

**Implementable Privacy
for
RFID Systems**

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science,

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy in

Computer Engineering

by

Karsten Nohl

January 2009

Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree
Doctor of Philosophy in
Computer Engineering

Karsten Nohl

Approved:

David E. Evans (Advisor)

Gabriel Robins (Chair)

Toby Berger

David G. Smith

Benton Calhoun

Accepted by the School of Engineering and Applied Science:

James H. Aylor (Dean)

August 2008

Abstract

Radio Frequency Identification (RFID) technology bridges the physical and virtual worlds by enabling computers to track the movement of objects. Within a few years, RFID tags will replace barcodes on consumer items to increase the efficiency in logistics processes. The same tags, however, can be used to monitor business processes of competitors and to track individuals by the items they carry or wear. This work seeks to diminish this loss of privacy by adding affordable privacy protection to RFID systems.

Technical privacy measures should be integrated in RFID tags in order to thwart rogue scanning and preserve the privacy of individuals and corporations. To be available for the upcoming deployment of item-level tags, protection measures must not substantially increase the costs of RFID systems. Previously proposed solutions, however, would unacceptably increase the costs of RFID tags, because the solutions use building blocks which were not optimized for privacy applications. Privacy, therefore, has been considered too expensive to be included in low-cost tags. This dissertation instead argues that privacy can be achieved at very low cost within the tight constraints of the smallest RFID tags and the largest installations.

Designing more economical protection systems requires a better understanding of what properties are crucial for privacy. By modeling the incentives of attackers and measuring the extent to which different protection measures rescind these incentives, protection systems can be found that prevent different attacks. Sufficient protection is achieved if the cost of rogue scanning exceeds its expected return for all likely attackers. Perfect protection is neither possible nor necessary to achieve strong privacy.

Protection can be realized through the combination of purposefully designed cryptographic

primitives and optimized private identification protocols. These protocols achieve privacy only probabilistically, but—when parameterized well—disclose very little information. Adding noise to tag responses is one example for a protocol-level measure that provides a tradeoff between privacy and cost. The noise makes most tags indistinguishable to rogue readers while only modestly increasing the workload for the backend system.

Privacy protocols rely on cryptographic functions, but all available functions are too expensive for RFID tags. New functions should not provide expensive properties that are not necessary for privacy, but be an order of magnitude cheaper. Adapting small noise-based hash functions proposed for authentication is one alternative to achieving some of the properties of cryptographic functions without incurring their costs. Another alternative is designing new cryptographic primitives to share resources with functions already present on RFID tags. Such functions can be found through automated tests that measure the cryptographic strength of a large number of possible designs.

To achieve maximal privacy within a given cost budget, all design choices need to be considered concurrently, as similar tradeoffs often exist in different building blocks. This dissertation provides the building blocks needed to achieve strong privacy at low cost as well as a design method for building private systems from these building blocks. Towards this end, contributions are made in modeling the value of information, measuring privacy, optimizing privacy protocols, and designing cryptographic primitives.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Thesis	2
1.3	Background	3
1.4	Contributions	7
2	Measuring Privacy	13
2.1	Private Identification	13
2.2	Privacy Definition	14
2.3	Information Leakage	17
2.4	Threat Model	23
2.5	Attacker Types	30
3	Protocol Layer Privacy	33
3.1	Probabilistic Protocols	33
3.2	Analyzing Probabilistic Privacy	36
3.3	Strengthening Probabilistic Protocols	40
3.4	Improving Privacy by Adding Noise	44
3.5	Feasibility	56
4	Implementing Privacy Protocols	59
4.1	Noise-based Hash Functions	60

<i>Contents</i>	vi
4.2 Private Hash Functions	65
5 Measuring Cryptographic Strength	70
5.1 Related Work	72
5.2 Randomness of Monomial Distributions	74
5.3 CRC-MAC	80
6 Building Private Identification Systems	92
6.1 Picking a Hash Function	93
6.2 Tweaking Protocol Parameters	94
6.3 Choosing Key Sizes	96
6.4 Determining Output and Nonce Length	98
7 Conclusions	100
7.1 Summary of Contributions	100
7.2 Limitations	102
7.3 Open Questions	103
7.4 Summary	104
Bibliography	105

Chapter 1

Introduction

1.1 Motivation

Protection of privacy is often demanded, but rarely implemented. Each time a new consumer-technology is introduced, the protection of private data is seen as a burden to quick deployment. Later on, adding protection becomes even more prohibitive due to the high cost of upgrading legacy installations. For many of today's applications in which privacy is desired, it is no longer possible to add protection to the already deployed infrastructure. This includes e-mail and credit cards—two applications that disclose much more information than necessary to potentially untrusted transport networks and intermediate parties. Radio-readable credit cards provide a recent example of a large-scale system whose current lack of privacy protection has raised some alarm [29].

Radio Frequency Identification (RFID) tags are the next example of an emerging and increasingly widespread technology for which the lack of adequate protection poses a privacy hazard. RFID tags on consumer items enable tracking of the items through the logistics chain. The tags will, among other uses, replace barcodes on retail products and be integrated in clothing—all in an attempt to help computers learn about physical objects in their vicinity. Often cited examples of why computers should gain this ability include washing machines that could prevent clothes from being washed at the wrong temperature, cars that automatically detect their owner approaching, and automated grocery stores where computers track each item to optimally stock, order, and recall products. Available privacy protection measures are seen as prohibitively expensive, and imple-

menters of RFID systems are concerned that making protection mandatory will further delay the wide-scale rollout of RFIDs, which has already been postponed several times due to issues with reading range and manufacturing cost.

Changing RFID standards to include privacy protection becomes increasingly harder as the technology becomes more widely deployed. The number of RFID tags in circulation already exceeds both the number of e-mail addresses and credit cards and will further grow exponentially as tags start replacing bar codes. For the upcoming item-level deployment of RFID tags, privacy measures are needed that are inexpensive to implement and non-disruptive to normal functionality.

Adding privacy protection to RFID tags leads to higher per-tag costs and lower reading ranges (due to the increased power consumption), as well as increased computational cost in the backend system. To support RFID systems with billions of tags, this backend cost must grow sub-linearly with the size of the system. To enable privacy protection through cryptographic protocols on RFID tags, new cryptographic primitives are needed that do not exceed the tags' area and power constraints.

1.2 Thesis

This dissertation argues that privacy is not intrinsically expensive and that sufficient levels of protection can be achieved at very little extra cost for RFID systems. To be effective, such protection should be included in the upcoming deployment of logistics RFID tags to protect consumers and businesses from data theft.

Strong privacy protocols are a necessary building block for RFID privacy, but cannot guarantee privacy since extra information is leaked through other channels such as radio fingerprints and side channels. The magnitude of this extra information appears to be small in comparison with information disclosure on the protocol layer. It is still large enough, however, to dominate the privacy loss of a system once the protocol leaks almost no information. Perfect privacy at the protocol layer, hence, has no significant advantage over nearly perfect privacy, which can be achieved at a significantly lower cost.

Thesis statement: A privacy protection scheme for RFID tags can be realized cheaply when all of its building blocks are designed to provide only those properties that are essential for privacy. A more realistic and comprehensive privacy metric leads to improved protocols and new cryptographic primitives that can provide sufficient privacy by providing protection from all likely threats within the economic and physical constraints of RFIDs.

1.3 Background

The techniques presented in this dissertation achieve good levels of privacy, scale to large systems, and can be implemented on highly resource-constrained devices. Throughout the dissertation, we use RFID applications as a motivating example of constrained devices in large systems with a strong demand for privacy. This section provides background on RFID systems and privacy-protection for RFID systems.

1.3.1 RFID Systems

The RFID tags we consider are small, cheap, and passive radio-readable labels. The tags are powered through induction from a reader field, whose strength decreases rapidly with the distance between tag and reader. Hence, a tag's maximal reading distance depends on its received power consumption and roughly halves when the tag requires 19% more received power¹. Each tag has a unique identification number, which can be read by the reader. The reader uses this ID to obtain information from a backend database. The most severely constrained tags are those used in logistics applications such as Electronic Product Code (EPC) tags. Logistics tags must use very little power in order to achieve sufficient reading ranges and must be very cheap to be affordable for tagging of low-cost items. Advances in chip scaling would make tags with increasingly more transistors affordable for low-cost applications, but, as of yet, not enough power is available on the tag to supply the additional circuitry. Any privacy measure on RFID tags, therefore, has to be low power and low area to achieve acceptable reading ranges.

¹This power estimate assumes parallel sheet-antennas; in particular it is assumed that the available power decreases with $(\frac{1}{d})^4$, where d is the distance between reader and tag antenna.

The basic tags in circulation today provide no privacy protection; any reader can read the unique ID from the tag and look up information in often public databases. One system that causes such direct information disclosure is the planned Object Name Service (ONS) system designed by EPC-global, which uses the Internet Domain Name System (DNS) to locate product information about tagged items [22]. To locate the right ONS server, the system requires that the tag IDs be highly structured (e.g., the same products need to have the same ID prefix), thus leaking significant information even when only small parts of an ID are disclosed. The first steps towards more privacy are to restrict access to the back-end databases and to use unstructured ID numbers.

Even with random IDs and no publicly available metadata, information can be deduced from the tags. In particular, if a tag with a static identifier is read at different times or in different locations, an adversary can trace its movement through the physical world. These traces of RFID movement are similar to Internet traces that record the web pages a user has visited [4]. In the same way that Internet traces have a value and are actively traded [46], we hypothesize that RFID traces also have a value that makes them worth collecting. Applications in which RFID traces can be used include surveillance of individuals and corporate spying. Competitors can, for example, estimate a merchant's turnover by repeatedly browsing through the aisles and reading the tags. Just as Internet traces are increasingly used to derive information about a user's preferences, the movement of individuals can encode information about their shopping behavior, price sensitivity, taste, and many other items of personal information. The information can be used to implement profit-maximizing schemes, including price discrimination and targeted advertising. It should, thus, be desirable for corporations and individuals to control and restrict access to their RFID tags.

1.3.2 RFID Privacy Protection

RFID tags constitute a privacy risk if individuals can be tracked or internal business processes can be monitored through the tags. Hence, privacy requires that different RFID tags cannot be distinguished by rogue readers. Privacy can be achieved through public key cryptography. Users would concatenate their ID with a random number and encrypt it with the public key of the database. Only the legitimate database could decrypt the response to determine the ID. Public key encryption, how-

ever, cannot be implemented on small devices. Hardware implementations of public key ciphers are at least six orders of magnitude more expensive than symmetric primitives such as block ciphers, stream ciphers, and hash functions (as illustrated in Section 3.5). Even optimized silicon implementations currently exceed the power budget by several orders of magnitude [65]. These optimized implementations will likely not become sufficiently power efficient for many years, because of a fixed power overhead caused by the analog front-end and other parts of the chip, and because of diminishing scaling effects [12]. While the active power consumption of each transistor decreases with its size, leakage increases exponentially in smaller technologies thereby offsetting the gain in power efficiency. In order to fit in the power budget of today's RFID chips, all the protocols we consider employ hash functions and use symmetric keys shared between the tag and legitimate readers.

Research into RFID privacy over the past several years has produced a number of possible protection schemes. Early proposals such as breaking or deactivating those tags that leave the domain of the owner—typically at check-out time [31]—have meanwhile been integrated into some RFID standards including EPC Gen2 [21]. These simple solutions sacrifice the possibility of post-sales applications including smart homes and warranty management and are not widely used in practice. Other RFID applications such as payment tokens and passports require the tags to remain active when given to individuals and therefore need more elaborate privacy protection schemes. Furthermore, deactivating tags does not provide any protection from the corporate espionage that would happen before checkout.

1.3.3 RFID Privacy Protocols

Finer-grained privacy schemes employ cryptography to protect the tags from rogue readers. Cryptographic protocols obfuscate the tag ID using random numbers and secret keys. A legitimate server that shares these secret keys with its RFID tags can identify the tags, while any rogue reader without access to the secret keys learns little about the tag's identity. No protocol can satisfy all requirements for an ideal protection scheme at once [64]. Those protocols that provide strong privacy sacrifice either scalability or availability [47] while alternative solutions such as RFID firewalls provide an

added layer of privacy but place additional burdens on the consumer [52].

Hash chain protocols, for example, update secrets on the tag and the server on every read [47]. The protocols provide strong privacy, but risk losing some tags from the database. The legitimate server has to maintain a chain of the next secrets for each tag because any tag can be interrogated by readers of other systems. If unauthorized readers ever read a tag too often, the chain is exceeded and the tag is no longer identifiable even by the legitimate server. Availability is a core requirement for most RFID systems and will therefore be given preference over privacy in many cases.

Basic Hash Protocol. Weis et al. propose a simple privacy protocol that does not sacrifice availability or privacy, but comes at a high server cost. In this basic hash protocol, each user is assigned a single unique key [64]. When queried, the user responds with a random nonce and the keyed hash of that random number:

$$\langle H(s, r), r \rangle$$

where $H(\cdot, \cdot)$ is a one-way function, s is a secret key, and r is a random nonce. To identify the user, the server hashes the nonce under all keys in the database until it finds a match. Assuming a strong one-way function, the basic hash protocol provides perfect privacy (e.g., $\epsilon < 1/p(n)$ over all polynomial functions, $p(n)$) but does not scale well, which results in computationally prohibitive overhead for large systems.

The design of the basic hash protocol acknowledges that RFID tags are highly constrained in cost and power. The protocol consequently offloads most of the work of considering the large number of possible identities from the tags to the server. However, the work required by the server limits the system's scalability. Servers are many orders of magnitude more powerful than RFID tags but can still compute only millions of possibilities in each second. They can hence only handle at most a few reads per second in a system with millions of tags when the basic hash protocol is used. The anticipated scale of many RFID systems, however, is on the order of several billions and the anticipated read rate is at least on the order of tens-of-thousands per minute. Hence, the basic hash scheme cannot be executed cheaply at the scale of some RFID systems. In fact, no protocol

that achieves zero information leakage can be executed cheaply for billions of tags.

Probabilistic Privacy Protocols. Since perfect indistinguishability between different tags (which would guarantee strong privacy) cannot be achieved for systems of large scale, a family of other proposed protocols trade some privacy for highly improved scalability [39]. Many scalable protocols are instantiations of a general tree protocol. For example, the first tree-based protocol proposed by Molnar and Wagner assigns several secrets to each tag [39]. 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, user t_i responds with:

$$\langle H(s_{i,1}, r_1), r_1, H(s_{i,2}, r_2), r_2, \dots, H(s_{i,d}, r_d), r_d \rangle.$$

We further discuss the tree protocol in Section 3.1 and show in Section 3.3.2 that its special case with only two secrets is optimal in many cases.

The scalable protocols do not provide perfect privacy when an attacker can extract group secrets from some of the tags. All tags can be classified based on whether they belong to groups whose secrets are known to the attacker. While the attacker usually learns only little information from identifying which group a tag belongs to (since the number of tags in each group is large), the accumulation of information from several tags that an individual carries might sometimes be enough to uniquely identify that person [43]. However, scalable protocols might still provide sufficient protection when parameterized well and when making certain weakening assumptions about the attacker. Many attackers can, for example, be assumed to act rationally, and will only attack a system when the expected revenue of the attack exceeds its cost, as explained in Section 2.4.

1.4 Contributions

This dissertation presents and analyses building blocks needed for low-cost privacy protection. The focus of the dissertation is on privacy-preserving protocols, cryptographic primitives to instantiate these protocols, and privacy metrics needed to analyze the protocols.

Protocols. Previously proposed protocols for probabilistically private identification are scalable but have been found to provide insufficient privacy protection in several analyses [2] [13] [43]. Furthermore, the only design choices proposed to date provide good privacy at low scalability or good scalability at insufficient privacy. No current scheme offers design choices for points on the tradeoff between these extremes. It is exactly these design points with sufficient privacy and high scalability that would appear most appropriate for protection of consumer privacy. One contribution of this thesis is a family of protocols (described and analyzed in Chapter 3) that provides a continuous tradeoff between privacy and scalability. This family of protocols provides system designers with design choices that achieve enough privacy while keeping the extra cost for the backend server low.

Cryptographic primitives. The privacy protocols we are considering assume the availability of a secure one-way hash on the RFID tag. It is an open research question whether or not such functions could be built small enough to not substantially increase the material cost of RFID tags. Standardized cryptographic functions such as the Secure Hash Algorithm (SHA) family of hashes and the Advanced Encryption Standard (AES) encryption are more than an order of magnitude too expensive in area and power to provide privacy for logistics RFID tags [24] [25]. Other primitives proposed in recent academic works such as the PRESENT block cipher are custom-built for the tight requirements of RFIDs and fit in about a quarter of the resources needed for AES [10]. These primitives would still significantly increase the cost of an RFID tag since an entire tag can be implemented in about half the size of the AES function.

Lastly, some cryptographic functions that are already found in high-cost RFID tags would be small enough for consumer tags. These algorithms, however, were kept secret in an attempt to lock out competitors and avoid attacks. Since peer-review is an essential step in designing cryptographic functions, these secret algorithms tend to be cryptographically weak. We recovered one such proprietary algorithm from the most widely used cryptographic RFID tag, *Mifare Classic*, by reverse-engineering the tag's silicon implementation [45]. We found the algorithm, *Crypto-1*, to be very weak due to statistical flaws that allow secrets keys to be recovered in seconds [17]. Other researchers found another cryptographic algorithm used on RFID tags in car ignition and payment

to be similarly weak due to its small key size after reverse-engineering the chip by black-box testing [11].

This dissertation argues that all current proposals for RFID cryptography are too expensive in part because they were designed as general-purpose primitives for applications other than privacy protection. A cryptographic function designed to provide the properties necessary for privacy protocols, but without providing other functions such as data encryption or digital signing, can be much cheaper than current proposals and perhaps cheap enough to barely increase the cost of an RFID tag. One contribution of the thesis is the formal definition of those properties a function should provide for privacy; we call a function that satisfies these properties a *private hash function*. Another contribution is the design of a private hash function that we tailored for the capabilities and requirements of RFID tags. Our function, *CRC-MAC*, reuses the circuitry of an error checking code that is already present on most RFID tags. Consequently, the implementation overhead of *CRC-MAC* is low enough for privacy to become affordable even for the most cost and power-constrained RFID tags as we show in Section 4.2.

The dissertation, furthermore, generalizes an approach to designing new cryptographic primitives by O’Neil [49], which we adapted to find *CRC-MAC* as illustrated in Chapter 5. Unlike traditional cipher design which focuses on testing small building blocks and tends to over-engineer a function, this design methodology looks at a function as a whole. New constructs found using the approach may have some imperfect building blocks as long as the composition of different blocks prevents their weaknesses from being exploited. We argue that the design of new cryptographic functions should focus on creating automated tests to measure the cryptographic strength of constructs. Given these tools, the strongest primitive can be found from any design space through automated testing. The tests proposed by O’Neil heuristically determine whether the structure of a given construct is distinguishable from a random structure. Any construct that passes the tests is resistant to all known cryptographic attacks that exploit a statistical bias if the right heuristics are chosen. The tests do not evaluate resistance against attacks on a particular implementation, such as side-channel attacks. Our private hash function, *CRC-MAC*, is the result of applying this methodology to a design space consisting of functions built from the *CRC* circuit.

