Randomized Pseudo-Random Function Tree Walking Algorithm for Secure Radio-Frequency Identification*

Leonid Bolotnyy
University of Virginia
Department of Computer Science
Charlottesville, VA 22904
lb9xk@cs.virginia.edu

Gabriel Robins
University of Virginia
Department of Computer Science
Charlottesville, VA 22904
robins@cs.virginia.edu

Abstract

Privacy and security are two main concerns in Radio Frequency Identification (RFID) systems. We first extend the analysis of the Randomized Tree Walking Algorithm for RFID tag collision avoidance, which is secure against passive adversaries. Then, we devise a new Randomized Pseudo-Random Function (PRF) Tree Walking Algorithm, which is secure against active eavesdroppers and allows for the efficient interrogation of many tags. Our algorithm accommodates the addition and removal of tags from the system, and dynamically adapts to security and privacy policy changes.

1. Introduction

In Radio Frequency Identification (RFID) Systems, a reader interrogates tags located within the reader's interrogation range [4]. In security-conscious systems, we would like to secure the communication between a reader and the tags, preventing an eavesdropper from learning secret information (e.g., the tags' ID numbers). The difficulty here stems not only from the need to provide efficient and secure communication, but also from avoiding collisions (i.e., simultaneous interfering transmissions) between the tags.

We extend the analysis of the Randomized Tree Walking Algorithm introduced in [3] and elaborated upon in [12]. This algorithm is secure against a passive eavesdropper. Next, we use the Randomized Tree Walking Algorithm and an idea from [9] to create an efficient algorithm called Randomized Pseudo-Random Function (PRF) Tree Walking Algorithm that is also secure against an active adversary. Our algorithm builds upon several security papers and adds an important technique that enables dynamic tradeoffs between security, privacy, and singulation time.

There are two types of eavesdroppers: passive and active. A passive eavesdropper can only hear the signal sent from a reader to a tag. The communication channel from the reader to the tag is called the forward channel. An active eavesdropper can hear signals sent from the reader to the tags, and also from the tags to the reader. The communication channel from the tag to the reader is called the backward channel [11].

1.1. Brief Background on RFID Security

We briefly summarize some of the RFID-related security work. Juels proposes a "minimalist" approach to authentication using a one-time-pad scheme [7]. Molnar and Wagner use a tree structure with pseudo-random function to perform authentication [9], and our algorithm builds heavily on their scheme. Weis proposes an authentication algorithm using an enhanced version of Hopper and Blum's human authentication protocol [13]. Juels focuses on authentication problem for Euro banknotes [8].

Weis at. al. propose several access control schemes to enhance security and privacy using hash functions [11] [10] [12]. Garfinkel suggests an RFID Bill of Rights to provide privacy [5]. Bolotnyy and Robins propose a low-overhead approach to RFID data confidentiality by attaching more than one tag to an object and having tags transmit chaff to confuse eavesdroppers [2].

The rest of the paper is organized as follows. Section 2 describes the Randomized Tree Walking Algorithm, and Section 3 develops its analysis. Section 4 introduces and analyzes the Randomized PRF Tree Walking Algorithm. Section 5 concludes with future research directions.

* This research was supported by a Packard Foundation Fellowship, by NSF Young Investigator Award MIP-9457412, and by NSF grant CCF-0429737. For additional related papers see http://www.cs.virginia.edu/robins

2. The Randomized Tree Walking Algorithm

In tree-walking algorithms for RFID tag singulation, tags are placed at the leaves of a tree, and a reader performs a tree-walk beginning at the root in order to identify these tags [6]. The randomized tree walking technique was proposed in [3]. Weis developed this concept further, gave an algorithm, and provided some analysis. In this paper we address three questions that are raised and left open in [12]. Before looking at these issues, we first describe the details of the algorithm of [12].

The algorithm consists of the following steps [12]. Each tag generates a random number, which serves as its temporary ID for the duration of the interrogation algorithm. Then, the reader performs a tree-walk, singulating these random numbers. Once a reader reaches a tree leaf, it queries the tag that generated that random number, and this tag sends its real-ID back to the reader. Since the tag's response is transmitted over the backward channel, a passive eavesdropper does not hear the response, making transmission of the real-ID secure. The pseudo-code for this algorithm is presented in Figure 1 [12].

