probability than any of the wrong branches (which all have the same probability). The amount of lost entropy (and information leakage) only depends on the difference of these two probabilities.

The entropy of the overall distribution is the weighted sum of the entropies of the tree branch that the user resides in,  $E_1$ , and the entropy of all other branches,  $E_2$ :

$$\begin{aligned}
 E_1 &= a_1 \cdot \log(a_1) & E_2 &= a_2 \cdot \log(a_2) & r &= \frac{a_2}{a_1} \\
 E &= \frac{N}{k} \cdot E_1 + N \cdot \left(1 - \frac{1}{k}\right) \cdot E_2 \\
 &= \frac{k}{k+r-1} \cdot \left( \frac{1}{k} \log(a_2) - \log(a_2) - \frac{r}{k} \log(r \cdot a_2) \right) \\
 &= -\log(a_2) + \frac{r}{k+r-1} \cdot \log(r) \\
 &= -\log\left(\frac{1}{N} \cdot \frac{k}{k-r-1}\right) + \frac{r}{k+r-1} \cdot \log(r) \\
 &= \log(N) - \log(k) + \log(k+r-1) - \frac{1}{k+r-1} \cdot \log(r)
 \end{aligned}$$

The first term,  $\log(N)$ , is the entropy of the prior distribution; the first two terms describe the entropy of the deterministic tree protocol; and the remaining two terms describe the entropy gain through randomization.