Privacy metrics. The combination of our randomized privacy protocol and our private hash function provides a privacy system that comes at little cost for RFID tag and server while providing a high level of protection. This solution can provide privacy for many applications for which it was so far considered too expensive. Although the level of protection is very high when compared to previous proposals, our system provides privacy only heuristically and does not provide perfect privacy [33].

To evaluate which types of attacks are prevented by heuristic privacy systems like ours, we model the privacy of a system using a new entropy-based metric, which we call *information leakage*. Information leakage measures how much entropy a system lacks when compared to a perfectly private system in which all participants are completely indistinguishable. Unlike previous works on measuring privacy, we do not condense the privacy of a system into a single value, but rather preserve the variance across different users in a privacy distribution. In Section 2.4, we show that privacy distribution can be used to analyze the success probability of different attackers. Our metric is not limited to privacy leaks at the protocol layer, but also can capture information disclosed from any other information source such as the physical layer and other side channels. This thesis contributes a framework to combining multiple information sources into a single privacy distribution that represents the overall information leakage of a system.

We also introduce a new way of modeling attackers through distributions that capture an attacker's ability to learn information from data collected from privacy-protected tags. We argue that an attacker will only attack a system if the expected monetary return of the attack is higher than its cost. Consequently, introducing enough privacy to decrease the expected return below the acquisition cost of RFID traces can deter attacks. RFID privacy attackers have a relatively sensitive cost function, because collecting RFID traces is only rational if it can be done more cheaply than collecting information from all other sources. Most attackers can, hence, be deterred with modest levels of privacy protection.

We model three likely attackers: a corporate spy, a stalker, and a business that builds customer profiles from RFID data. These attackers have distinctly different incentives and, therefore, different success criteria. We capture these differences in the cost functions we use to model the attackers.

These cost functions encode the ability of an attacker to use ambiguous information as well as the value of each extracted datum. A stalker, for instance, is only interested in data from one person’s tags, while a corporate spy wants to extract data from a large number of items sold by a competitor.

Optimizing our protection scheme to be as cheap as possible at a given level of protection leads to different parameterizations depending on which attacker is assumed. Therefore, threat modeling must be an essential part of the design of probabilistic privacy measures. Our framework for measuring information leakage combined with our model for rational attackers allows for privacy threats to be modeled realistically.

Design method. Combining the protocols and cryptographic primitives proposed in this thesis into a privacy protection system opens up many ways to trade off different desirable properties such as privacy, cost, and performance. Qualities such as high privacy and low reader workload often can be exchanged at different levels. Therefore, the most economical solution can be found only by simultaneously considering tradeoffs on different levels.

Measure	Tradeoff	Section
Protocol Layer:		
Vary tree structure	Reader work vs. privacy (performance vs. privacy)	3.3.2
Add/vary noise	Reader work vs. privacy	3.4
Implementation Layer (using HB protocols):		
Vary noise	Performance vs. privacy	4.1
Implementation Layer (using private hash functions):		
Vary key size/number of rounds	Implementation size/reader work vs. privacy	5.3.6

Table 1.1: Overview of design tradeoffs presented in this dissertation.

Table 1.1 summarizes the design space enabled by the contributions of this dissertation. To aid designers of privacy systems, Chapter 6 offers a method for finding the optimal parameterization of the various tradeoffs within a given cost budget. In the design method, our privacy protocols are used either in conjunction with private hash functions or with probabilistic hash functions such as

those proposed by Juels and Weis and those proposed by Soos and Castelluccia [32] [14]. The design space described by the method provides tradeoffs among reader work, privacy, implementation size, and performance.

Summary. This dissertation makes contributions in the analysis, design, and implementation of privacy protection schemes and cryptographic primitives, and suggests low-cost ways to provide adequate privacy in large-scale, resource-constrained systems. To support the thesis, we make the following contributions:

1. Develop a new intuitive privacy metric that measures the effectiveness of privacy protection in distributions of lost entropy (Section 2.3).
2. Formalize a more realistic model for rational attackers that, when combined with our privacy metric, reveals the relative value of an attack (Section 2.4).
3. Improve a privacy protocol to provide a significantly wider and better tradeoff between privacy and cost (Sections 3.3 and 3.4).
4. Introduce *private hash functions*, a new cryptographic primitive that avoids expensive properties not needed for privacy protection (Section 4.2).
5. Present *CRC-MAC*, a private hash function found through automated tests of candidate designs from a large design space. CRC-MAC reuses the CRC circuitry and hence has a very small implementation overhead on RFID tags (Chapter 5).
6. Introduce a design method that aids system designers in making choices about the various tradeoffs on protocol and the implementation layer between privacy, scalability, implementation cost, and performance (Chapter 6).

Chapter 2

Measuring Privacy¹

Comparing and improving privacy protection schemes requires a metric that measures the degree of privacy that different systems provide. The standard notion of privacy in the RFID literature captures only two states of protection: perfect privacy and no privacy [33]. Perfect privacy, however, is not achievable in realistic RFID systems where information leakage occurs on many layers. Furthermore, levels of privacy below perfect privacy can still defeat realistic attackers on RFID systems. Therefore, the design and evaluation of realistic privacy protection in RFID systems requires a more continuous metric that also covers shades of privacy. The next section introduces our notion of privacy, which we show to be related to *information leakage*, an entropy-based metric, in Section 2.3. Section 2.4 explains how an abstract attacker uses leaked information to distinguish between tags, and Section 2.5 explains how different rational attackers would use this ability.

2.1 Private Identification

We define privacy for identification systems such as RFID systems. An identification system allows one entity to identify another. We call the entity that is being identified the *user* (typically an RFID tag, smart card, sensor device, etc.) and the party doing the identification the *reader*. In RFID systems, the reader is usually also exchanging data with a backend server that stores data about users.

¹This chapter is partly based on [44] and [43].

An identification system is considered private if no reasonably powerful attacker is able to distinguish among different users of the system with more than very small probability when compared to other means of automated identification such as face recognition. Attackers differ from legitimate readers in the amount of information they know about the users. In cryptographic protocols, the legitimate reader knows all the secret keys each user possesses whereas an attacker knows at most a subset of these keys.

2.2 Privacy Definition

The goal of a *private identification protocol* is to enable legitimate readers to correctly identify users, while preventing rogue readers and eavesdroppers from learning user identities. We consider attackers who are able to compromise a large (yet limited) number of user devices (i.e., extracting all their key material), and who can actively interact with users and readers. Private identification protocols should provide low levels of information disclosure, and conversely a high level of indistinguishability among users.

Private identification protocols take two inputs: a secret key, s , that is part of a set of secrets, S , and a random nonce, r , to produce an output, h , which usually is a one-way hash that hides the key:

$$h \leftarrow P(s, r).$$

A second requirement for private identification protocols besides indistinguishability is correctness. A legitimate reader should be able to efficiently and correctly identify the sender, s , given knowledge of S , h , and r . For scalability, the search for the correct key should be fast on average and should grow much slower than linearly in the number of users.

Rogue readers do not have access to S . Without knowledge of S , obtaining h and r must leak very little information about s . The legitimate reader secret, S , could be a set of keys, or a single private key (as in a public-key protocol).

The protocols must allow an attacker at most a small advantage in the distinguishing game later defined in Section 2.2.2. Conversely, the protocols must only leak very small amounts of

information.

For a protocol to also provide authentication, an attacker must not be able to spoof a user's response without knowing the 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 dissertation, we concentrate on improving private identification. The results apply equally to private authentication, because our approach only alters the higher levels of the tree, while authentication is limited to the lowest level of the tree.

2.2.1 Related Work

Perfect privacy in RFID systems is captured by the notion of *strong privacy*. For a system to provide strong privacy, an attacker must have no better method of distinguishing different users other than random guessing. Juels and Weiss formalized strong privacy in the form of a two-stage game [33]. In the first stage, an attacker can query each tag a bounded number of times and can acquire key material from a limited number of tags. In the second stage, the attacker picks two uncompromised tags, one of which is then randomly chosen and presented to the attacker. After interacting with this tag a limited number of times, the attacker decides which of the two tags was presented. Strong privacy is given if no attacker can do so correctly more than 50% of the time.

Strong privacy has been shown to guarantee privacy (on the protocol layer) against all possible adversaries [33]. However, perfect privacy is generally not achievable in any RFID systems, because not all side-channels can be eliminated. Achieving strong privacy on the protocol layer is costly because privacy, scalability, and availability cannot be achieved simultaneously for large systems. Therefore, all proposed privacy schemes make limiting assumptions that weaken their applicability to certain systems. This dissertation assumes that degraded levels of protection are sufficient as long as the expected value (monetary or other) of collected data is lower than the cost of collecting it.

Our notion of privacy is closely related to anonymity, which has been studied in the context of mix-nets [20] [56]. Mix-nets try to make the sender (and recipient) of a message anonymous.

The degree of anonymity depends on the size of the *anonymity set*, which is defined as the set of all potential senders of a given message. Perfect anonymity is achieved if the set includes all members capable of sending messages in the system. The metric used by Serjantov and Danezis in the analysis of mix-nets is similar to the metric we propose for the analysis of RFID privacy in that both are based on Shannon’s information theory [57].

Nohara et al. were the first to use entropy in the analysis of RFID privacy protocols [42]. They considered only the case where a single user’s secrets are known to the attacker and concluded that almost no information is leaked if the number of users in the system is large. Our results are consistent with this, but extend to a more likely scenario where multiple user secrets are compromised. Work by Buttyan et al. also considers this case, but uses the average anonymity set size as its measure of privacy instead of entropy [13]. Their measure does not encode the degree to which different tags can be distinguished and, therefore, does not measure protection against rational attackers.

Previous analyses capture the privacy for all users in one value, while we describe the privacy of different users in a distribution. As later discussed in Section 2.4, capturing privacy in distributions rather than single values is crucial to estimating the protection from realistic attackers.

2.2.2 Formal Definition

Our notion of privacy can be described as a game that captures the likelihood that an attacker can distinguish among different users. We will argue in Section 2.4.2 that an attacker will only try to compromise the privacy of a system when this game can be won with probability higher than some threshold. For this *distinguishability game*, we define the user response of one round of a (private) identification as $P(k, r)$ with secret key k and random number (nonce) r . This response mostly consists of the output of an identification protocol, but may include other information such as physical characteristics of the user’s communication device. An attacker who can distinguish tags based on $P(k, r)$ wins the distinguishability game.

For all polynomial time distinguishers, $D(\cdot)$, there exists a bound, ϵ , on the probability that two users with different keys can be distinguished. The attacker’s probability of winning the game is depicted and explained in Figure 2.1.

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

Figure 2.1: Distinguishability game. The distinguisher is given four values, two input nonces, r_i , and two responses, x_i . The attacker is then asked to decide between two different cases for how these values were generated. In the first case, one secret key and two nonces are chosen at random (U_n is a value randomly chosen from the uniform distribution over all strings of length n). 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 of the two cases of the game, 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 two given responses were generated from two different secrets (in which case it outputs **0**) or using one secret (in which case it outputs **1**). The advantage, ϵ , 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 system to be considered private, we need ϵ to be small. We define a protocol to provide *polynomial privacy* if and only if the attacker advantage decreases faster than any polynomial in the key length, n . This is given when for every polynomial function $p(n)$ there exists an m for which for all $n > m : \epsilon \leq p(n)^{-1}$.

2.3 Information Leakage

The advantage ϵ represents the privacy of a system, but there is no fixed threshold value of ϵ that separates private from non-private. Instead the attacker advantage necessary to put a system at risk will vary across systems as well as attackers.

Instead of arbitrarily creating such a bound, we provide a way of measuring privacy through *information leakage*. Based on this measure, the designer of a system will have to decide what level of privacy is required (and affordable). The metric enables protocol designers to trade off between scalability and privacy.

Information leakage measures how accurately an attacker can distinguish groups of users. It

is calculated as the loss of entropy over perfect privacy (i.e., $\epsilon = 0$) averaged over all users of a system. The loss in entropy is computed as the entropy of the sizes of groups of users that are distinguishable:

$$I = \sum_i p_i \cdot \log_2(p_i^{-1})$$

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 $I = 2 \cdot \frac{1}{4} \cdot \log_2(4) + \frac{1}{2} \cdot \log_2(2) = 1.5$ bits.

Information leakage can also be calculated for the case where users can only be distinguished probabilistically; that is: if different users fall into the same set of groups, but with different probabilities. Serjantov and Danezis analyze this question for mix-based anonymity systems [56]. Let the probability that the tags p and s are the same tag be $P(p, s)$, which can be different for different tags. The amount of information disclosed, I_k , for a given tag, p_k , is:

$$I_k = \sum_{i=1}^N \left(P(p_k, g_i) \cdot \log_2 \left(P(p_k, g_i)^{-1} \right) \right).$$

Probabilistic information leakage is not used in this dissertation. It will be required in the analysis of attack scenarios such as radio fingerprinting in which the information collected by the adversary is fuzzy. While the information leaked from the randomized version of the tree-based protocol we analyze is not deterministic, we only compute its average privacy distribution, which is deterministic as will be illustrated in Section 3.2.

The attacker advantage, ϵ , can be expressed in terms of information leakage. If on average x bits of information are 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:

$$\epsilon \geq 1 - \frac{1}{2^x}. \quad (2.1)$$

If N users are evenly distributed over 2^x groups, each user is in a group of size $N \cdot 2^{-x}$, and ϵ is the lower bound described by Equation 2.1. If the groups are distributed differently, ϵ is larger. If,

for example, there exist groups of two sizes, $(2^{-x} + \alpha) \cdot N$ and $(2^{-x} - \alpha) \cdot N$, then $\epsilon = 2^{-x} + 2 \cdot \alpha^2$. As long as the deviation from the uniform distribution, α , is small (e.g., $\alpha < 0.01$), the attacker advantage is very close to the bound. Given this conversion between information leakage and attacker advantage, we can estimate the attacker advantage, ϵ , caused by an information source, by measuring its information leakage.

For the remainder of this dissertation we use information leakage as our measure of privacy since it converts to distinguishability as defined in Section 2.2.2 but is usually more intuitive to calculate. Information leakage can be calculated for various sources of information disclosure, including protocols, side channels, and radio fingerprints as illustrated in the next subsections.

2.3.1 Side Channels

Privacy is potentially compromised at many layers including side channels. Side channels are often caused by physical variance across different tags and can be observed as different timing, power consumption, or radio characteristics. Through observing side channels, for example, the speed at which a tag operates can be learned. When a pseudo-random number generator is present on the tag, the speed at which the pseudo-random numbers are generated discloses the operating speed of the tag.

Random number generators found on some current tags (including cryptographic tags) attempt to generate pseudo-random numbers using a linear feedback shift register (LFSR) with a constant initial condition [45]. Each random value, therefore, only depends on the number of clock cycles elapsed between the time the tag is powered up (and the register starts shifting) and the time the random number is extracted.

We found that one popular type of tag, *Mifare Classic*, initializes its pseudo-random number generator with the same value each time the tag is powered up. In an experiment, we captured the pseudorandom numbers issued after a fixed delay. After the same time, different tags are in different states of their random number generator, but the same tag is usually in the same state. Through the random number, we therefore learn the speed of the tag and the lag between the moment the tag is supplied with power and when it starts operating, which also varies across different tags.

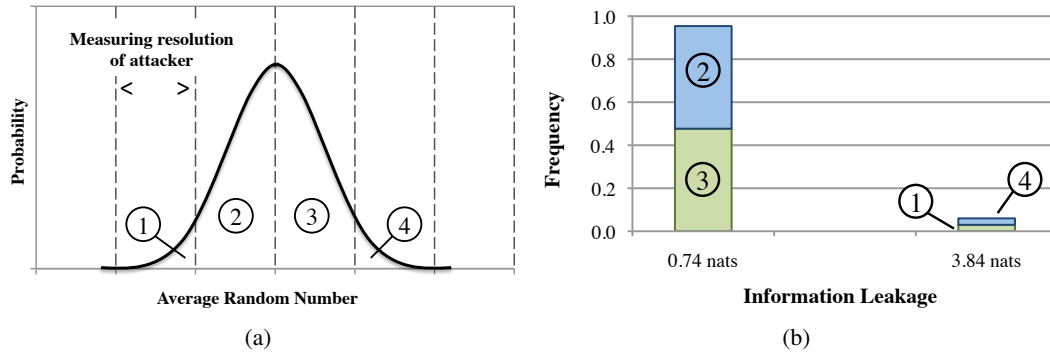


Figure 2.2: Information disclosure of weak random number generator (a) distribution of varying expected value due to process variation; (b) distribution of information leakage.

We assume that the process variation that causes the different operating speeds follow a normal (Gaussian) distribution, which is very typical for VLSI processes. Hence, the divergence of the pseudorandom numbers from the average value also follows a normal distribution. Note that the random numbers are transmitted in the clear and are accessible to rogue readers.

Most tags have too little variance to be distinguishable while a few tags power up or operate noticeably slower or faster than the average so that the expected random value is either slightly lower or higher than the average value in the LFSR sequence. Figure 2.2(a) shows a typical distribution of expected average values for different tags. While our simulation showed that the average of each tag is slightly different from the average of all tags, our experiment was too small to make any claims about the exact shape of the distribution. The graphs should therefore only be seen as an example of a possible side-channel.

The number and size of the groups that are distinguishable due to their random numbers depends on the amount of process variation and the measuring accuracy of the attacker. More variation among the tags and the ability of the attacker to better control the timing of the tag results in more groups being distinguishable. For simplicity in our example, we assume the expected values follow a normal distribution with standard deviation σ and an attacker with a timing resolution of 2σ . All tags can be placed in one of four groups, as shown in Figure 2.2: those that are slightly ahead or behind compared to the average sequence of random numbers produced, and those that are significantly behind or ahead of the average.

An attacker learns more information from tags in smaller groups. A group of z tags corresponds to $\log_2(N/z)$ bits of leaked information where N is the total number of tags that an attacker potentially encounters (and is the total number of tags in the system unless the attacker has extra knowledge to exclude some tags). The distribution of information leakage corresponding to the distribution depicted in Figure 2.2(a) is shown in Figure 2.2(b).

In our example, each of the two groups close to the average value holds 47.7% of the tags. The information leakage from tags in these two large groups is $\ln\left(\frac{1}{0.477}\right) = 0.98$ bits. The leakage from tags in the two smaller groups that diverge more from the average value is much higher at 5.10 bits.

The privacy of a system is directly related to the distribution of information leakage. For the analyzed random number generator, privacy is therefore also directly related to the amount of process variation. Privacy can be significantly increased by lowering the process variation or by excluding a small number of outliers from the system. A better lesson still to be learned from this analysis is that any RNG design that gives control over the generated numbers to the attacker is clearly flawed, especially when used for any cryptographic purpose.

Information leakage through side channels is hard—if not impossible—to entirely avoid. Information about small differences in the physical design, process generation, and manufacturing variance leak through a variety of sources. Therefore, perfect privacy cannot be achieved for RFID systems, even if no information is disclosed on the protocol layer. Fortunately, perfect privacy is not necessary for defeating likely attackers as we will show in Section 2.4.