Note that the algorithm's pseudo-code omits certain details, e.g., it does not include a termination condition when a tree traversal reaches a leaf. It also omits the final phase that reads the real tag IDs. Also, note that if no collisions have occurred for a given threshold number of bits, the algorithm assumes that there is a single tag on that branch, and then selects it. The purpose of this threshold is to enable the selection of a tag before all the bits of its temporary ID are traversed, thus saving time as well as bandwidth. If temporary IDs generated by the tags have high entropy, such a threshold is quite beneficial.

There were three questions left unanswered in [12]:

1. How to handle collisions of the tags' real-IDs?
2. What is the optimal length for random numbers?
3. How to select the threshold?

In the next section we address these questions.

```
Traverse(i, count)
    bi := Read random bit i.
    if collision on bi detected:
        Suspend all tags with bi == 1.
        Each suspended tag stores i.
        Traverse(i+1, 0).
        Wake up tags suspended on bit i.
        Traverse(i+1, 0).
    else if no collision on bi detected:
        if(count > threshold)
            Tree-Walk remaining tags.
        else Traverse(i+1, count+1).
```

Figure 1. Randomized Tree Walking [12].

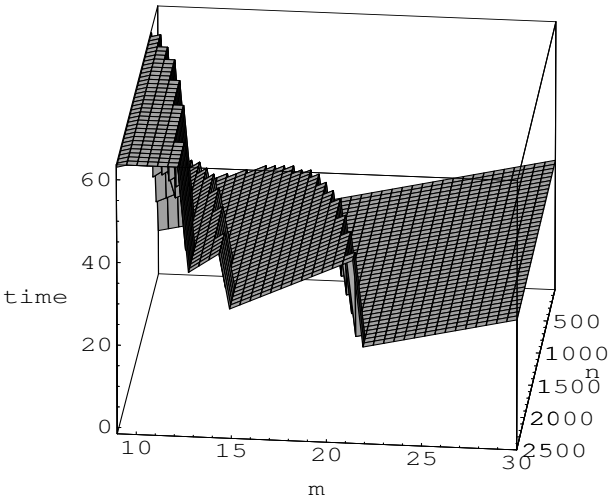


Figure 2. Optimal random number length

3. Analysis of Randomized Tree Walking

We will start by addressing the first question regarding real tag ID collision handling. There are two possible types of collisions on a real ID. The first type is the collision that occurs after there were no collisions for the threshold number of bits. In this case, the algorithm should resume tree-walking from where it left off. The second type of collision is due to different tags coincidentally generating the same random number ID. In this case, the tags that have collided should be temporarily suspended, and the remaining tags singulated. The suspended tags enter a special state, and the Randomized Tree Walking algorithm is repeated only for the tags that had collisions in the previous round.

Next, we will analyze how to choose the optimal length of a random number/ID. Let n be the number of tags in the reader's interrogation zone, and let m be the number of bits in a random number. Weis observed that the number of tags per random number will follow a Poisson distribution with average $\lambda = \frac{n}{2^m}$. Then, the probability that k tags will collide equals $e^{-\lambda} \cdot \frac{\lambda^k}{k!}$. Therefore, the expected number of random numbers with k collisions is $f(n, m, k) = 2^m \cdot e^{-\lambda} \cdot \frac{\lambda^k}{k!}$. The expected total number of colliding tags is $g(n, m) = \sum_{k=2}^{2^m} f(n, m, k) \cdot k$. The algorithm's expected tree-walking time (including potential restarts) is therefore $h(t, m) = t \cdot m \cdot cost_{bit}$ where $cost_{bit}$ is the cost to traverse a single bit, and t is the number of times the traversal algorithm runs.