2.3.2 Protocol Layer

In the same way that we found the distribution of information leakage for the number generator, we can find similar distributions for most other sources. When scalable privacy protocols are used, an attacker can extract some of the shared secrets and then distinguish groups of tags based on which of these secrets they use. To analyze the privacy of these probabilistic privacy protocols, we need to know the distribution of group sizes and the likelihood that a randomly selected tag would fall into a group of a certain size. The distribution of group sizes depends on the fraction of users that share a secret and the set of secrets known to the attacker.

The privacy of one probabilistic privacy protocol, the tree-based hash protocol, has been estimated in several research papers. A first analysis of the tree protocol by Avoine et al. calculated the probability that two readings from the same tag can be linked and concluded that the protocol provides insufficient privacy, since the probability is not negligible [1]. Their notion of privacy is too strong to capture realistic attackers and too expensive to implement. We advocate that the ability of an attacker to build whole traces should instead be considered.

Alternatively we can calculate the privacy of a system by measuring the average anonymity of all tags. One metric that uses this idea expresses privacy as the average number of tags from which each tag cannot be distinguished [13]. A more precise metric measures the entropy of these groups [43]. Both approaches measure privacy as a single value, which is limiting in two ways. First, condensing privacy into a scalar value presumes how the attacker will use the leaked information. Different attackers, however, use the information in disparate ways and hence the privacy of a system depends on the attacker's incentives and capabilities. Secondly, when averaging the information leakage over all tags in a system, information about which parts of the system are mostly responsible for privacy deficits is lost. Understanding the distribution of information leakage is crucial for reducing the amount of information leaked.

In Section 3.2, we show that the information leakage for probabilistic protocols follows an exponential distribution similar to the RNG side channel discussed in the previous section.

2.3.3 Combining Information Sources

Attackers are not limited to any one source of information such as protocols, side channels, or physical characteristics. An attacker might even integrate information from sources outside of the RFID system such as face and voice recognition. While the exact distribution of many of these sources is yet unknown, all of them can be modeled using distributions of information leakage.

Our approach of modeling information leakage as a probability distribution allows for all these different sources of information disclosure to be combined into a single distribution. Different exponential privacy distributions, for instance, can be condensed into a single gamma distribution as shown in Section 3.2.2. The data leaked from protocol and side channels of an individual's

various tags can therefore be expressed in a single distribution that encodes the overall privacy of the person.

2.4 Threat Model

Measuring the level of privacy that tag holders experience presents a hard problem since the perception of privacy varies widely among different individuals and organizations. Furthermore, individuals typically lack good understanding of how much protection is required to match their personal privacy preferences. Information disclosed voluntarily will often compromise privacy when combined with other information or when analyzed using data mining techniques. Because of these difficulties in modeling individual privacy perception, we model privacy as perceived by an attacker and argue that consumer privacy is achieved when no attacker has incentive to compromise privacy. We derive the incentives and capabilities of different attackers in this section and show in Section 3.3 how protocols can be designed to defeat particular types of attackers. Considering attackers' incentives is important in the design of protection schemes since some protection measures improve privacy in the face of certain attackers, but hurt privacy when facing others.

Attackers can be divided into categories according to their capabilities and goals. We consider three likely classes of attacks in Section 2.5—compromise of sensitive business information, tracking of customers for profiling, and surveillance of individuals—but first establish a basic strategy that all attackers have in common.

2.4.1 Extracting Traces

We consider attackers who mine RFID data sets for traces, where a *trace* is a set of readings from the same tag. In order to build traces, an attacker wants to link the different RFID readings collected from a tag or individual. Each reading consists of time, place, the randomized tag identifier, and potentially additional metadata. The attacker's goal is to extract individual traces from a collection of many intermingled traces. The likelihood that the attacker is successful grows with the amount of information leaked by the tags. Readings have higher value to the attacker when they carry

more information. The exact value function that maps information to value (monetary or other) is unknown and will be different for different attackers. We can, however, model the approximate shape of such a function and find an upper bound that corresponds to the strongest attacker that is most skilled in using additional information to distinguish traces.

An attacker wants to extract information from data collected by reading RFID tags. The data the attacker collects is composed of many intermingled traces from different tags. We assume that each individual trace becomes valuable only when it can be separated from all other traces, thus identifying a set of readings from a single tag (or a conjoined group of tags). Whether a trace can be separated depends on the amount of information leaked by tags, the employed protocol, and the secrets known to the attacker.

To separate traces that are indistinguishable given deterministic information (i.e. from the protocol layer), the attacker will further employ data mining techniques, use additional information sources such as the weak random numbers from Section 2.3.1, other side channel information, or contextual information, such as place and time of read, to distinguish traces. To capture the success of the attacker in doing so, we introduce two functions: the *attacker strategy function*, which describes the sophistication of the attack, and the *binning function*, which captures the clustering of traces.

Based on information leaked by the protocol layer, groups of tags can be distinguished with certainty². Indistinguishability at the protocol layer, however, is not sufficient for privacy since an attacker can potentially use information from other sources. The attacker strategy function encodes the probability with which an attacker can distinguish traces that are indistinguishable at the protocol layer. The function captures the side channel information and data mining techniques that are available to the attacker, and varies widely for different attackers. Even though the function is generally unknown—even to the attacker—we can derive an upper bound on it.

First, we define P as the average probability that two traces can be distinguished in a set of readings that contains only these two traces. This average probability is closely related to the success of our attacker who is interested in collecting a large number of traces.

²We add randomization to probabilistic privacy protocols, which blurs the border between the different groups, in Section 3.4.

Given a set of any number of readings known to come from two different tags, the attacker can separate the readings into two groups based on the tag from which they were generated with probability P . The actual probability also depends on the number of readings for each tag in the set. The attacker, however, does not control the number of readings collected from a specific tag in different locations, but only the total number of readings collected from all tags. Hence, our definition of P does not depend on the number of readings, just on the number of different tags which could have generated the readings. This models an attacker trying to extract traces from a data set without having detailed knowledge of the composition of the data.

Given this definition of P for separating two traces, it follows that traces cannot be separated from groups of three intermingled traces with average probability better than P^2 . If the chances of extracting a trace from a set of three traces were higher, an attacker could distinguish two traces with a probability higher than P by intermingling an additional trace. We can generalize to any number of intermingled traces:

Theorem 2.1. (Trace Extraction Theorem) *Let P be the average probability over all traces that two traces can be distinguished. Then a trace cannot be extracted from a set with $g > 1$ traces with an average probability better than $P^{(g-1)}$.*

To prove this theorem we show that when a trace is added to a set of traces, extracting that trace from the set is at least as difficult as distinguishing the trace from every member of the set. We further show that this probability is always maximized when all pairs of traces can be distinguished with the same probability.

A *trace* is a series of readings of the same RFID tag, where each reading is a data item consisting of time, place, a randomized identifier, and potentially further metadata. Traces are denoted t_i . A *set of traces*, $T = \{t_i, t_j\}$, is an intermingled collection of traces from different tags that are indistinguishable on the protocol layer. The event that trace t_i can be extracted from set $\{t_i, t_j\}$ is denoted $\{t_i, t_j\} \rightarrow t_i$. The probability of this event, $Pr[\{t_i, t_j\} \rightarrow t_i]$, is denoted $p_{i,j}$. It follows that $p_{i,i} = 1$ and $p_{i,j} = p_{j,i}$.

Denote by S_k the set of all sets of k traces. Let $n = |S_1|$ be the number of traces in the system. The average probability that any two traces can be distinguished, denoted as P , is:

$$P = \frac{1}{\binom{n}{2}} \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^n p_{i,j}.$$

We assume that P is known and show how the maximum average probability of distinguishing more than two intermingled traces depends on P .

Lemma 2.1. *If a trace t_i cannot be extracted from $\{t_i, t_j\}$ then the trace cannot be extracted from $\{t_i, t_j, t_z\}$ for any z .*

Proof. Suppose there exists an algorithm $A(\cdot)$ that extracts t_i from $\{t_i, t_j, t_z\}$ but no algorithm that extracts t_i from $\{t_i, t_j\}$. Since $A(\cdot)$ must work independently of the number of readings in t_z , executing $A(\{t_i, t_j, \emptyset\})$ extracts t_i from $\{t_i, t_j\}$, which is a contradiction. ■

The following corollary follows by generalizing the lemma to sets with any number of members:

Corollary 2.2. *Trace t_i can be extracted from set T only if it can be distinguished from each member of T , hence:*

$$Pr[T \rightarrow t_i] \leq \prod_{t_j \in T} p_{i,j}.$$

Proof. (Proof of Trace Extraction Theorem) The value $Pr(g)$ is the average probability of extracting any member from any set of g traces:

$$Pr(g) = \frac{1}{k \cdot \binom{n}{k}} \cdot \sum_{G \in \mathcal{S}_k} \sum_{b \in G} Pr[G \rightarrow b].$$

Following the Corollary, this is at most:

$$Pr(g) \leq \frac{1}{k \cdot \binom{n}{k}} \cdot \sum_{a \in \mathcal{S}_1} \sum_{G \in \mathcal{S}_{k-1}/a} \prod_{b \in G} p_{a,b}.$$

Assume towards contradiction that $Pr(g)$ was maximized for a distribution where not all $p_{i,j}$ values are equal. Then, there exists a pair of p values that differ by some non-zero value:

$$\exists p_{a,b}, p_{c,d}, \alpha > 0 : p_{a,b} = p_{c,d} + 2 \cdot \alpha.$$

Assuming a scenario where all other pairs are in equilibrium, shifting weight from $p_{a,b}$ to $p_{c,d}$ decreases those summands of $Pr(g)$ that contain only $p_{a,b}$, but increases those summands that only contain $p_{c,d}$ by the same amount. Those summands that contain both pairs increase by a factor of:

$$\frac{(p_{a,b} - \alpha)(p_{c,d} + \alpha)}{p_{a,b}p_{c,d}} = 1 + \frac{\alpha^2}{p_{a,b}p_{c,d}}.$$

For any distribution of the $p_{i,j}$ values, starting at the largest gap and successively shifting weights from larger to smaller values constantly increases $Pr(g)$ until all $p_{i,j}$ are equal. The probability of extracting traces from sets of any size is maximized when all $p_{i,j}$ are in equilibrium distribution and is at most:

$$Pr(g) \leq \prod_{g-1} p_{i,j} = P^{(g-1)}.$$

■

The *attacker strategy function* is the probability that a trace can be extracted from a set of traces of a given size, g , averaged over all traces in the set. The strategy function is upper-bounded by:

$$S(g) = P^{(g-1)}.$$

2.4.2 Modeling Attack Value

To derive the overall success probability of an attack, we require a second function, in addition to the strategy function, which we call *binning function*. The binning function describes the distribution of traces into groups that can be distinguished using protocol data or information from other sources. The amount of information leakage differs for different users. In many cases the level of privacy for different users follows an exponential distribution: most users experience high levels of protection while a few users stand out by experiencing high levels of information leakage. For example, the information leakage from the RNG side channel in Section 2.3.1 was assumed to follow an exponential distribution, similar to the protocol-layer leakage of probabilistic privacy protocols.

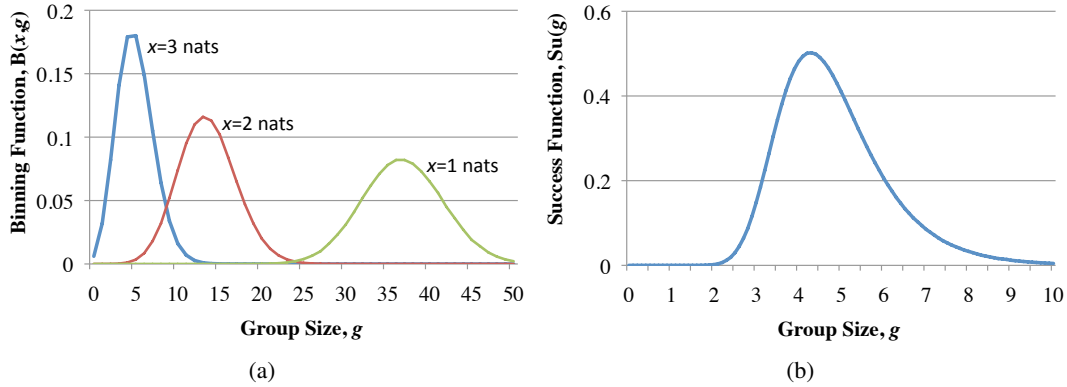


Figure 2.3: (a) Binning Function: probability that a trace with x nats of entropy is intermingled with g other traces; data set with $t=100$ traces. (b) Success Function: probability that a trace with x nats of entropy can be extracted; $t=100$.

To simplify the following calculations, we calculate information in *nats* rather than *bits*. Nats are similar to bits but computed with base e ; 1 nat equals $\frac{1}{\ln(2)}$ bits (approximately 1.33 bits). A group size z therefore corresponds to $\ln\left(\frac{N}{z}\right)$ nats of information.

Let the probability that some information source discloses x or more nats of information be:

$$Pr(X \geq x) \leq \frac{1}{e^x}.$$

Given this exponential privacy distribution of information leakage and a data set with $t + 1$ intermingled traces, the probability that g indistinguishable traces with x nats of entropy are found in the set is:

$$B(x, g) = \binom{t}{g} \cdot \left(1 - \frac{1}{e^x}\right)^{t-g} \cdot \left(\frac{1}{e^x}\right)^g.$$

The binning function is shown in Figure 2.3(a) for traces with different entropies.

Given a strategy function and a binning function, we derive the success probability of a distinguishing attack. The *success function* encodes the probability that a trace with given entropy can be extracted. This is the likelihood that the trace is intermingled with $(g - 1)$ other traces, $B(x, g)$, multiplied by the probability that the attacker can extract a trace from a set of g traces, $S(g)$, and summed over all possible mixes $g = 0, 1, \dots, t$ ($g = 0$ is a single, not intermingled trace):

$$Su(x) = \sum_{g=0}^t (B(x, g) \cdot S(g)).$$

The success function for our example information source is shown in Figure 2.3(b).

To determine whether an attack is successful, the success function needs to be interpreted in light of the attacker's requirements. The value of an attack to the attacker depends on the likelihood that traces can be extracted from the collected data, which is given by the success function. Multiplying the success function by the likelihood that a trace with given entropy occurs, and integrating over all possible entropies (that is, the entropies from that largest to the smallest group) gives us the expected value of the attack:

$$Value = \int_{ent(lrg)}^{ent(sm)} (Su(x) \cdot Pr(X = x)) \cdot dx, \quad (2.2)$$

where $ent(lrg)$ and $ent(sm)$ are the entropies of the largest and smallest groups in the system.

For the exponential distribution of information leakage that many information sources experience, significant amounts of information are only found in some groups. In case of the random number side channel, only those tags that deviate significantly from the average leak notable amounts of information. In both the random number generator and probabilistic privacy protocols, this is due to the fact that a large majority of the tags hide in few large groups, but only the few tags in small groups can easily be distinguished. Hence, the majority of the attack value is generated by these groups of the smaller sizes while the majority of tags contribute only very little. In Section 3.3.2, we use this insight to design a protocol modification that decreases the attack value sufficiently to deter rational attackers.

Regardless of the other information sources integrated into a large-scale privacy attack, RFID protocol data will be an essential source in any such attack unless the tags are better protected. Limiting the leakage from RFID tags can significantly decrease the attack value, even when all the other (unknown) information sources are left unchanged.

2.5 Attacker Types

Attackers differ in their motivations and capabilities for why they collect RFID information. Some attackers want to track individuals, others want to derive profiling information from the data, and yet others want to spy on their competitors. Understanding the constraints and incentives of the most likely attackers on a system is a crucial step in evaluating which privacy countermeasures are most effective. We model three different attackers in this section and derive appropriate countermeasures in Section 3.3.

2.5.1 Corporate Espionage Attacker

The incorporation of RFID tags into the logistics chain bears the risk of disclosing internal business information to competitors. Several years ago, RFID technology started infiltrating the logistics chain and other domains, where the tags help in automating processes. Soon, RFID tags will be further deployed into the retail space and become accessible to customers and competitors. The retailers that use RFID technology without sufficient privacy protection risk leaking internal business data.

The *corporate espionage attacker* periodically reads tags in the retail space or eavesdrops on readings. For practical matters, access to the tags is restricted to small numbers of snapshots each day and may be limited to certain hours of the day. From the collected information, statistics about internal processes are derived (e.g., turn-over of different goods). The corporate espionage attacker is mostly interested in the number of items and the rate at which the items get replaced. These statistics can be extrapolated from a representative subsample with sufficient accuracy, which makes the espionage attacker relatively resilient to probabilistic privacy protection. In Section 3.3.1 we show that the corporate attacker can likely be defeated by periodically updating the tag secrets, thereby preventing an attacker from correlating different snapshots.

2.5.2 Rational Profiling Attacker

A second attacker on RFID systems uses RFID information to construct customer profiles. The profiles can be used to generate profit through, for example, directed advertisement and price discrimination. Businesses are eagerly building profiles in anticipation of future applications [15] and RFID data is an optimal source to be included in their information bouquet. Currently, there is little available data on the exact value of RFID traces, but the similarity of these traces to valuable Internet traces suggests that collecting RFID data is becoming highly profitable.

The *profiling attacker* acts rationally and will only attack a system if the expected value of the traces exceeds the cost of the attack. A trace herein means several readings of the same tag at different times or locations. The value of different traces varies widely, but the attacker values all traces at the same average price because their true value only becomes evident after the attack (but not even then, necessarily). Decreasing this expected value will prove to be key in defeating the profiling attacker when improving privacy measures, as later state in Section 3.3. Since profiling attackers act rationally, their behavior directly derives from the value function.

2.5.3 Surveillance Attacker

Individuals carrying RFID tags can potentially become victims of surveillance. The surveillance attacker tries to track particular individuals by analyzing RFID readings. The attacker needs to learn significantly more information about an individual than the other two attackers. Anything short of distinguishing a surveillance subject from all other individuals is considered a failure. Because the large amount of information required is not likely to be leaked from a single tag, the attacker has to pool information from several tags belonging to the same subject in order to collect enough information.

As with the profiling attacker, the abilities of the surveillance attacker can be quantified using the success function and value function (Section 2.4.1); the same countermeasures apply. When compared to the profiling attacker, distinguishing several tags is more difficult for the surveillance attacker as the attack will typically be limited to a smaller area, rendering the location meta-

information less useful. On the other hand, when the subject's habits are known, the data can be filtered more efficiently.

Privacy advocates highlight the risks of surveillance posed by RFID systems, but more realistic attackers determined to track a particular individual can employ other tracking means that may often be more effective than using leaked RFID information. Such attacks include face and voice recognition and tagging the individual with RFID tags that are owned by the attacker. The surveillance attacker is therefore perhaps the easiest to defeat when only considering information leakage from RFID tags, but probably the hardest to defeat when all possible information sources are considered. Because of the availability of alternative tracking technologies, even modest levels of privacy protection for RFID tags can increase the cost of tracking through RFID tags beyond the cost of the alternatives. This would prevent RFID information from being included in the tracking, which appears to be the most that the privacy techniques in this dissertation can achieve.

Chapter 3

Protocol Layer Privacy¹

The largest potential source of RFID information leakage is the protocol layer. Protection schemes should hence primarily address this source. While all economically feasible privacy protocols disclose some amount of information, all likely attackers are defeated by the decreased levels of leakage if the protocols are optimized to provide a good tradeoff between cost and privacy. We consider private identification protocols according to the definition given in Section 2.1.

3.1 Probabilistic Protocols

Perfect privacy cannot be achieved in RFID systems since the existence of the tags alone already leaks a small amount of information. Therefore, more privacy on the protocol layer does not necessarily lead to more privacy for the overall system. Furthermore, the protocols that achieve perfect privacy are expensive on the reader side and do not scale to the anticipated size of some RFID systems. More scalable protocols discussed in this section might provide enough protection to defeat all likely attackers.

The protocols described shortly provide *probabilistic privacy*, meaning that an attacker who has compromised the keys of some tags is able to distinguish between certain tags, but not others. The probabilities with which randomly chosen tags can be distinguished depend on which secret keys are known to the attacker.

¹This chapter is partly based on [43].

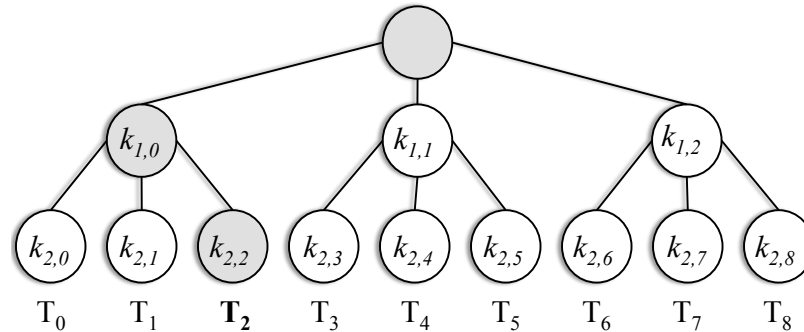


Figure 3.1: Tree of secrets with nine tags. Each tag holds two secrets, one of which is shared with other tags.

Protocols that provide probabilistic privacy have been developed based on the basic hash scheme, altered to share some secrets among several tags. All proposed schemes with probabilistic privacy are special cases of a general *tree-based hash protocol* [39].

The general tree-based hash protocol structures all secret keys in a tree. Each node in the tree holds one secret; each tag is assigned to a different leaf of the tree, and each tag knows all the secrets on the path from the root to its leaf. The tree shown in Figure 3.1 has nine leaves at the lowest level and hence holds nine tags. Each tag in this example tree is assigned two secrets, some of which are shared among several tags. The legitimate reader knows all the secrets and uses the basic hash protocol to identify a given tag. The basic protocol is executed on each tree level to find the branch that the tag resides in. For the example tree, the tag T_2 holds the two highlighted secrets and responds to a reader query with

$$\langle H(k_{1,0}, r), H(k_{2,2}, r), r \rangle$$

where r is a random nonce and $H(\cdot, \cdot)$ is a one-way hash function.

The reader searches for a match of the first hash by hashing the nonce with all secrets on the first tree level. After identifying that the tag resides in the left third of the tree, the reader hashes the nonce with the three remaining possible secrets to identify the tag.

In the most general case, the spreading factor (that is, the number of branches at a tree node) can vary for different tree nodes. Similarly, the depth of the tree can vary for different tags, which

always increases cost and decreases privacy compared to an evenly balanced tree but might become necessary if a partly filled tree needs to be extended.

The tree protocol leads to degraded privacy, since an attacker who is able to extract secrets from some tags is able to identify groups of tags based on which of these secrets they share. Section 3.2 analyzes the effect of stealing secrets from tags on the level of privacy.

Several special-case instantiations of the tree protocol have been proposed as RFID privacy protocols. One protocol uses a binary tree with a spreading factor of two at every node [39]. A binary tree with depth d holds $N = 2^d$ tags. To identify a tag, at most two hashing operations are needed on each tree level, totaling to $2d$ hashing operations (on average, $d + 1$ operations are needed). The cost in terms of hash operations grows logarithmically with the number of tags and is at most $2 \log_2 N$.

A better size-cost ratio is achieved when the branching factor, k , grows with the number of tags. The minimum cost of $dN^{1/d}$ hash operations is realized for $k = N^{1/d}$. This setup also needs fewer secrets on the tags, where storage is particularly expensive, as well as on the reader.

Another protocol related to the tree protocol structures the secrets in a matrix [18]. Each tag is assigned a unique set of secrets; one for each matrix row. As with the tree protocol, to identify the tag, the reader executes the basic hash scheme for each row. The matrix protocol can be viewed as a special case of the general tree protocol with some of the secrets duplicated. When compared to a tree, the matrix provides the same computational cost but much degraded privacy because an attacker learns a larger fraction of secrets from each tag compromise. The only apparent advantage of the matrix protocol is the smaller number of secrets stored on the reader. Since reader storage is not likely to be an important constraint, the tree is generally a better design choice.

The tree and matrix protocols are just two examples from a large design space of tree-like protocols. Proposed variations of the tree protocol maximize privacy under different constraints, such as upper-bounded reading time [13] or tree depth. These variations achieve different, and apparently conflicting, goals. When maximizing privacy for a setup with upper-bounded reading time, the tree grows and has more secrets on the higher level; whereas maximizing privacy in a bounded tree puts more secrets on the lower levels and increases the reading time, as will be shown

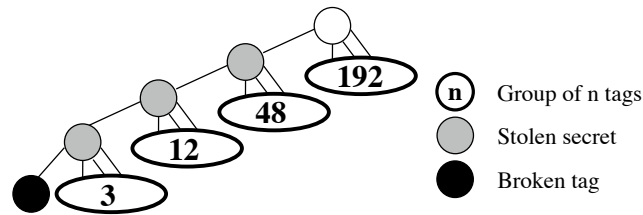


Figure 3.2: Groups of tags that an attacker can distinguish after one tag compromise.

in Section 3.3.2.

3.2 Analyzing Probabilistic Privacy

We analyze the privacy of the tree protocol by estimating its distribution of information leakage as motivated in Section 2.3.3. This privacy distribution depends on the partition of the tree into groups of different sizes and the likelihood that a randomly chosen tag falls into a group of a certain size. Tags are indistinguishable to the attacker if they use the same subset of those secrets that are known to the attacker; that is: whether they fall into the same group.

The amount of information learned from a tag in the tree protocol depends on the tree setup, the set of secrets known to the attacker, and the position of the tag in the tree. If no secrets are known to an attacker, any read could have originated from any of N tags and the attacker faces $\log_2(N)$ bits of uncertainty as to which tag it might be. As secrets are disclosed to the attacker, the tags can be placed in groups of varying sizes. A tag 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* (which is directly related to ϵ -privacy, as shown in Section 2.3).

3.2.1 Single Tag Leakage

As the attacker learns secrets from the tree, smaller groups of tags can be distinguished and strong privacy is lost. When considering a tree with depth d and spreading factor k , the first broken tag gives the attacker d secrets, one on each tree level. These secrets help to distinguish the broken tag from its $(k - 1)$ immediate neighbors, from their $(k^2 - k)$ immediate neighbors, and so on.

Figure 3.2 depicts an example tree of size $N = 256$ with spreading factor $k = 4$. After the bottom left tag is compromised, an attacker can sort the tags into groups of sizes 3, 12, 48, and 192 tags.

For $z = 3, 12, 48, 192$ there are z tags in a group of size z and z/k tags in smaller groups. In our example, there are 48 tags in a group of size 48 and $12+3+1$ tags in smaller groups. For all other values of z there are less than $z + z/k$ tags in groups smaller or equal to z . Therefore, the probability that a randomly chosen tag falls into a group of size z or smaller after the secrets on a single tag have been captured is upper-bounded by

$$\Pr(Z \leq z) \leq \frac{k}{k-1} \cdot \frac{z}{N}. \quad (3.1)$$

The one broken tag is no longer considered part of the system and Equation 3.1 is defined over the range of group sizes actually found in the tree; that is,

$$k-1 \leq z \leq N \cdot \frac{k-1}{k}.$$

To simplify the following calculations, we consider only the upper bound of Equation 3.1. This bound corresponds to the case where one group for each possible size $(1, 2, 3, \dots)$ exists. These groups each hold $k/(k-1)$ tags. While this case is impossible to achieve in reality because a group of size z should have exactly z members, it provides a close upper bound on the real distributions of groups.

This upper bound on the cumulative distribution is shown in Figure 3.3(a) along with the values of the real distribution. Note that the upper bound diverges most from the real distribution for those groups that leak the least information, and least for the more important groups that leak the most information. In our example tree of 256 tags with one broken tag, there are $4/(4-1) = 1.33$ tags in a group of size one, another 1.33 tags in a group of size 2, and so forth. Unlike the real distribution, this bound can be expressed in a closed-form probability distribution.

The probability that a randomly chosen tag leaks x or more nats of information is upper bounded by:

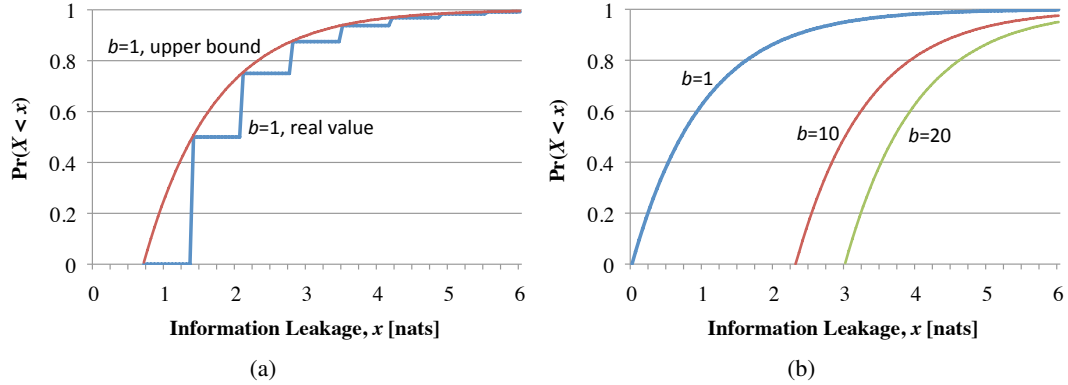


Figure 3.3: Probability that for a tree with b broken tags less than x nats of information are leaked by a randomly selected tag, k is large. (a) real value vs. upper bound, one broken tag; (b) upper bound for different numbers of broken tags.

$$\Pr(X \geq x) \leq \frac{k}{k-1} \cdot \frac{1}{e^x}.$$

This is defined over the range of inputs that correspond to groups actually found in the tree; that is,

$$\ln\left(\frac{k}{k-1}\right) \leq x \leq \ln\left(\frac{N}{k-1}\right).$$

Multiple compromised tags. So far, we have only considered the case of a single broken tag. Assuming tags are evenly distributed, each additional broken tag adds the same number of members to each group (with the exception of the largest group which shrinks in size). For b broken tags², the probability that at least x nats of information for a given tag are disclosed becomes

$$\Pr(X \geq x) \leq b \cdot \frac{k}{k-1} \cdot \frac{1}{e^x} \quad (3.2)$$

for the range

$$\ln\left(\frac{k}{k-b}\right) \leq x \leq \ln\left(\frac{N}{k-1}\right).$$

The probability that the system defeats an attacker who requires at least x nats of information

²Our approximation is closest if $b < k$, but still valid otherwise.

to be successful is: $Pr(X < x) = 1 - Pr(X \geq x)$. Figure 3.3(b) shows this probability for different numbers of broken tags. Note that the distribution is independent of the number of tags in the system and for large trees it is essentially independent of the spreading factor, k . The graphs present a simplified representation of the protocol's privacy. Realistic attackers assign different values to traces with different amounts of entropy rather than deciding whether a trace is valuable based on a simple threshold (as motivated in Section 2.4). All attackers, however, have a higher success probability for higher levels of information leakage and the condensed representation summarizes the overall protection against different attackers.

3.2.2 Multiple Tags

A sophisticated attacker will use information from all tags that a person carries. To calculate the overall privacy of an individual that carries several tags, we apply the idea of merging all leakage distributions into a single distribution as motivated in Section 2.3.3.

Suppose each individual carries m tags that are randomly selected from the tree. To create an identifier for a person, the attacker simply concatenates the information learned from the various tags the person carries. The different identifier that can be built this way separate all individuals into separate groups, which can again be reflected by a distribution of information leakage. The information learned from each of the tags follows the exponential distribution described by Equation 3.2. Summing several exponential distributions leads to a gamma distribution. Hence, the probability that a total of x nats of information are leaked from a person that carries m tags is:

$$Pr(X = x) = \frac{x^{m-1}}{(m-1)!} \cdot b \cdot \frac{k}{k-1} \cdot \frac{1}{e^x}.$$

This distribution is shown in Figure 3.4(a) for different m . The probability that not more than c nats of information are leaked from a randomly chosen tag can be calculated as the integral of this function up to c , which is shown in Figure 3.4(b). Note that the graph shows the probability that the system is *not* compromised, so lower values indicate a higher probability of privacy compromise.

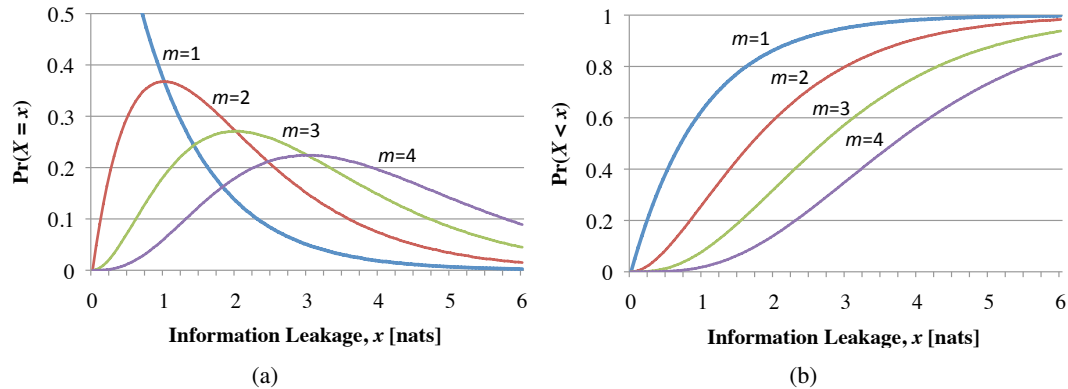


Figure 3.4: (a) Distribution of information leaked by collection of m tags. (b) Probability that information leaked by m tags is smaller than x nats.

3.3 Strengthening Probabilistic Protocols

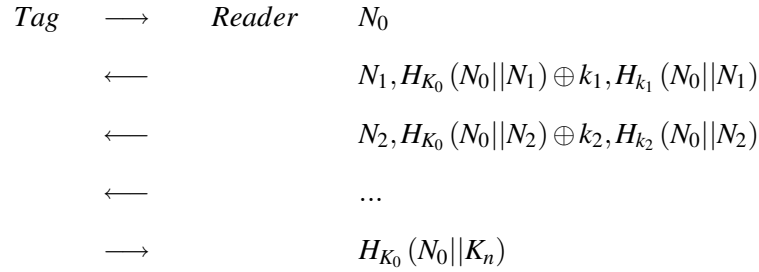
We propose two modifications to the tree protocol and analyze how well they defeat different types of attackers. Section 3.4 describes a more general modification to private identification protocols that can also be applied effectively to the tree protocols.

3.3.1 Updating Secrets

The privacy of the tree protocol can be increased by periodically writing new secrets to the tags. An attacker who reads a tag before and after an update cannot link these two readings. In a retail setting, for instance, tags are constantly within reach of readers to monitor the fill-level of shelves. In this retail setting, secrets could be updated at least every night.

The problem of changing the identifier of a tag has been considered in several ownership transfer protocols in which the old owner only knows one identity and the new owner only knows the other identity [34] [38]. These protocols can also be used to update the keys on the tag to improve privacy, but simpler protocols are sufficient as well.

For example, the following simple key update protocol can be used to update all the keys on a tag:



where k_0 is the unique secret of the tag and k_1, k_2, \dots are the other secrets on the tag that need to be updated. The tag initializes the protocol by sending a fresh nonce, N_0 , to the reader. For each secret to be updated, the reader sends a reader nonce, N_i , and the new secret, k_i , XORed with the hash of the reader nonce, N_i , and the tag nonce, N_0 . An attacker cannot learn the new secret without reproducing this hash, which requires knowledge of the secret k_0 .

The reader also includes a cryptographic integrity check, which is the hash of the tag nonce and each new key. This integrity check ensures that an attacker cannot desynchronize reader and tag by altering the submitted key. Finally, the tag signals to the reader that all secrets were received correctly by signing the tag nonce and the last received secret with the unique tag secret. Only after receiving this proof, the reader removes the tag from its old location in the tree.

To improve privacy, only those secrets need to be updated that are shared between several tags. The tags are therefore only moved around in the tree and no new secrets need to be generated. If forward security is also required, the unique secret, k_0 , must be updated as well, which can also be done with our protocol.

The cryptographic primitives used by our key update protocol, namely a hash function and random number generator, are already available on tags that implement the tree protocol. The key update protocol, therefore, provides a cheap way to refresh the secrets on a tag. Writing to the tag memory, however, requires significantly more power than reading the tag and can only be done over short distances. Should the tag exceed the maximal writing distance during the update, the protocol cannot be completed and all keys need to be reverted to their old values.

The update scheme provides additional privacy only for sequences of readings that are interrupted by at least one update. In the retail setting where tags can safely be updated at least every night, tags might be tracked for a day but not over the course of several days. If the attacker can

learn information from short sequences of readings or when updates are rare, the measure is not effective.

Although the scheme significantly increases the privacy in face of some attackers, it might rapidly eliminate privacy in face of others. An attacker can impersonate a compromised tag during the update process in order to learn more secrets. If the system has no way of knowing which tags are compromised, the attacker can learn a new set of secrets in each update round. Soon, the attacker knows all secrets and privacy is lost. To avoid leaking new secrets to that attacker, the system must notice the break of tags and exclude them from future updates. Some systems such as those designed to detect counterfeit tags already attempt to provide this property by checking whether readings are reasonable with respect to past readings [59]. To circumvent detection, an attacker would have to steal a tag, compromise it, and start impersonating it before the tag is reported as missing. While this might be easy in some scenarios, in other scenarios, such as retail, tags can be constantly tracked. Tags that go missing for some time and turn up again should be excluded from the update process. Unless this happens frequently, the attacker still has only a small chance of tracking tags for longer periods of time.

In summary, the update protocol improves security in an environment where an attacker has only limited access and where tag compromise can be detected. In scenarios where these assumptions hold (such as warehouses and retail spaces), periodic key updates can increase privacy at reasonable cost. Note that for the same scenarios the hash chain protocol provides a good solution [47]. For other scenarios, key updates weaken privacy by disclosing new secrets to the attacker through the update protocol. As postulated before, the effectiveness of a measure must be seen in light of an attacker model and the update protocol is effective only against the corporate espionage attacker.