This formula is a simplification of the real cost, as it does not take into account the threshold effect. The cost to read real IDs without collisions was intentionally left out since it is a constant independent of how many traversals are required. Mathematically, t is the smallest exponent for which $g^t = g(g^{t-1}(m,n),m) <= 1$, where $g^2(m,n) = g(g(m,n),m)$. To determine the optimal number of bits for a random number for each traversal, we can calculate the average number of tags in the reader's interrogation zone over many runs, and use it as n in these calculations. The bit cost $cost_{bit}$ can simply be normalized to one. Figure 2 shows a 3-D graph that depicts pictorially how m may be chosen. From the graph, we see that for $n = 2000$, the optimal m is approximately 20 bits.

Finally, we address the issue of threshold size. First, note that with random numbers as IDs, on average, the tags are expected to be uniformly spread out in the tree. We therefore expect about $t_i = \frac{n}{2^i}$ tags on each branch of a binary tree after i bits of traversal. Therefore, the probability that t_i tags match within the $threshold$ number of bits equals $\frac{1}{2^{threshold \cdot (t_i - 1)}}$. From this equation it appears that a small threshold will suffice; however, the random number generator on-board a tag will not be perfect, resulting in non-uniform distribution of random tag IDs. This suggests that some adaptive threshold scheme may be necessary.

We note that the threshold is important in the Random Tree-Walking algorithm. For example, for $n = 2000$ and $m = 20$, as computed above after about 11 bits, we would expect at most two tags per branch, with many branches containing only a single tag. Forcing the traversal algorithm to proceed for another 9 bits on each branch will incur a relatively high runtime cost while garnering almost no additional benefit. We therefore propose the following adaptive scheme. We start the threshold at 2, and increment it each time we encounter a post-threshold collision for a real-ID. Conversely, we decrement the threshold each time an entire traversal produces no threshold-related real-ID collisions. This adaptive threshold mechanism can accommodate skewed or non-uniform random tag IDs resulting from possible flaws in the random number generator.

4. Randomized PRF Tree Walking Algorithm

The Randomized PRF Tree Walking Algorithm is designed to efficiently solve the reader-tag authentication problem in the presence of many tags. The algorithm consists of the following three steps:

1. Each tag generates a random number, and the reader performs a tree-walk on these random numbers;

2. Once a tag is selected, the reader and the tag engage in a tree-walking private authentication protocol;

3. The reader moves the tag to a different position in a tree.

The purpose of the first step is to allow the reader to communicate one-on-one with each tag while avoiding collisions. It is a modification of the Randomized Tree Walking algorithm described in the previous sections. Once a tag is selected, instead of sending its real-ID to the reader, the reader and a tag engage in a private authentication protocol. This is done in order to prevent an active eavesdropper from learning the ID of a tag.

In the second step, the reader and the tag authenticate each other. Previously suggested private authentication algorithms require time linear in the number of tags to determine the tag's identity. We use the idea presented in [9], which positions the tags at the leaves of a tree and associates a secret with each edge of the tree. Then, the reader performs a tree-walk on this tree, executing an authentication algorithm with each secret along the path from the root to the leaf where a tag is located. Such positioning improves the authentication time from $O(n)$ to an expected $O(\log n)$, where n is the total number of tags in the system.

The algorithm of [9] performs such authentication on both the left and the right sub-trees of a node, which is inefficient in practice since the authentication algorithm is performed twice as many times as necessary. To eliminate this inefficiency, the tag informs the reader whether it is located in the left or in the right sub-tree. To avoid tracking and revealing a tag's location in a tree, which is equivalent to its pseudo-ID, the tag is moved to a different position in the tree in step 3 of this algorithm.

If secrets are hard to steal, an authentication algorithm can be run for a few rounds (less than the depth of a tree where a tag is located), and then a tag can reveal its position in a tree to an authenticated reader. The reader will then move the tag to a new position in a tree after the identification is complete.

If tags are read often, we can relax the algorithm even further and allow the reader to occasionally skip the third step. This can be done either openly or in secrecy (e.g., by sending a special value for one of the secrets to be updated). If tags are active or semi-active, this modification may also have the added advantage of saving power.