3.3.2 Varying Tree Structure

The tree protocol provides very different levels of privacy to different users depending on the user's location in the tree relative to the secrets known to the attacker. The privacy follows an exponential or gamma distribution as derived in Section 3.2—a few tags in small groups leak large amounts of information, while the other tags in larger groups leak very little information. The insight that

the bulk of the attack value is generated by the tags in smaller groups points to a simple but very effective improvement of the tree protocol: restructure the tree so that no tags fall in groups smaller than some threshold. Moving tags from these smallest groups to larger groups decreases the attack value dramatically.

The least-cost tree for a given tree depth, d , is a balanced, fixed- k tree that has a constant spreading factor of $k = N^{1/d}$. We assume that d is fixed because it is limited by the tag memory and by the maximum reading time. Varying the spreading factor only at the last level of the tree significantly improves privacy.

The privacy-improved tree is constructed by dividing the tags into groups of some threshold size k_{last} and constructing a balanced, fixed- k tree with these groups as the leaves. The threshold is chosen to be larger than the largest group size that still carries a high value to the attacker. The computational time required to find a tag is the time required to find the correct leaf plus the time to identify the correct tag within that leaf:

$$Cost = \left(\frac{N}{k_{last}} \right)^{\frac{1}{d-1}} \cdot (d-1) + k_{last}.$$

In comparison, the balanced tree has a maximal cost of $dN^{1/d}$. Increasing k_{last} increases privacy and computational cost. The attack value of the modified tree is computed using Equation 2.2 but with a larger minimum group size. The attack value is:

$$Value(k_{last}) = \int_{\ln(\frac{k}{k-b})}^{\ln(\frac{N}{k_{last}-1})} (Su(x) \cdot Pr(X=x)) \cdot dx.$$

For each choice of the minimum group size, k_{last} , we get a different point on the tradeoff curve between privacy and cost that is depicted in Figure 3.5 for an example scenario. The example represents a large system with many broken tags and many tags per person, specifically $N = 10^7$, $d = 4$, $b = 50$, $m = 10$, $P = \frac{1}{2}$, and $t = 100$.

A fixed- k tree for this setup requires a spreading factor of 56. Increasing the size of the groups on the last level beyond 56 decreases the attack value. It falls to 40% of its original value at a group size of 1,100 tags, which increases the reader cost nine times. Limiting the attack value to

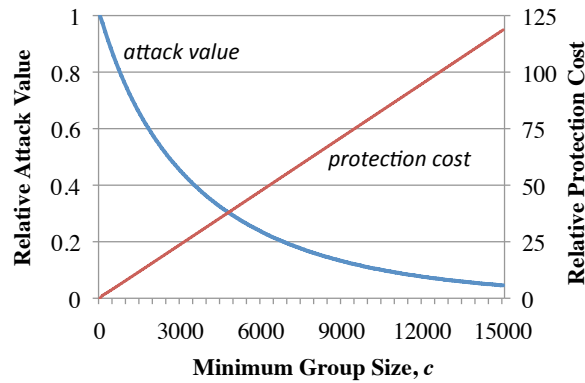


Figure 3.5: Tradeoff between attack value and computational cost when varying the tree structure. The attacker value function from Section 2.5.2 is used with $p=0.5$ and $t=100$. The analyzed system holds one million tags, of which 50 tags are broken; each individual carries 10 tags, and each tag stores four secrets.

20% leads to a 30-fold cost increase. These design points correspond to 1,139 and 3,729 hashing operations, which is still orders of magnitude below the 10^7 operations required to reduce the attack value to zero using the basic hashing scheme with no shared keys. The appropriate tree for many RFID privacy scenarios therefore only has two levels and is hence the opposite configuration from the initially proposed binary tree [39].

For a large group of attackers on many RFID systems, minimal group sizes of \sqrt{N} provide sufficient privacy. The resulting tree has only two levels, which requires smaller memories on the tag and leads to quicker reading times when compared to a deeper tree. In a system with one billion tags, each read requires about 6,000 hashing operations, which takes only a fraction of a second on a general-purpose processor.

3.4 Improving Privacy by Adding Noise

The privacy of probabilistic privacy protocols derives from the fact that an attacker only knows a small fraction of the keys in a system while the legitimate reader knows all keys. The gap in the ability to distinguish users can be amplified by adding noise to user responses. The noise blurs the borders between groups of users that the attacker would otherwise be able to distinguish.

Our technique for improving the tree protocol's privacy 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 each user, the last level of the tree is not randomized. Where as before an attacker could deterministically identify a user as being a member of a single group, the randomization means an attacker can only determine the probability that the user is in each group.

The randomization never leads to false identifications, because the last round that uses the unique secret of the tag is not randomized. While the legitimate reader will always be able to correctly identify a tag, some wrong tree branches might be evaluated before the correct branch, thereby increasing the identification cost. Randomization enables privacy levels anywhere in between the deterministic tree-protocol and the basic hash protocol at varying costs.

In a simple instantiation of our randomization technique, the user generates a nonce, then hashes that nonce and a 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 as $\frac{1}{2}$, even the legitimate reader has no advantage in identifying the user over trying all keys until the right one is found. If p is chosen suitably, it can be close enough to $\frac{1}{2}$ to provide little information to a rogue reader, but far enough from $\frac{1}{2}$ to provide useful information to a legitimate reader. The legitimate reader only has to try a few groups to find the right user, while an attacker is left with a large uncertainty and cannot conclusively decide to which of the groups a user belongs.

The next subsection analyzes the privacy properties of the proposed scheme, and Section 3.4.2 estimates the costs a legitimate reader incurs for different choices of p . In Section 3.4.3, we present and analyze a variation on the simple randomization technique that provides improved privacy-cost tradeoffs. Our analysis here assumes an attacker cannot obtain multiple reads known to be from the same tag; in Section 3.4.5, we consider the problem of multiple reads. Randomization provides privacy options anywhere in between the deterministic tree-protocol and public key cryptography; the new design tradeoffs offer cheaper ways to achieve privacy requirements for many systems.

3.4.1 Privacy Analysis

In a two-level tree, the secrets on the second level are unique and no information is leaked from this level. We assume the strongest possible attacker who knows all secrets on the first level but none

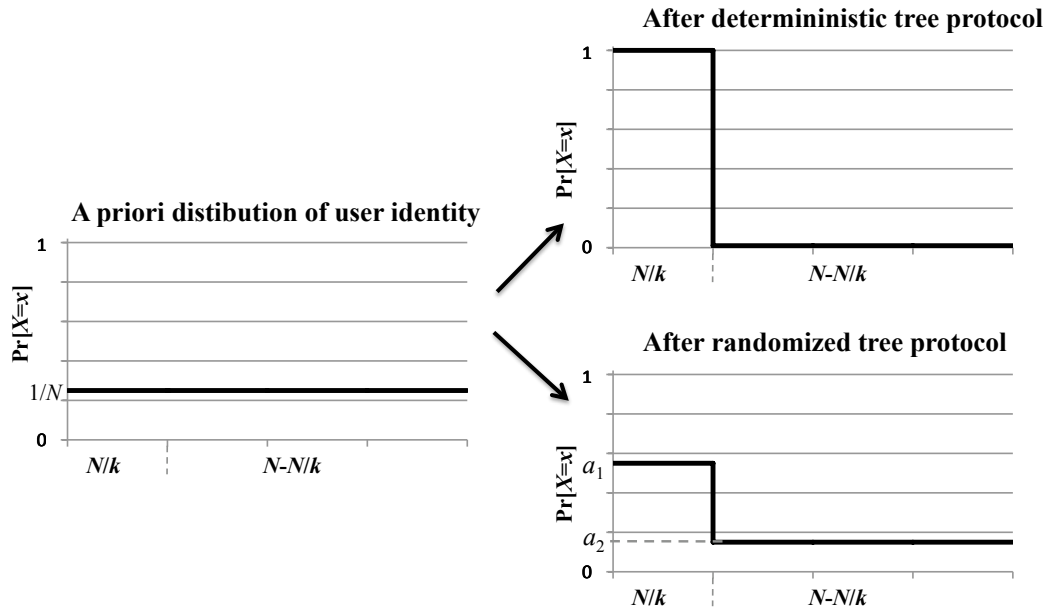


Figure 3.6: Change in probability distribution of user identity as seen by attacker compared for deterministic and randomized tree protocol. Tree with N users, spreading factor k .

of the second level secrets (the secrets on the second level are not shared among tags and hence not stealable). Note that we do not think that any attacker will ever be able to access all the first-level secrets, as that would involve physically opening tens of thousands of tags. Our calculations, therefore, provide a lower bound on privacy. The attacker's goal is, once again, to distinguish between tags as accurately as possible.

Figure 3.6 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 $\frac{N}{k}$ and leaks $\log_2 k$ bits of information, where k is the spreading factor on the first tree level. When randomization is used, the attacker only learns with what probability the user resides in the different branches.

The amount of entropy (and information leakage) depends on the probability an attacker guesses that the correct secret was used (marked as a_1 in Figure 3.6) and the probability that the attacker guesses that each of the other secrets was used (a_2 in the graph). These *correct guess* and *wrong guess probabilities* are directly related to the degree of randomization, p . The correct guess prob-

ability is the chance that a received response corresponds to the hash output using an assumed key plus randomization. The wrong guess probability corresponds to the case that the response corresponds to the output from some other key. The closer these two probabilities are, the more privacy is provided since an attacker can no longer decide whether or not a certain key is used.

The correct guess probability is calculated as the likelihood that a given level of noise is the result of randomization summed over all possible levels of noise. Each chance corresponds to the probability that a certain level of noise was produced by a binomial distribution, $\text{Binom}(i, n, p)$, which stands for the chance that i of n bits in the output are flipped for randomization of degree p .

The wrong guess probability is conversely derived from an unbiased distribution, $\text{Binom}(i, n, \frac{1}{2})$. The ratio between correct and wrong guess probability, $r = \frac{a_1}{a_2}$, is calculated as:

$$r = \sum_{i=0}^n \frac{\text{Binom}(i, n, p)^2}{\text{Binom}(i, n, \frac{1}{2})}.$$

Theorem 3.1. *The average entropy of tags in the randomized tree protocol as seen by an attacker is:*

$$E = \log\left(\frac{N}{k}\right) + \log(k + r - 1) - \frac{r}{k + r - 1} \log(r). \quad (3.3)$$

Proof. 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 $\frac{1}{N}$, 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 protocol, an attacker never learns the exact branch in which a user resides but rather a probability distribution over the different branches as illustrated in Figure 3.6. 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 (i.e., information leakage) only depends on the

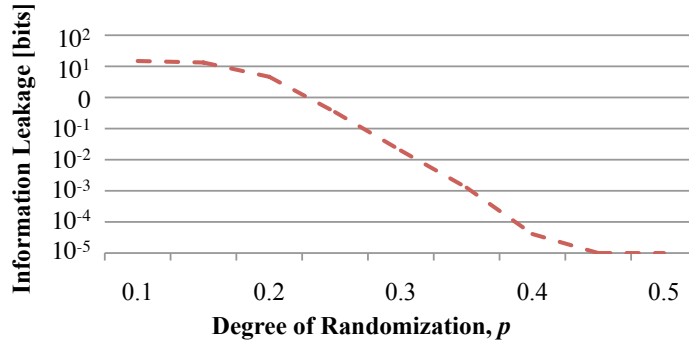


Figure 3.7: Workload required for one identification in deterministic tree with no randomization vs. randomized tree. System with 1 billion users, tree with depth 2. Average values over 100k simulations each.

difference of these two probabilities and the tree parameters N and k .

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 :

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

■

The first term is the maximum entropy of a group of size N ; the second term is the amount of entropy given up by the deterministic tree protocol when the attacker knows all secrets on the first level; and the remaining two terms describe the entropy gain through randomization. The border cases correspond to the linear protocol ($r = 1$, no information leaked) and the tree protocol ($r = \infty$, completely deterministic grouping of users).

Note that the probability distribution shown in the Figure 3.6 depicts the average probabilities over all possible user responses. For most single responses the distribution will be very different.

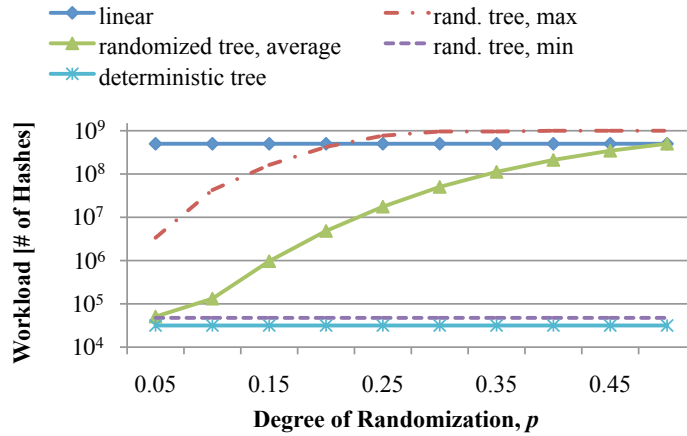


Figure 3.8: Workload required by different protocols for each identification. System with 1 billion users, tree with depth 2. Values are averaged over 100k simulations each.

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.

Figure 3.7 shows the different amounts of information leakage achieved by varying the degree of randomization. The amount of disclosed information decreases roughly exponentially with the degree of randomization (and the attacker advantage, ϵ , decreases roughly linearly). This is a conservative approximation since we are still assuming that the attacker knows all secrets on the first tree level. For high levels of randomization, the information leakage drops to virtually zero. For a two-level tree with one billion tags, for example, the information leakage drops to $1/10,000^h$ of its original value for $r = 19$.

3.4.2 Cost Analysis

Adding noise increases the reader cost of each identification. This verification overhead grows as more bits of the user's responses are randomized, as shown in Figure 3.8. In the unmodified tree protocol with a tree of height d and spreading factor k , an average of $\frac{1}{2}kd$ hashing operations are needed for every identification. When adding randomization, in the very unlikely worst case, the entire tree would need to be evaluated for a single identification (a sensible implementation would cut off the search once the probability drops below some threshold to avoid searching the entire tree on a misread).

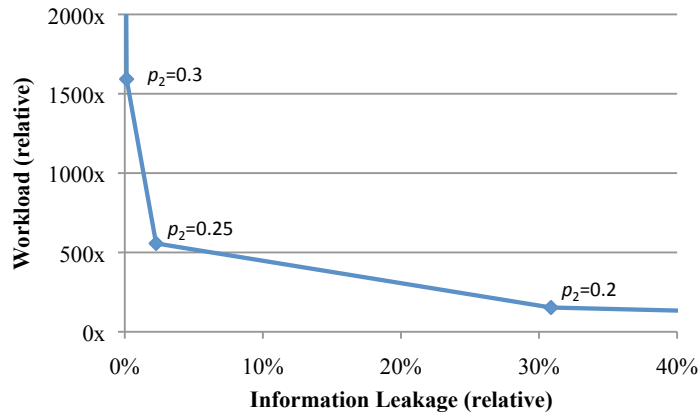


Figure 3.9: Design space for *simple randomization* ($p_1 = 1$). System with one billion users, two-level tree. Values are averaged over 100k simulations each.

To estimate the expected cost of the randomization, we simulated the server workload for a large number of possible parameters. We chose a simple depth-first search strategy for the server where the branches are evaluated in order of their initial probability of containing the match. We chose this search strategy merely for its simplicity, although more adaptive strategies might lead to lower costs. Note, that for the two-level tree, which is optimal for many applications, there is no difference between the simple and adaptive strategies.

Since the average cost grows roughly exponentially with the degree of randomization as depicted in Figure 3.7 and information leakage decreases exponentially with randomization, the trade-off between information leakage and cost is roughly linear.

The degree to which privacy is improved through randomization varies with the height of the tree. Two-level trees have better privacy than deeper trees and are therefore the preferred setup for our randomization [44]. Normally, the opportunity to increase privacy by decreasing the height of the tree should be fully exhausted before adding randomization if the goal is extremely low information leakage. Changing a three-level tree with one billion users into a two-level tree decreases information leakage by 80% while increasing identification cost by 60x. Randomization costs about twice as much to achieve the first 80% reduction in information leakage and increasingly more for additional reductions.

The randomized tree protocol provides design options 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 tradeoff between randomization and information leakage as depicted in Figure 3.7 and the tradeoff between randomization and cost in Figure 3.8. The resulting tradeoff curve is shown in Figure 3.9. In all the figures, we assume a large system with one billion users and a two-level tree in which the attacker has acquired all first-level keys.

3.4.3 Selective Randomization

To reach more points in the design space, most 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 $p_1 n$ 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 $p_1 = 1$.

Selective randomization leads to a distribution with the same expected number of flipped bits as the simple randomization with $p = p_1 p_2$, but the actual distribution is more concentrated around this average. In particular, no value with more than $p_1 n$ flipped bits can be reached. This constraint helps the attacker because some users are known to not have generated some responses. For well-chosen p_1 and p_2 , however, the probability is very high that at least a few of the wrong secrets could have generated a response.

We calculated the amounts of entropy that the selective randomization preserve 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 3.10. The design space includes one design point, $p_1 = 0.8, p_2 = 0.25$, where only $1/50^{th}$ as much information is leaked when compared to the deterministic tree (i.e., information leakage is 98% lower) and cost increases 22 times; another design point, $p_1 = 0.8, p_2 = 0.35$, decreases information leakage to $1/2500^{th}$ its original value and increases cost by a factor of 304.

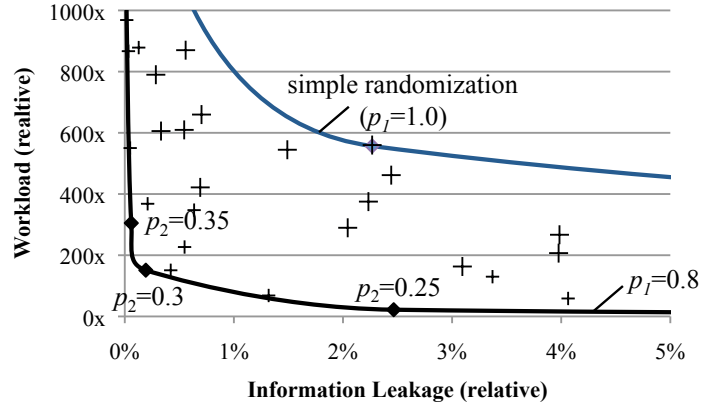


Figure 3.10: Design space for *selective randomization*. Each cross corresponds to one p_1, p_2 choice. The lower line corresponds to $p_1 = 0.8$ and various choices for p_2 , while the upper line corresponds to the simple randomization ($p_1 = 1$). System with one billion users, two-level tree. Values are averaged over 100k simulations each.

3.4.4 Cost of Selective Randomization

The computational cost of each legitimate identification grows with the level of randomization. To enable further analysis of the randomization idea, this section provides a closed-form approximation of the cost of randomization.

To find the overall cost of one identification, we calculate the number of leaf groups that need to be evaluated until the correct leaf group is found. Note, that we are not considering the small cost of evaluating the higher tree level(s) to further simplify the analysis.

For a received response that deviates in x bits from the n -bit hash of the correct secret, the probability that a wrong secret matches in more bits is calculated as the cumulative distribution function (CDF) of random matching, $(1 - \text{Binom}_{CDF}(x, n, 1/2))$. The number of secrets evaluated in each group before the right secret is evaluated for a given deviation is therefore this probability multiplied with the number of wrong secrets in the group, $(k - 1)$.

The average number of evaluated secrets is this number multiplied by the chance that the assumed deviation occurs, $\text{Binom}(x, n \cdot p_1, p_2)$, summed over all possible deviations. Finally, the total number of groups that need to be evaluated is calculated as the number of groups that need to be

evaluated on each level raised to the number of levels in the tree:

$$\text{Cost} = \left(\sum_{i=0}^{n \cdot p_1} \left(\text{Binom}_{PDF}(i, n \cdot p_1, p_2) \cdot (k-1) \cdot \left(1 - \text{Binom}_{CDF}\left(i, n, \frac{1}{2}\right) \right) \right) \right)^{(d-1)} \cdot k_{last}. \quad (3.4)$$

This gives the expected number of hashes that must be performed by a legitimate reader to identify a tag using the selective randomization protocol.

3.4.5 Multiple Readings

Our analysis has so far assumed that an attacker cannot obtain multiple readings that are known to come from the same tag. If an attacker can learn several such responses, 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.

The deterministic tree protocol lets the attacker decide on the correct key (that is, the correct group) with 100% probability. When adding enough noise, the attacker can no longer decide with certainty under which key a given response was generated. With multiple readings known to come from the same tag, however, an attacker may be able to determine the group with near certainty even with randomization.

We still assume the worst-case scenario where the attacker knows all secrets on a given tree level. The best attacker strategy for identifying the correct secret is by first computing the probability for each of the secrets that a certain deviation was caused by added noise and then choosing the secret with the highest probability. For a single read that deviates from the computed response in x bits, the probability that the deviation was caused by added noise is described by the binomial distribution

$$Pr_{correct}(x) = \text{Binom}(x, n \cdot p_1, p_2).$$

Conversely, the probability that a response generated under a different secret randomly matches in x bits is:

$$Pr_{wrong}(x) = Binom\left(x, n, \frac{1}{2}\right).$$

The ratio between the correct secret's probability of having a certain deviation and the probability that any of the secrets have that deviation is:

$$\frac{Pr_{correct}(x)}{Pr_{correct}(x) + (k-1) \cdot Pr_{wrong}(x)}$$

The average probability that the correct secret is chosen after a single read is this ratio multiplied by the probability that this deviation occurs, $Pr_{correct}(x)$. Summing over all possible deviations:

$$Pr_{identification} = \sum_{i=0}^{n \cdot p_1} \left(\frac{Pr_{correct}(i)^2}{Pr_{correct}(i) + (k-1) \cdot Pr_{wrong}(i)} \right). \quad (3.5)$$

The equation assumes independence of the different secrets, which is approximately accurate when k is large.

In order to calculate the probabilities for the case where the attacker combines several readings, we first have to convert the binomial distributions into more flexible normal distributions. The binomial distributions are closely approximated by normal distributions with expected values $\mu_{correct} = np_1p_2$ and $\mu_{wrong} = n\frac{1}{2}$; and standard deviations $\sigma_{correct}^2 = np_2(1-p_2)$ and $\sigma_{wrong}^2 = n\frac{1}{4}$.

The effect of multiple reads can be expressed as the average sum of several such normal distributions (one for each read). According to the *weak law of large numbers*, the average sum of r equally distributed normal distributions is also a normal distribution with the same expected value, but a standard deviation that is r times smaller than that of the normal distributions.

The probability that the correct secret matches in x bits averaged over r reads is approximately:

$$Pr_{correct}(x, r) = Normal\left(x, \mu = np_1p_2, \sigma^2 = \frac{1}{r}np_2(1-p_2)\right).$$

The probability that the responses randomly match on average in x bits, when an incorrect secret is assumed is:

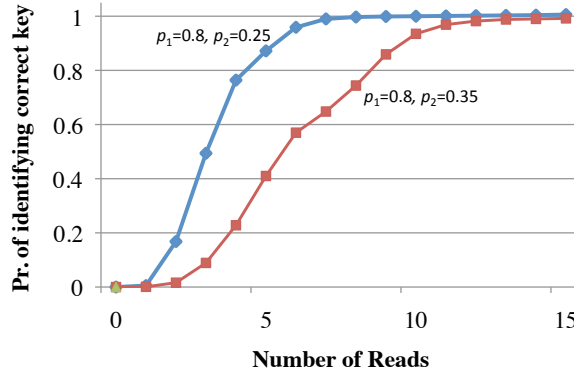


Figure 3.11: Effect of multiple readings on the level of privacy provided by the randomized tree protocol. Probability that the right group (but not necessarily the right key) is identified.

$$Pr_{wrong}(x, r) = Normal\left(x, \mu = n\frac{1}{2}, \sigma^2 = \frac{1}{r}n\frac{1}{4}\right).$$

Substituting these probabilities in Equation 3.3 provides the average probability that an attacker can identify the correct secret (and hence the correct group) as a function of the number of reads.

The probability of successfully identifying the correct group when combining multiple reads is depicted in Figure 3.11 for the two example parameterizations of the selective randomization. For both parameterizations, the effect of multiple reads quickly amortizes the privacy benefits of the randomization. Over half of the benefits are lost after 4 reads in the scenario with 98% privacy gain and after 6 reads in the scenario where information leakage is decreased to $1/2500^{th}$ of its original value. After a larger number of reads, the effect of randomization is completely lost. Note, these are worst-case estimates, where the attacker knows all secrets on one tree level and only a single RFID tag is present in the reader field. If the attacker does not know some of the secrets or if multiple tags are present at the same time, the negative effect of multiple reads is much less dramatic.

Defenses. To prevent an attacker from collecting a large number of readings, while maintaining the functionality of the tag, a tag should 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 by storing the once-computed and randomized hash in capacitor-backed RAM. When the tag leaves 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 behavior will prevent an attacker who deploys rogue readers in an area where users travel from learning several responses that are known to be from the same user. Attackers who have access to a unique user for a long period of time, however, would be able to obtain multiple readings by keeping the reader field off for most of the time. (Note that we assume an attacker who obtains physical access to a user device can extract the keys directly, so the fact that an attacker with physical access can obtain multiple readings easily is not a major concern.)

3.5 Feasibility

The tree-based hash protocol with a two-level tree and added noise for improved privacy leaks very little information. In this section, we quantify the cost and privacy of the protocol for a realistic implementation scenario and estimate the server resources needed to operate it. We find that close to zero information leakage can be achieved at low cost even for very large systems. We only discuss one example scenario to motivate the need for cheap cryptographic functions to maintain a low cost of privacy. In this scenario, we assume a two-level tree with fixed spreading factor. A larger design space with more design options is presented in Chapter 6.

We consider a system with one billion tags (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 approximately 32,000. The attacker is conservatively assumed to know all the secrets on the first level.

To compare the cost of probabilistic protocols to the cost of alternatives such as public key cryptography, we translate the cost of the selectively randomized tree protocol into concrete running time on dedicated hardware. In the deterministic key protocol, the reader computes 32,000 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,

where about 20 cores fit on a single high-end FPGA chip [60]. As some level of protection is already sacrificed in symmetric protocols to make them scalable, ciphers weaker than AES might still provide sufficient protection. A smaller algorithm such as Tiny Encryption Algorithm (TEA) and its relatives XTEA and XXTEA can be implemented more efficiently in the same hardware and perhaps provide a better fit for RFID tags [30]. Even more efficient ciphers such as A5/1 achieve up to 2 billion operations per second on a single FPGA.³

A hash function for privacy protocols 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. We show in Section 4.2 that cheaper alternatives can well provide the cryptographic strength needed for probabilistic privacy. Another alternative is to use probabilistic hash functions such as HB protocols [32] that can potentially build a very low-cost one-way function.

Public key cryptography that could also be used to provide privacy compares unfavorably to all symmetric-key alternatives (besides exceeding the implementation cost of RFID tags). In comparison to symmetric one-way functions, public key cryptography, such as Elliptic Curve Cryptography or RSA, is much more expensive to implement in hardware. One high-end FPGA fits about 10 instances of RSA that each compute about 30 operations per second [61]. RSA is hence about two million times less efficient than AES and seven million times less efficient than A5/1 in terms of operations per area and time. RSA is about 100 million times less efficient than *CRC-MAC*, the private hash function we propose in Section 5.3.

On the other hand, only a single RSA operation is required for each identification. One identification using public key cryptography, therefore, is as efficient in server hardware as the tree protocol with two million AES hashing operations, which also is the cost of decreasing the information leakage of the tree protocol by more than 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 server cost and RFID implementation. Furthermore, in applications where larger amounts of information leakage are tolerable, the randomized tree protocol provides a tradeoff between more privacy and less cost.

³Thanks to David Hulton from Pico Computing for this estimate.

State of the art implementations of standard hash functions provide hundreds of Mbit/s [54] of throughput, which corresponds to millions of hashing operations per second. A hashing computer built from many of these chips can execute on the order of a billion hashing operations per second. For the system with a billion tags, this would support authenticating more than 100,000 tags per second on a single server. In Section 4.2 we propose a hash function that is several orders of magnitude cheaper, making the overhead cost of privacy protection negligible.

Chapter 4

Implementing Privacy Protocols

To implement private identification protocols on RFID tags, a one-way hash function is required. Standard functions could be used for this purpose, such as cryptographic hashes and block ciphers, but are at least one order of magnitude too large and power-hungry to fit on RFID tags. At the same time, these functions provide features such as collision-resistance or decryption, which are not needed to achieve privacy. This chapter presents two design ideas for one-way hash functions that achieve the properties needed for privacy but stay well below the cost of alternative cryptographic primitives.

The first approach uses noise-based privacy protocols that were proposed for private authentication and converts them to a hash function for privacy. While these noise-based hash functions can be implemented with minimal hardware overhead, they lead to significant communication overhead. We present a simple technique to curb this overhead in Section 4.1.4. We propose a second approach for privacy protocols that uses a new class of cryptographic primitives. These *private hash functions* operate similarly to cryptographic hashes without providing their most expensive property, collision resistance, which is not necessary for privacy. Thus, the implementation footprint and power consumption of private hash functions can be much lower than that required for known cryptographic hash functions.

4.1 Noise-based Hash Functions¹

Several authentication protocols have been proposed that build their security on simple functions such as the dot-product of binary strings [9]. Because these simple functions are easy to invert when several instances are given, noise is added to obfuscate the secret inputs. We propose using these noise-based protocols as a hash function in the randomized tree protocol. Note that these cheap hash functions require noise to protect the secret key, while the protocols proposed in the last chapter use noise to enhance privacy. It matters little to the privacy protocol whether outputs from a deterministic hash function are randomized or the hash function produces already-randomized outputs. When cheap HB hash functions are used, noise needs to be added only once to achieve irreversibility and privacy. The amount of randomization needed will often be different for irreversibility and privacy. However, we find that the levels of randomization we showed to provide high levels of randomization in Section 3.4.3 also provide sufficient irreversibility, as we discuss in Section 4.1.3.

4.1.1 HB-based Security

Randomizing user responses was previously used to achieve privacy in RFID systems by the HB family of protocols that were originally developed by Nicholas Hopper and Manual Blum to support authentication by humans without computer assistance [9]. These protocols use 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 security of the HB hash functions relies on the hardness of the *learning parity with noise* (LPN) problem that has not conclusively been shown to be computationally hard.

Juels and Weis proposed using adapted HB protocols on RFID tags since the simple operations of the functions can be implemented very efficiently in hardware [32]. In their HB+ protocol, tags identify themselves to the reader privately through a simple challenge-response protocol:

¹Research for this section was done in collaboration with Mate Soos.

$$\text{Reader} \rightarrow \text{Tag} : R_1, \dots, R_n$$

$$\text{Tag} \rightarrow \text{Reader} : (k \times R_1) \oplus z_1, \dots, (k \times R_n) \oplus z_n$$

R_i are random strings chosen by the reader, k is the secret key shared by tag and reader, \times denotes dot-product, and each z_i is a biased coin toss that is $\mathbf{1}$ with fixed probability $p < \frac{1}{2}$. The number of rounds n depends on this level of added noise—the closer p is to $\frac{1}{2}$, the more rounds are needed to be reasonably certain that the sender of the response knew the secret key. The hardness of finding the secret key from responses depends on p , the length of the key, and the number of response bits available. Assuming $p = 0.25$ and a virtually unlimited number of response bits the secret key is hard to find only if the key is at least 500 bits long [8]. The same attack will not find the secret key faster than brute force when the number of response bits is limited to a reasonable number. Instead of using such long keys, that can neither be stored on the tag, nor processed n times within the timing constraints, we propose to limit the number of responses an attacker can collect, as explained in Section 4.1.3.

HB+ was proven secure against active attackers in a limited attacker model [32], but later shown to be vulnerable against very practical attacks that are outside of the scope of the proofs [27]. In this attacker-in-the-middle attack, the attacker flips bits in challenges sent by a legitimate reader to a tag and learns key bits from observing whether or not the protocol succeeds. Improved variants of the function have not yet received sufficient scrutiny to assess their security [40]. A second drawback of HB functions, besides their vulnerability to active attacks, is the high levels of noise and the large number of rounds they require for the LPN problem to be hard to invert. We mitigate these problems by using HB functions in the privacy setting where no active attacks are possible and where the rate of responses can be limited (and has to be limited when the randomized tree protocol is used).

4.1.2 HB-based Privacy

When HB-based functions are used to achieve privacy, all data is generated and sent by the tag:

$$\text{Tag} \rightarrow \text{Reader} : R_1, (k \times R_1) \oplus z_1, \dots, R_n, (k \times R_n) \oplus z_n.$$

Active attacks against the tag (on the protocol layer) are therefore not possible by design. What might still be possible is recovering the secret key from user responses when the level of noise is too low or many user responses are available to the attacker. Since the privacy setting differs from the authentication setting, we developed new guidelines on how to choose parameters for the degree of randomization, p , the number of response bits, n , and the size of the secret key, $|k|$. These parameters need to be chosen in accordance with choices on the (tree-)protocol layer in order to achieve the best tradeoffs.

When combining the randomized tree protocol and HB hash function, noise serves two purposes: privacy on the protocol layer and non-invertibility on the hash function level. Non-invertibility is achieved in HB functions by obfuscating the responses so that the key cannot be learned. It strictly grows with the level of noise. Privacy is achieved by obfuscating the responses so that even when the keys are known, it is not entirely clear which key was used to generate a response. Privacy also grows with the level of noise, but is further influenced by the number of secrets that could have been used to generate a response. In a two-level tree with 1 billion tags, for instance, about 32,000 (2^{15}) secrets could have generated each response. Any non-randomized hash function would, therefore, need more than 15 bits of output to generate unambiguous responses most of the time. When privacy is the goal instead of non-ambiguity, just slightly more than 15 response bits suffice since each response is further randomized to intentionally make responses ambiguous. For a two level tree, for example, $n = 18$ bits suffices for the first level. For the second level, larger responses are needed to prevent ambiguous responses. A choice of 45 bits for the second level keeps the probability of an ambiguous identification smaller than one in a billion. Assuming 64-bit secret keys for now, the total amount of data sent in one identification is half a Kbyte for both levels combined. We show in Section 4.1.3 how this communication overhead can greatly be reduced. Note, that HB protocols are by design probabilistic and therefore cannot completely rule out false identifications. However, when parametrized so that the chance of a false identification is less than one in a billion, we argue that a protocol with HB hash function provides the correctness property in a practical sense.

The only parameter left to be determined is the size of the secret key. The key must be large

enough to be difficult to find through brute-force or other attacks. Most known attacks on LPN that are faster than brute-force, require a very large number of samples. The most efficient published attack can break LPN instances with $p = 0.25$ for keys up to some 200 bits, but requires many million samples [8]. By limiting the amount of samples that an attacker can collect, the cost of these attacks increases exponentially and quickly exceeds the cost of brute force. One recent (not yet published) attack requires many fewer samples to break LPN keys in reasonable time [28]. For $p = 0.25$, keys of 96 bits can be broken in about a month on a PC given only 300 samples and within about 17 hours, when given 10,000 samples. The attacks become exponentially more expensive as the level of added noise, p , increases.

The level of noise must be chosen to make the LPN problem hard to invert and, at the same time, provide enough privacy in the randomized tree protocol. As calculated in Section 3.4.3, levels of noise around $p = 0.3$ provide optimal design choices for scenarios where good privacy protection is required. At this noise level, breaking 64-bit keys becomes fairly expensive at about 3 days computational time when given 10,000 samples using the best known attack [28]. Note, that this security level is only an upper bound on LPN's security that will likely be improved as LPN solving continues to be a topic of active research. When assuming that the current attacks are not far from the best possible attacks, LPN provides sufficient privacy protection when the number of samples can be limited to below 10,000.

4.1.3 Rate-Limiting Responses

The number of samples an attacker can collect from a single RFID tag is naturally limited by the speed of the tag and the time an attacker can interact with the tag. The randomized tree protocol already requires the answer rate to be significantly lower than this limit so an attacker can only collect very few samples in each physical location. Note that rate limitation was implicitly assumed when HB protocols were first proposed as an authentication protocol for humans—there simply is no way to get a human to compute millions of dot products. As argued in Section 3.4.5, rate limitation for RFID tags can easily be achieved through capacitor-backed RAM on the tag. If the capacitor is sized to buffer responses for one minute, an attacker needs constant access to the tag

for more than 6 days to collect the required 10,000 samples. This level of access is outside of the attacker model we consider; especially because the attacker would need to keep the reader field mostly switched off thereby forgoing the possibility of reading other passing tags.

4.1.4 Limiting Nonce Entropy

To make our approach economic and implementable on an RFID tag, there is one last obstacle to overcome: just temporarily storing half a Kbyte of data in the capacitor-backed RAM is expensive and might increase the overhead of the hash function beyond the cost of alternatives. We resolve this problem by only storing a seed from which the responses can be recreated. The k -bit seed is (part of) one random nonce from which all the other nonces are derived. In hardware, pseudorandom sequences can be generated cheaply using a linear feedback shift register (LFSR). This measure decreases the entropy of the nonces from $(18 + 45) \cdot 64 = 4032$ bits to k bits. Generating all nonces from a single seed enables the response to be stored in $k + 63$ bits of capacitor-backed RAM.

To understand why the lower entropy does not impact privacy in any practical sense, recall that the attacker already gains a small advantage in the privacy game because of protocol layer information leakage. Through limiting the entropy of the nonces, the attacker wins the game negligibly more often even for relatively small values of k . The attacker on average learns that two out of every $2^{k/2}$ readings did not originate from the same tag (by the birthday paradox) and learns that two out of every 2^k readings did originate from the same tag. Expressed in information leakage, an additional $2.2 \cdot 10^{-5}$ bits of information are leaked for $k = 32$ bits.

An additional advantage of generating the nonces from a small seed is that instead of the nonces, only the seed has to be sent. The communication overhead hence drops from 4,000 bits to $k + 63$ bits.