Having only a single tree is not adaptable to changes in the number of tags in the system. Creating one large tree with many free leaves is impractical since it increases the number of secrets stored on a tag and the time required for the singulation process to reach a leaf. A single tree also prevents secrets update and fixes the system (i.e., if some secrets are stolen, there is no remedy, and it is also difficult to accommodate new tags). We therefore propose maintaining a forest instead of a single tree, while allowing non-uniform tree depths. With a forest, a new tree can be created as required, and tags can migrate among trees, as well as within a single tree. The number of trees and their depths enables a tradeoff between privacy and the authentication time and

space required on-board each tag to store the secrets. Increasing the depth of a tree will thus strengthen privacy at the expense of efficiency, in a controllable tradeoff.

The steps of the algorithm are shown in Figure 3. Any private authentication scheme would work in step 2 of the algorithm, but the use of pseudo-random functions seems reasonable, even though it is an open problem whether it is possible to implement pseudo-random function ensembles with less hardware than is required for private key encryption [11]. Note that the choice of a scheme in step 2 affects the implementation of the update in step 3 of this algorithm.

4.1. The Algorithm's Notation

In step 1 of the algorithm in Figure 3, a tree of random numbers is generated by the tags and a reader tree-walks the tree. Notice the line "proceed to step 2 with $r = b_1, ..., b_i$" in the algorithm of Figure 3, which does not appear in the algorithm of Figure 1.

In step 2, a reader starts an authentication process with a tag selected in step 1. Here, t is the identifier of a tree where a tag is located. The notation $\in_R$ denotes uniform selection at random from a specified set. In r_1^i and r_2^i, i is not a power of r_1 and r_2. Rather, it is an index referring to the round in which r_1 and r_2 are generated.

The sequence $b = b_1, b_2, ..., b_k$ is the position of a tag in a tree, where $b_j = 0$ refers to a left sub-tree and $b_j = 1$ refers to a right sub-tree. The reader and a tag share k secrets, denoted as $s_{1,b_1}, s_{2,b_2}, ..., s_{k,b_k}$, located on a path from a root to a tag in a tree. Note that $b_i, 1 <= i <= k$ in step 2 are different from $b_j, 1 <= j <= i$ in step 1, since they refer to a tag's position in two different trees.

In step 3, t' is a new tree identifier; b' is a new tag position in t'; t'_{depth} is the depth of t'; and $s_i, 1 <= i <= t'_{depth}$ are new secrets of a tag corresponding to a path from a root to a leaf, where a tag will be located in t'. Secret s_k is the last secret used during an authentication in step 2.

4.2. Properties of the Algorithm

Our algorithm (i) allows tags to be arbitrarily added to and removed from the system; (ii) provides security against active eavesdroppers; (iii) offers security against foreign readers; and (iv) enables dynamic tradeoff between security, privacy, and singulation time.

Tags can be added and removed from the system since the depth and the number of trees is not fixed. The algorithm is secure against active eavesdroppers since a tag ID is never sent in the clear. A tag's pseudo-ID is gradually revealed during the authentication step to authenticated readers only, and it is immediately changed afterwards. Unauthorized readers will not be able to pass through even the first level of a tree where a tag is located. Notice, that even if some tags are compromised and their secrets are stolen, this creates only a temporary privacy weakness for the other tags, since their secrets will be modified on the next read iteration.

The smaller the depth of a tree where a tag is located, the fewer authentication rounds a reader and a tag need to perform, reducing singulation time but enabling the eavesdropper to learn more about the tag. Learning all the secrets of a tag will allow a foreign reader to "steal a tag" by changing its secrets. We do not consider such an attack to be a serious threat, since secrets are not transmitted in the clear, and if an attacker can gain physical access to a tag, then he can simply destroy it physically. If "stealing a tag" is plausible, an enhanced physical protection of the secret ID s_k, at a tree leaf, is required (see the next section for details). An attack consisting of learning a tag's identity and then tracking it (i.e., "hotlisting") would not be possible, since unauthorized readers cannot learn the tag's identity in the first place. Thus, the system is secure.

To implement the algorithm, a tag needs to support pseudo-random functions, have a random number generator on-board, allow reads, writes, and rewrites, and have enough memory to accommodate the three steps of the algorithm.