Summary. All obstacles of HB protocols can be overcome when used as a hash function for privacy. Active attacks are not possible by design when all data is generated on the tag. Revealing the secret key by attacking the underlying LPN problem is infeasible when keys are too large for brute-force and sampling is rate-limited to prevent all other known attacks. Such rate limitation can be achieved by generating responses from a seed stored in a battery-backed RAM. The lower

entropy of the seed has no practical effect on privacy, but highly improves the communication-efficiency of the HB function. However, the hardness assumption that all HB protocols rely on may need to be revised considering that recent research has made progress on solving LPN instances and further progress appears likely.

4.2 Private Hash Functions

Another possibility for implementing privacy protocols are cryptographic hash functions or block ciphers. None of the available cryptographic functions, however, can be implemented on an RFID tag. The smallest published implementation of the AES block cipher, for example, requires almost 4,000 gate equivalents [24] and other cryptographic primitives including cryptographic hash functions and asymmetric cryptography are even larger. While these primitives are already used on contactless smart-cards with a reading range of only a few centimeters (i.e., NXP DESfire), the power constraints of logistics RFID tags, which need to be read from meters away, require primitives that are at least one order of magnitude smaller. To find smaller functions, we revisit the properties needed for privacy and argue that cryptographic hash functions are neither necessary nor sufficient. The properties needed for privacy are captured in the notion of a *private hash function*.

Hash functions in security applications are cryptographic hashes which are defined by three main properties:

(1) recovering the input from the output must be computationally hard (*one-wayness* or *pre-image resistance*),

(2) finding any input that hashes to a given output must be hard (*second pre-image resistance*),
and

(3) finding any two inputs that hash to the same output must be hard (*collision resistance*).

Even functions that satisfy all three properties required of cryptographic hash functions do not necessarily provide privacy. As a degenerate example, given any cryptographic hash, $H(\cdot)$, the function $H'(x) = H(x) || ID$ (where ID is the identifier of the sender) also satisfies the three properties of a cryptographic hash. But because it includes the sender's ID in plaintext, the hash

provides absolutely no privacy. This is obviously a contrived example, but hash functions intended for use in privacy protocols have suffered from similar but less obvious flaws, as shown in the next section.

Smaller and faster hash functions are needed for privacy, but the design of new hashes has been shown to be difficult and many widely used functions have been broken, including MD5 and SHA-0 [62] [63]. The attacks on these functions, however, only compromise collision resistance, which is not necessary for many applications such as message authentication codes (MACs) [5]. Collision resistance is also not needed for privacy protocols, and neither is second pre-image resistance. Of the three properties of cryptographic hashes, only pre-image resistance is important in privacy applications. Therefore, standard cryptographic hash functions are too expensive and potentially insufficient for privacy protection since they were originally designed to server different functions.

4.2.1 Privacy Attack on Simple CRC Hash

Duc et al. propose using CRC as the hash function for the simple hashing protocol [19]. We show that this approach does not provide privacy. In their protocol, a queried tag replies with:

Tag \rightarrow Reader :

$$\langle R, A = CRC(ID||R) \oplus K, B = CRC(A \oplus R) \rangle$$

where ID is the secret identifier of the tag, R is a random nonce, A is the authenticator, B is a checksum, and K is a secret key that only changes when the tag is interacting with a legitimate reader. An attacker who wants to trace a tag can assume that K does not change over an extended period of time.

The attacker wants to compromise the privacy property by finding out whether two responses were generated by the same tag. Let X_g be the set of values that are divisible by g . For the authenticator in the tag response, the following holds:

$$\exists x \in X_g : A = CRC_g(ID||R) = (ID||R) \oplus x$$

Assuming that the two responses, $\langle A_1, R_1 \rangle$ and $\langle A_2, R_2 \rangle$, originated from the same tag, we know that $ID_1 = ID_2$. By XORing A_1 and A_2 we know that:

$$\exists x_1, x_2 \in X_g : A_1 \oplus A_2 = (ID_1 || R_1) \oplus (ID_2 || R_2) \oplus (x_1 \oplus x_2)$$

If $ID_1 = ID_2$, it holds that:

$$A_1 \oplus A_2 \oplus (00\dots00 || R_1) \oplus (00\dots00 || R_2) = (x_1 \oplus x_2)$$

By definition, the sum of two numbers divisible by the generator is also divisible by the generator, hence $CRC_g(x_1 \oplus x_2) = 0$. It follows that:

$$CRC_g(A_1 \oplus A_2 \oplus (00\dots00 || R_1) \oplus (00\dots00 || R_2)) = 0$$

This means that by computing the CRC over the XOR of known values, the attacker can test whether the IDs are equal. This disrupts privacy without breaking any of the strong cryptographic hash properties.

This attack is an example of an attack that compromises privacy, but not security. Other attacks on hash functions compromise properties needed for security applications (e.g., collisions break digital certificates), but do not affect privacy. As argued before, privacy protection needs its own set of requirements.

4.2.2 Definition

We define a set of properties that a hash function must provide to be sufficient for protecting privacy. We call a function that satisfies these properties a *private hash function*. A private hash function takes two inputs, a random nonce, r , and a secret key, k , selected from a set of possible keys, $k \in S$, and generates an output: $h \leftarrow H(k, r)$. A private hash function has the following two properties:

(1) Correctness property: given S and a $\langle r, h \rangle$ tuple, it is easy to decide which one of the secrets, k , was used to generate the output from the nonce. There must be at least one algorithm, $L(\cdot)$,

which runs in linear time $O(|S|)$, and a constant, c , so that:

$$\exists L : k \in S, r \leftarrow U_n, h \leftarrow H(k, r) : \Pr[L(h, r, S) = k] \geq 1 - \frac{c}{2^{|h|}}.$$

(2) Privacy property: given several $\langle r, h \rangle$ tuples, but no knowledge of S , it is hard to decide whether any two outputs have been derived from the same k . Over all polynomial time adversaries, $A(\cdot)$, there must exist a constant c so that:

$$\forall A : k \leftarrow U_{|k|}, r \leftarrow U_n, h \leftarrow H(k, r) : \Pr[A(h, r) = k] \leq \frac{c}{2^{|k|}}.$$

The correctness property states that legitimate readers are able to uniquely identify tags. For functions with even output distributions, this property is fulfilled when the output space is much larger than S . The privacy property guarantees that responses are unlinkable.

4.2.3 Reduction to Random Function

While there may be cryptographic hash functions that satisfy these requirements, our goal is to find private hash functions that are substantially cheaper than cryptographic hashes. We will not directly show that our proposed construct has the privacy property, but instead show that it is indistinguishable from a random function [35]. The following theorem shows that indistinguishability from random is a stronger property than the privacy property and therefore sufficient in showing that a construct provides privacy.

Theorem 4.1. *Every pseudorandom function family is a private hash function.*

A *pseudorandom function family* is a keyed function that, for a randomly chosen key, cannot be distinguished from a true randomness source by any efficient algorithm.

Proof. In order for a function to be a private hash function, it must have the privacy and the correctness property, which we prove consecutively. Any function with an output distribution distinguishable from a uniform distribution is distinguishable from a random function. Every pseudorandom function family (PRF), and therefore has an output distribution indistinguishable from a uniform

distribution. Hence, every PRF fulfills the correctness property, because the chance of an ambiguous identification is $\frac{|S|-1}{2^{|h|}}$.

Next, we prove that every PRF fulfills the privacy property. Assume towards contradiction that there exists a PRF that does not provide the privacy property of private hash functions. Then, there exists a polynomial time algorithm that, given two instantiations of the PRF with different keys, provides a non-negligible advantage in the distinguishability game. This algorithm distinguishes tuples containing two outputs from one PRF from tuples that contain one output from each PRF. When replacing one of the PRFs with a truly random function, the algorithm can be used to distinguish the PRF from random with non-negligible probability, which is a contradiction. ■

Authentication. Note that a private hash function does not necessarily offer authentication, which would require an additional property:

(3) Given many outputs generated using the same secret, but no knowledge of the secret, it is hard to generate a correct response for a new challenge (unforgeability property).

Since this dissertation does not consider authentication, we concentrate on privacy and do not address the unforgeability property.

We present our design of a private hash function, CRC-MAC, in Section 5.3, but will first introduce the design methodology that was used to find CRC-MAC.

Chapter 5

Measuring Cryptographic Strength¹

The design of cryptographic functions remains largely a trial-and-error process in which known statistical attacks are tested against candidate functions until constructs are found that withstand all attacks. The AES competition, for example, found the current standard in block ciphers by soliciting constructs that are hard to break, but did not require the design methodology of the constructs to be reproducible or even comprehensible. While the functions that come out of cipher competitions are usually very strong, little insight is gained into how the different components of a function interact to provide its strength. Furthermore, the evaluation of new cryptographic functions is often limited to the analysis of its different parts, all of which have to be strong for a cipher to be accepted. This design approach leads to constructs that are unnecessarily complex and expensive.

We argue that cheaper designs can be found by testing candidate functions as a whole and allowing for weak building blocks in these functions as long as the interaction between the building blocks protects the weaknesses from being exploited. By design, any selected cipher is resistant to all attacks considered in the design methodology as long as the right heuristics are chosen in making the design tools practical. Anchoring the security claims in the design methodology and not in any concrete function has the additional advantage that cryptanalysis is significantly faster and therefore has increased value. Cryptanalysts can concentrate on analyzing the simple tools used to create a cipher instead of having to cut through the (often unnecessary) layers of complexity inherent in many primitives. Should new statistical attacks be discovered through cryptanalysis,

¹Research for this chapter was done in collaboration with Sean O'Neil.

useful feedback is gained for improving the design tools' heuristics to consider the new attacks as well. Our approach, therefore, provides a positive feedback loop that converts new cryptanalysis techniques into improved tools to create ciphers that are resistant to the new attacks.

Our approach is focused on testing polynomial structures of iterative constructs for local non-randomness as proposed by O'Neil [49]. The polynomial structure of a function encodes its complexity, and functions with sufficiently high complexity cannot be distinguished from random functions (which means they satisfy the properties needed for private hash functions). To test a candidate function for its complexity, we partly compute its polynomial structure and run standard randomness tests on the structure to determine whether it seems to be distinguishable from a random function. Given a set of (independent) round functions indistinguishable from random, the Luby-Rackoff theorem provides us with a straightforward conversion into a function resistant to all statistical attacks. The theorem states that consecutively applying four independent functions that are indistinguishable from a random function creates a cryptographic function resistant to all adaptive and non-adaptive statistical attacks. Therefore, finding cryptographic functions immune to statistical attacks is reducible to finding functions indistinguishable from random. Furthermore, ciphers with a sufficiently random structure are also highly resistant to algebraic attacks since these attacks rely on the sparseness of a cipher's polynomial structure.

We demonstrate the effectiveness of our design methodology by identifying a promising hash function construct from a large design space predominantly populated with weak design choices in Section 5.3. Our design space is defined by the CRC function that is already found on low-cost RFID tags. Through automated testing of a large number of ways to include this function (without adding much hardware overhead), we identify a few constructs that could potentially be strong and a single construct that withstands all of our tests and is reasonably efficient. The hash function that we propose, CRC-MAC, is executed in the hardware of an adapted CRC function. Hence, CRC-MAC has a very small implementation overhead in devices that already provide this functionality. Even small changes to the construct can have a large impact on its polynomial complexity and hence its cryptographic strength. Therefore, the traditional manual design approaches limited by strict criteria would likely have failed to find the proposed function or recognize it as superior to

other possible constructions in the design space within a reasonable amount of time.

5.1 Related Work

The techniques proposed in this chapter build on previous research on designing cryptographic primitives and extend or generalize some previous works. Our results also relate to previous research on lightweight cryptography that proposes size-optimized primitives.

Cipher Design: Previous works on cipher design focus on building blocks, abstract constructs, or the output distribution of ciphers.

Among the best-studied building blocks are S-boxes for Feistel networks. An extensive mathematical framework exists to design S-boxes with high resistance to differential and linear cryptanalysis [51]. The analysis of S-boxes has been employed in the design of several ciphers from DES to AES and the new lightweight block cipher PRESENT [10]. This approach is detached from hardware implementation and is quite inflexible in the size and structure of the overall cipher. Furthermore, the analysis of S-boxes covers only a subset of possible attacks and, therefore, still requires extensive cryptanalysis to show resistance to other attacks such as algebraic attacks.

A second complementary line of work analyzes abstract designs for cryptographic primitives. The most prominent such constructs are Feistel networks [23] and Unbalanced Feistel networks [55] for block ciphers and the Merkle-Damgard construct for hash functions [37]. Our design methodology can be used in conjunction with these results in that we provide a method of finding good round functions to instantiate those constructs.

A last approach to cipher design is the statistical analysis of cipher outputs, which has been used in the evaluation of the AES candidates [58]. The challenge in testing ciphers based on their output distribution is to choose appropriate inputs to model all known attacks. Our tests that analyze the algebraic structure of ciphers encompass randomness tests of the output distribution since a sufficiently randomized algebraic structure always leads to outputs that are indistinguishable from random (Proposition 1 in [26]). For our tests, however, no restrictive assumptions of the inputs are necessary that would exclude resistance to some attacks. We instead rely on the Luby-Rackoff

theorem that guarantees resistance against all statistical attacks as long as the round function is indistinguishable from random [35].

The design methodology discussed in this chapter has previously been employed by O’Neil in the design of EnRUPT, a multi-purpose cryptographic primitive that can be used as block or stream cipher or hash function [48]. EnRUPT provides two main advantages over the alternative primitives such as AES or SHA. First, its design methodology based on the analysis of monomial distributions is comprehensible. Given the same design space and the same randomness tests, other researchers can reproduce the results that show the strength of the primitive. Secondly, the design space for EnRUPT was specifically chosen to contain only combinations of a few simple operations such as addition and rotation. The resulting design is hence much easier to understand than any of the known alternatives, which simplifies implementation and cryptanalysis. While EnRUPT provides a competitive alternative to the existing primitives, we show in Section 5.3 that the same semi-automated design methodology can find cryptographic primitives for previously unexplored application domains.

Lightweight Cryptography: To match the requirements of small devices better, minimum size implementations for some standardized cryptographic primitives have been proposed. Public key cryptography, in particular, has very desirable properties for privacy and authentication applications; hence, the possibility for its integration in small devices has attracted considerable research. The smallest implementations of 131-bit elliptic curve cryptography (ECC) come at 8,600 GEs (not including RAM) [3]. The circuits achieve their small sizes through sequential use of the same hardware for different functions — the highly serial computation of a single ECC operation requires about a second at the typical 106 kHz of RFID tags. Public key cryptography is hence 1-2 orders of magnitude too large and one order of magnitude too slow for the smallest of devices. A much faster and somewhat smaller alternative to asymmetric ciphers are symmetric block ciphers such as AES. The smallest implementation of the AES requires 3,800 GE and encrypts a 128-bit block in 1 millisecond [24]. While modern block ciphers are fast enough for embedded applications, their implementation is still an order of magnitude too large for the smallest devices. Protection measures that exceed 500 GE significantly increase the cost and power consumption of an RFID tag, because

a complete tag without privacy protection can be implemented in roughly 2,000 GE.

Besides ciphers designed for general use such as the AES, some cryptographic primitives have been specifically designed for embedded devices. These ciphers typically have received some amount of public scrutiny and provide a decent level of confidence in their security. The TEA block cipher, for instance, has been updated to the XTEA and XXTEA ciphers to address some weaknesses. XTEA appears to provide a sufficient level of security as the cipher has withstood cryptanalysis for several years [10]. The TEA ciphers are source-heavy unbalanced Feistel networks like CRC-MAC. Randomness tests of the monomial distribution of XTEA show that the cipher's polynomial structure is indistinguishable from random after 19 of the suggested 64 rounds [49]. TEA and CRC-MAC combine shifts, XOR, and addition operations to build a non-linear substitution-permutation network. Unlike CRC-MAC, however, the TEA ciphers require more dedicated hardware and have no redundancy with the CRC function. A minimum size implementation of XTEA was estimated to require at least 2,000 GE [10] and is hence too expensive for many applications. According to our tests, CRC-MAC achieves the same security in the same number of rounds at a significantly lower hardware cost.

5.2 Randomness of Monomial Distributions

We are adapting the design methodology by O'Neil that uses automated tests to determine whether a function is distinguishable from a random function by analyzing its polynomial structure in Algebraic Normal Form [49].

5.2.1 Preliminaries

A function can be defined by the polynomials that express output bits in terms of input bits. The i th bit of a string x is denoted x_i . We consider functions that take one input string, x (typically consisting of a secret key and a data value), and produce an output string: $y \leftarrow f(x)$. Each bit of the output string, y_i , can be expressed as a binary function of input bits. Each term, known as a *monomial*, in the *algebraic normal form* (ANF) of these binary functions is the conjunction of one

or more input bits. Each ANF of a function with n input bits (key bits plus data bits) has the general form:

$$\begin{aligned}
 y_i = & a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + \dots + a_{1,n} \cdot x_n \\
 & + a_{2,1} \cdot x_1 \cdot x_2 + \dots + a_{2,(n-1)n} \cdot x_{n-1} \cdot x_n \\
 & + \dots + a_{n,1} \cdot x_1 \cdot x_2 \cdot \dots \cdot x_n
 \end{aligned}$$

where the j th monomial of degree i has a coefficient $a_{i,j} \in \{0, 1\}$ and the terms correspond to the powerset of the input bits. The string formed by these coefficients, $a_{1,1}||a_{1,2}||\dots||a_{n,1}$ completely describes the function. We show in the next section that if a function's ANF is sufficiently random, the function is indistinguishable from a random function and hence cryptographically strong.

5.2.2 Random Functions

A *pseudorandom function family* (PRF) is a family of functions for which the output of a randomly chosen member of the family is indistinguishable from random for any efficient (i.e., polynomial time) distinguisher. The randomness of the output is a weaker property than the randomness of the ANF of a function; a random ANF always leads to an output that is indistinguishable from random (Proposition 1 in [26]), while a random-looking output can also be produced by a complex, but not random ANF.

A *locally random function family* (LRF) is a function family that is indistinguishable from a PRF up to a certain number of distinct inputs, k . While we would want to use real PRFs, they are too expensive. But since the attackers are usually limited in the amount of ciphertext they can obtain, LRFs with large k suffice for many applications. Formal definitions for PRF and LRF can be found in [35].

Given a PRF or LRF (or a function that is indistinguishable from them), we apply the Luby-Rackoff construct to build a more globally pseudorandom permutation that is resistant to both adaptive and non-adaptive cryptanalysis [35]. The Luby-Rackoff theorem states that after three iterations of a Feistel network with independent PRFs, the mapping between (random) inputs to the Feistel network and its output cannot be distinguished from a random function by a polynomial

time attacker. If the inputs to the Feistel network are not random, a fourth iteration is required to make the function indistinguishable from random by an adaptive polynomial time attacker with access to the decryption process. If LRFs are used instead of PRFs, the resulting cipher is only secure for the number of inputs for which the LRF is secure. Let n be the number of rounds after which a construct becomes a PRF or LRF. Normally, $2n$ Feistel rounds resist passive distinguishers from random without known plaintext, $3n$ rounds resist non-adaptive statistical attacks by known or chosen plaintext or ciphertext attacks, and $4n$ rounds resist all adaptive attacks (including square, rectangle, boomerang, etc.). Patarin, Naor, and Reingold provided similar proofs for Benes networks and for unbalanced Feistel networks [41] [50]. CRC-MAC is an example of a source-heavy unbalanced Feistel network.