4.3. Space and Time Complexity

The run time to authenticate a single tag, as well as the space requirements on a tag to store the secrets, depend on the depth of a tree where a tag is located, and are both bounded by $O(\log n)$, where n is the total number of tags in the system. Since our algorithm uses a forest, which is likely to contain more than a single tree, the time and space required to authenticate an object are bounded by $O(depth_{tree})$. Besides storing secrets, a tag needs to maintain its state and store its suspension information in step 1. It also needs to have enough memory to accommodate pseudo-random function computations and state updates.

By varying the number of trees and their depths, we trade-off run time against storage space and privacy. The smaller the depth of a tree, the fewer rounds authentication will require, and the more information an eavesdropper can learn about the tag. If the tags are read often, it is advantageous to make the tree depth a small constant and increase the number of trees, thus keeping the authentication time and space requirements modest without sacrificing privacy.

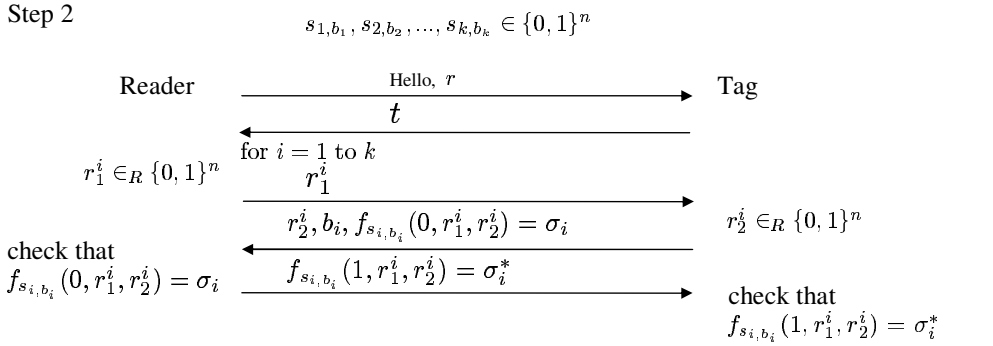
5. Random Number Generation Hardware

The algorithm presented in this paper, as well as many other proposed algorithms (e.g., the Randomized Hash Function -based access control scheme of [11]), use random number generators on-board an RFID tag, yet they

Step 1

Traverse $(i, count)$
  $b_i :=$ Read random bit i
  if collision on b_i detected:
    Suspend all tags with $b_i == 1$
    Each suspended tag stores i.
    Traverse $(i+1, 0)$.
    Wake up tags suspended on bit i.
    Traverse $(i+1), 0$.
  else if no collision on b_i detected:
    if $(count > threshold)$
      Proceed to step 2 with $r = b_1, ..., b_i$
      Tree-Walk remaining tags.
    else Traverse $(i+1, count+1)$.

Step 2



Step 3

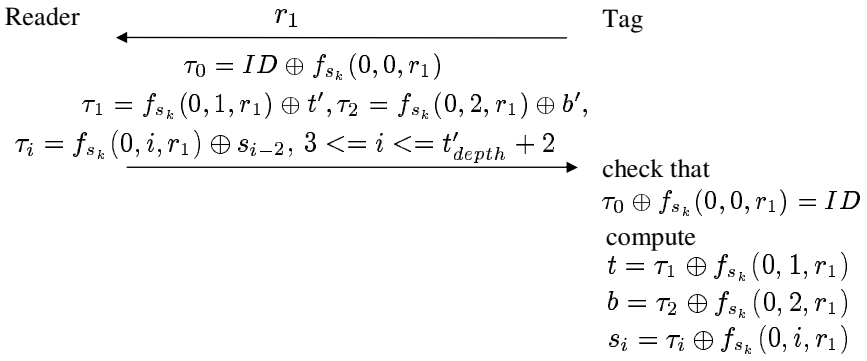


Figure 3. Randomized PRF Tree Walking Algorithm

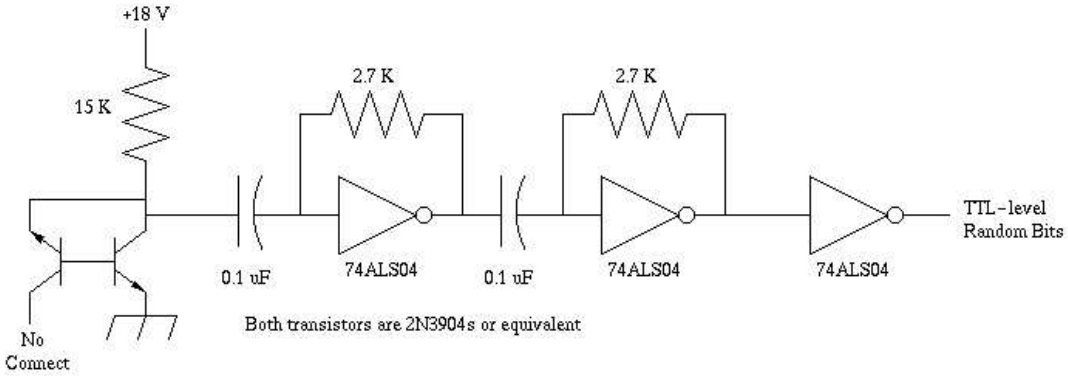


Figure 4. A simple hardware random Bit Generator [14].

do not specify how it may be implemented cheaply. The schematic in Figure 4 depicts a simple circuit for generating random bits [14]. Thus random bit generation does not necessarily require much hardware, only a few capacitors and transistors. The voltage signal is amplified, disturbed, stretched, and sampled, resulting in a stream of random bits. The circuit can be scaled down in size so it does not require a lot of power. An alternate circuit for random number generation and an analysis of its operation can be found in [1].

6. Conclusion

We presented an extended analysis of the Randomized Tree Walking Algorithm in RFID systems, which provides a low-cost, private identification algorithm that is secure against passive eavesdroppers. We then proposed and analyzed a Randomized Pseudo-Random Function (PRF) Tree Walking Algorithm, which efficiently identifies many tags in the presence of active eavesdroppers, and is adaptable to privacy and security requirements. Possible future work includes improving the hardware complexity of the suggested algorithm to further reduce the manufacturing cost of the tags, and to mathematically analyze the security and privacy of such algorithms that change the underlying system configuration with every tree traversal. RFID access control issues can also be considered in different contexts.

References

[1] Anonymous, *Electrical Characteristics and Measurements*, `http://www.protego.se/pdf/Electrical_Characteristics_and_Measurements.pdf`

[2] L. Bolotnyy and G. Robins, *Multi-Tag Radio Frequency Identification Systems*, Proceedings of the Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID), 2005.

[3] Auto-ID Center, *Draft protocol specification for a 900 MHz Class 0 Radio Frequency Identification Tag*, 2003.

[4] K. Finkenzeller, *RFID Handbook*, John Wiley and Sons, 2003.

[5] S. Garfinkel, *An RFID Bill of Rights*, Technology Review, p. 35, October 2002.

[6] A. Juels, R. L. Rivest, and M. Szedlo, *The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy*, Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 103-111, 2003.

[7] A. Juels, *Minimalist Cryptography for Low-Cost RFID Tags*, The Fourth International Conference on Security of Communication Networks (SCN), 2004.

[8] A. Juels, R. Pappu, *Squealing Euros: Privacy Protection in RFID-Enabled Banknotes*, Financial Cryptography, Lecture Notes in Computer Science, vol. 2742, pp. 103-121, 2003

[9] D. Molnar and D. Wagner, *Privacy and Security in Library RFID Issues, Practices, and Architecture*, Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 210-219, 2004.

[10] S. E. Sarma, S. A. Weis, and D. W. Engels, *RFID systems security and privacy implications*, Technical Report MIT-AUTOID-WH-014, AutoID Center, MIT, 2002.

[11] S. A. Weis et al, *Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems*, Security in Pervasive Computing, Lecture Notes in Computer Science, volume 2802, pp. 201-212, 2004.

[12] S. A. Weis, *Security and Privacy in Radio-Frequency Identification Devices*, Masters Thesis, MIT, May 2003.

[13] S. A. Weis, *Security Parallels Between People and Pervasive Devices*, Workshop on Pervasive Computing and Communications Security - PerSec, 2005.

[14] W. Ware, *Hardware Random Bit Generator*, `http://willware.net:8080/hw-rng.html`