5.2.3 Generating Output Polynomials

We argue that if a random function is indistinguishable from an oracle executing the ANF on the input value, then the function itself must be indistinguishable from random as well. All known statistical and algebraic attacks on symmetric cryptographic primitives are based on structural weaknesses that are reflected in the non-randomness of the output monomials. Differential and linear cryptanalysis, for instance, exploit the strong connection between small groups of input and output bits [6] [36], and algebraic attacks exploit the sparseness of ciphers in their polynomial representation [16], which is detectable as the sparseness of the cipher's ANF.

To test conclusively how well a given design implements a function that is indistinguishable from an LRF we would have to compute its complete ANF and test it for randomness. However, exhaustively computing and analyzing ANFs of arbitrary functions is intractable. Instead, we apply heuristics chosen to provide good coverage of all known cryptographic attacks. These heuristics may provide guaranteed immunity to statistical and algebraic attacks that operate on small sets of inputs and outputs.

As our heuristic, we chose to compute the monomials for each subset of input bits up to a certain small size. For a cipher with n input bits, we generate the polynomial relations between the output bits and all sets of p input bits for all values of p up to an upper bound m . This upper bound

is usually constrained by computational power, since computing the ANF of p input bits of an arbitrary function takes $p \cdot 2^{(p-1)}$ executions of the function by Algorithm 1 in [53]. The algorithm takes the truth table of a function (or an oracle that generates it) and computes the monomials starting from the smallest ones and iteratively xors these smaller monomials into the truth table as more inputs are set to **1**. All input bits that are not part of the p inputs under consideration are set to **0** so that the polynomials can have at most 2^p monomials.

Values up to $m = 40$ can be computed on commodity hardware for simple ciphers within minutes. During our tests, we find that all detectable flaws are reflected in subsets of lower degrees (between 15 and 19 bits), while the higher degree tests almost never find additional flaws. The tests that operate on large numbers of inputs often do not detect the flaws that are detected by testing smaller subsets since the noise in the tests is increased as more bits are included that do not contribute to the flaw. Therefore, it is important that smaller sets of inputs are tested first.

Our observation that most weaknesses are caused by small sets of inputs is consistent with results from linear and differential cryptanalysis, which always lead to attacks involving only small sets of inputs and outputs. Differential cryptanalysis is a class of attacks where a known difference in the input bits produces a known difference in the output bits with an exploitably high probability. Linear cryptanalysis is a related attack that finds the best affine approximations to the Boolean functions iterated for many rounds and exploits the higher probability of approximation to find some of the key bits faster than by brute-force. Linear and differential analysis, therefore, exploit a strong connection between small sets of input and output bits.

Algebraic attacks are another example of attacks that exploit only local non-randomness since they rely on the sparsity of the output monomials. Low sparsity has led to successful attacks on a number of stream ciphers [16], but is detected by our tests automatically. We therefore find that computing output polynomials only for low degree subsets is sufficient for detecting cryptographic weaknesses that are exploitable by known techniques.

5.2.4 Randomness Tests

Counting monomials in the ANF of ciphers to detect statistical weaknesses was proposed by Filiol [26]. His *d*-monomial test computes the number of monomials present in the polynomials of a given function and requires that roughly half of all possible monomials are present. The same test was applied by Saarinen to the candidates of the eSTREAM competition for stream ciphers [53]. Many of the analyzed ciphers were found to have monomial structures that are too sparse and could consequently be vulnerable to statistical and algebraic attacks. While any overly sparse (or overly dense) polynomial is distinguishable from random, the inverse does not hold true. We propose to test polynomial structures for properties beyond the average monomial frequency by using standard randomness tests.

Our analysis is constrained by the general infeasibility of conclusive randomness tests. None of the strings we generate are truly random (since we know a way of generating them), but they can still be indistinguishable from random in polynomial time. We define a sufficient level of local randomness in a practical way: any level of non-randomness that the attacker could possibly exploit can also be tested for with the right randomness tests. It should, therefore, be possible to prevent all statistical attacks through sufficient randomness testing of the algebraic structure of all the relationships between all the input and output bits. As shown by the Luby-Rackoff theorem, once a function is indistinguishable from random after a certain number of rounds, it becomes resistant to known ways of cryptanalysis after four times that number of rounds.

The choice of ordering of the monomials matters very little for the randomness tests. The amount of entropy is not altered by fixed transpositions, thus randomness tests distinguish differently ordered ANF sequences equally well. Simple accumulation of all the monomials into a single stream in the order they are generated (by Algorithm 1 in [53], for instance) is sufficient to detect local non-randomness.

Our tests rely on well-known randomness tests such as the NIST and diehard test suites. For example, the diehard bitstream test takes a string of 2^n binary symbols and counts how often different n -bit words are found as subsets of the string. In a random 2^n -bit long string, the number of missing n -bit words should be normally distributed with mean $\frac{2^n}{e}$. The test is performed on a

number of different 2^n -bit strings from the source under the test and is passed if all of them have a distribution reasonably close to the random distribution according to a Z-test.

By applying these standard tests to monomial distributions, weaknesses in the polynomial structure of several cryptographic primitives have been found. Randomness tests from the diehard suite successfully detect the vulnerability of the TEA cipher to related-key attacks for any number of rounds, detect large classes of weak keys in IDEA, as well as distinguishing up to 4 rounds of AES and up to 27 rounds of SHA-1 from random [49].

5.2.5 Design Methodology

Randomness tests of monomial distributions can be used to find weaknesses in the existing functions, but can also find strong new functions. The search for new functions can mostly be automated and our approach is able to find strong primitives even from design spaces with mostly weak options.

We assume iterated constructs, such as Feistel networks, as the subjects for our tests. We execute the tests in two phases. In the first phase, a large number of constructs are tested for easily detectable weaknesses, while the second phase tests the constructs that pass the first round more extensively. For each candidate construction from a given design space, we test the distributions of outputs and all the low degree monomials for randomness up to a limit set to a small number of bits (between $m = 13$ and $m = 27$ bits). If a function fails the randomness tests, we increase the number of rounds the function is run for and test again. If the round function continuously fails to build a set of polynomials indistinguishable from random via these fast randomness tests for an unacceptably large number of rounds, we discard that construction. If it passes the tests for some acceptable number of rounds, we continue to the second phase, where we test higher-degree monomials as much as our computational resources allow (usually up to $m = 40$ bits). These larger and more complex tests require larger amounts of data but are generally less effective than tests on smaller subsets, since most weaknesses are caused by small input sets. The results of the first phase are therefore never far from the maximal number of rounds that the most complex tests are able to distinguish from random. Randomness tests with input sizes around 19-20 bits are usually the most

effective. Of those constructs that pass all our tests, we chose the one that does so for the smallest number of rounds as the best design from the tested design space.

5.2.6 Custom Ciphers

The design methodology was used to create EnRUPT, a general-purpose primitive proposed to be used in place of more expensive functions including the AES and SHA. The same design methodology can be used even more efficiently by creating custom-tailored cryptographic functions for specific purposes. Any function found through automated randomness tests of its polynomial structure is by design immune to all statistical attacks that were considered in the heuristics of the test set. Once the right set of randomness tests (and their parameterization) is found, creating new strong functions merely consists of applying the automated tests to sieve through the given design space. Each design space can encode the specific constraints of one application. All the identified primitives provide satisfactory cryptographic strength, which is determined through the randomness tests, but the different functions will be optimized for different purposes. In the next section we illustrate how a keyed hash function was found that fits into the tight constraints of small devices such as RFID tags by reusing some of their circuitry.

5.3 CRC-MAC

We demonstrate the effectiveness of automated monomial tests by identifying the best design choice for a new one-way hash function from a large design space of mostly very weak choices. We deliberately constrain our design space in an unusual way by only allowing constructions that share circuitry with a function already found on some of the smallest devices. This makes the search particularly challenging as we are targeting a domain for which no cryptographic primitives exist yet.

We are proposing CRC-MAC as a keyed hash function that is smaller than any standard cryptographic hash by reusing other functionality already implemented on most communicating devices. CRC-MAC is designed as a private hash function and therefore not meant to provide collision-

resistance, which would be required for electronic signatures but is not needed for challenge-response protocols [5].

5.3.1 Design Space

The design choices for our one-way hash function are limited in that the design has to fit on even the smallest of devices. Our design, therefore, integrates the CRC function that is already found in a wide range of devices. The CRC function computes the square of a polynomial modulo another polynomial (i.e., the remainder of a division). Since the CRC function is highly linear, all attempts to employ it for cryptographic purposes have failed so far (e.g., [19]). Despite its weaknesses, we found a novel way to build strong substitution boxes (S-boxes) out of the CRC function by chaining multiple round of the CRC S-box.

Our design space is defined as any simple CRC-based function in a simple network structure. We assume two or three consecutive data words, one key word, and a round counter, as potential inputs to the CRC function. We had to limit the number of concurrently consumed data words to keep the number of data ports to the RAM low. More than two data ports significantly complicate RAM design and drive up its power and area costs. Note that even when three data words are used, only two RAM data ports are needed, since the last round's data word is still contained in the CRC circuit and can be read from there.

The different designs we consider vary in the way data words, key word, and round counter are combined to form the two inputs to the CRC function—generator polynomial and data input—and in the number of clocks the CRC function is run. Each of the two CRC inputs is formed by the XOR or XNOR of several inputs or parts of inputs. We only considered using data words in full, half, or without their first or last bit. Finally, we considered adding a constant to the CRC inputs such as FF00, F0F0, AAAA, among a few others. The combination of these choices spans a design space with about 10 million possible designs (for CRC-16).

The input to the automated test tool is a description of all these design possibilities as C source code. All design choices are encoded as switch-case (or if-else) statements in the code and the automated design tool tests each combination of choices in turn. The output of each test is the

number of rounds after which the construct's ANF becomes indistinguishable from random by our tests (or an indication that the construct appears to stay distinguishable even for large number of rounds). Testing time for the different constructions varies because each test ends once the first significant weakness is found. All the design choices were tested within a few days on a PC. Extended tests with higher m did not yield different results.

To find the best constructions within this design space, we employed the automated tests described in Section 5.2 to identify constructions that become indistinguishable from an LRF. In a first pass, we sort out constructions whose output polynomials do not grow consistently in size and randomness with the number of Feistel rounds. Our first run pruned the design space in several ways. First, we found that only CRC functions with the highest bit of the generator polynomial set to **1** achieve consistent polynomial growth. From our design space, these are only those choices where the generator polynomial is computed as the XNOR of input words without their first bit.

Second, it became apparent that we would need to use an unbalanced Feistel network (i.e., a Feistel network with more than two data words) to achieve sufficient mixing when using small CRC functions. This became clear because almost all parameterization that passed the test used three data words, and those that passed using only two data words required a significantly larger number of rounds to achieve sufficient mixing. If, however, the input data consists of only two words, then mixing between only these two words in every round is sufficient according to our tests. Third, we found that none of the constants we tried changed the cryptographic strength of the constructs according to our tests. Therefore, we leave out constants for simplicity.

5.3.2 CRC-based S-box design

The construction our tests find as the best choice is given in Figure 5.1(b). Figure 5.1(a) presents the same construct for the case where only two input data words are used. We verified that this simplified version provides the same cryptographic strength as the version with more input data words according to our tests. The choice between the two will hence only depend on the required input entropy.

The CRC-based S-box in this construction remains largely linear and therefore cannot be used

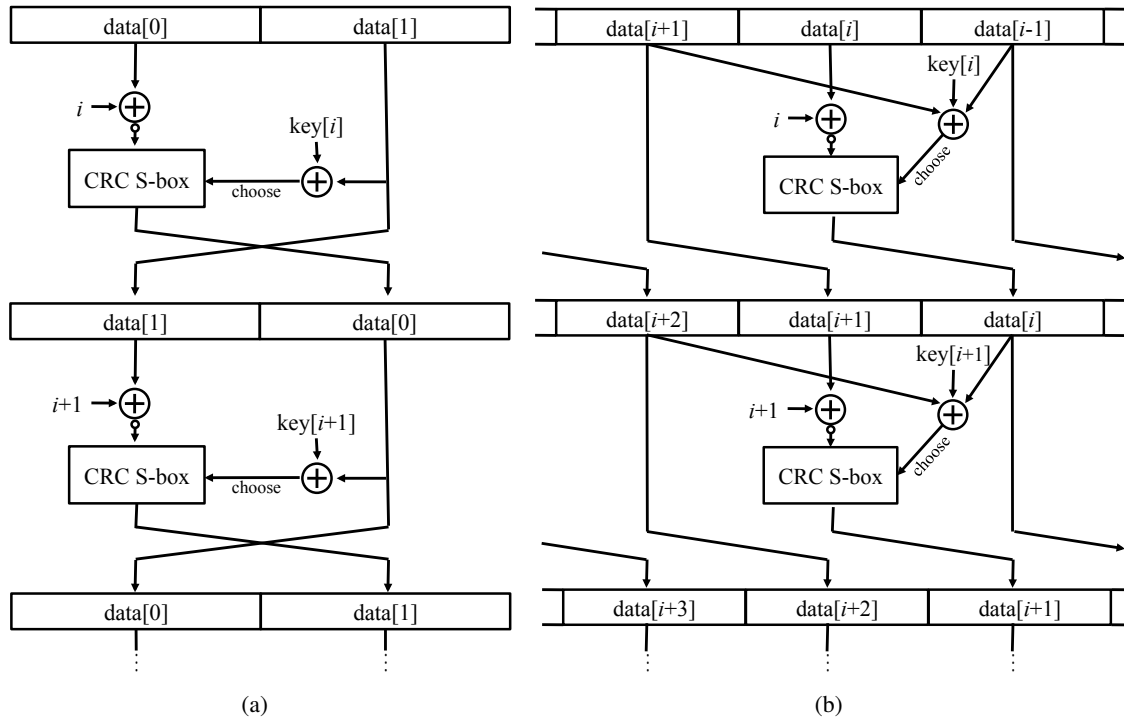


Figure 5.1: Feistel Network operation of CRC-MAC as (a) Feistel network with two input data words or (b) unbalanced Feistel network with more data words.

in the conventional Feistel network ciphers. Our tests have already eliminated constructions in which this would lead to an exploitable weakness. In particular, all the designs that were found as strong, do not combine the neighboring words with the current' word that is substituted by the S-box (as explained in Section 5.3.3). Instead each data word goes through a chain of different S-boxes unaltered by the bits that may be controlled by the attacker. We have verified that after 4 to 6 iterations, every possible S-box chain for CRC-8 builds up to an S-box with nonlinearity and correlation immunity equal to an average 8×8 permutation without any weak cases, roughly comparable with the pseudorandom permutations produced by two iterations of the AES S-box. Hence, even the S-boxes built from an 8-bit CRC provide high resistance against linear and differential cryptanalysis after a small number of Feistel rounds. Resistance of the larger CRC S-boxes is even higher as they are wider and more non-linear. Note that while this analysis is crucial for understanding why the constructions we find are strong by traditional metrics, it is by no means necessary for designing new primitives—simply generating the ANF and running the right randomness tests is

all that is needed.

The resulting S-box appears weak when analyzed using the traditional S-box criteria such as high nonlinearity, correlation immunity or avalanche criteria. Other design methodologies based on mathematical analysis of S-boxes would, hence, not have found this construction. An unbalanced Feistel network with our CRC-based S-box, however, is indistinguishable from a random function according to our tests. This discrepancy between our results and those of other design tools can partly be explained by the focus of the approaches: our tests look at the complete function after a number of rounds while other design methodologies concentrate on the individual building blocks.

5.3.3 CRC-MAC Structure

The construction our tests identify as the best choice from the design space is an unbalanced Feistel network in which one data word is substituted in each round as depicted in Figure 5.1. The network takes as inputs n_{data} data words and n_{key} key words. The word size, w , is determined by the size of the CRC circuit. Our initial tests considered only 16-bit CRC, but we verified that 8-bit and 32-bit CRC lead to equally strong constructs, yet after different numbers of rounds. When given the choice, larger CRC function are preferable since the same input size is divided into fewer words and hence fewer rounds are needed to make the construct secure.

Adding neighboring data words into the S-box input does not lead to stronger constructions according to the randomness tests. The data word that is substituted is, hence, linearly combined only with the round counter, but not with bits of the neighbors or key. The current data word's two neighbors influence the choice of substitution function, thereby achieving good mixing between the different data words. The S-box is determined by the CRC generator polynomial, which is best composed as the linear combination of the two neighbors, the current key word, and some bits of the current data word.

Integrating some bits of the data word leads to the necessary non-linearity in the S-box. We face a tradeoff here because for every z bits of the data word that we integrate into the generator, the CRC state can be shifted for no more than $w - z$ bits during the CRC calculation to preserve its bijectivity. The most efficient choice is to integrate half of the word into the CRC generator

polynomial shifting it by as many bits. Iterating the CRC function only for half of its size results in the higher half of the state word being added linearly to the lower half of the output and the lower half of the state influencing both halves of the output weakly nonlinearly. The roles of the two halves swap between data and key on every round as in a balanced Feistel Network. Iterating the CRC S-box for $w/2$ clock cycles while adding $w/2$ bits of the current word to the generator leads to very simple, but surprisingly strong, bijective S-boxes (as long as the highest bit of the generator is set to **1**).

Note again, that this analysis was only carried out after the tests identified the construction as the strongest available in the design space. Hence, the analysis is not a necessary part of the design process—if a strong construction is included in the design space, then the randomness tests will find it.

With all these conditions satisfied, any bias in the distribution of monomials of the function reduces continuously with the number of rounds to a strongly pseudorandom function. In this design, our multi-round CRC S-box builds a secure keyed one-way hash with a homogenous source-heavy unbalanced Feistel network. Our tests found the unbalanced Feistel network depicted in Figure 5.1 to be the fastest function to pass all the tests therefore withstanding all the attacks we have considered. According to Luby-Rackoff and Patarin, to be secure against all polynomial time adaptive statistical attacks, our construct should use at least four times the number of rounds indistinguishable from a set of independent PRFs in polynomial time. CRC-MAC passes all our ANF randomness tests after $(3 \cdot n_{data} + n_{key})$ rounds for the word widths that we have tested (8, 16, and 32 bits), which suggests that it should be able to resist all adaptive statistical attacks after $4 \cdot (3 \cdot n_{data} + n_{key})$ rounds.

5.3.4 Key Schedule

Even functions indistinguishable from random are potentially susceptible to attacks when they are not keyed properly. The randomness tests assume that completely independent keys are used for the four iterations, but in practice less key material is available that is expanded into round keys by a key schedule. Slide attacks are an example of attacks that exploit the similarity of different

rounds in iterated constructs where the same key is used in different rounds [7]. Even if the round function builds up to a PRF after a certain number of rounds, we still need all such groups of rounds to be different from one another in a way indistinguishable from random in polynomial time. To be resistant to slide attacks, a function needs a varying algebraic structure in every round. As a simple key schedule that makes the construct resistant to slide attacks, the round counter is added to the S-box on every round. With a sufficiently large number of rounds, this simple key schedule provides resistance to related-key attacks and provides some level of collision-resistance.

5.3.5 CRC-MAC Description

A software implementation of CRC-MAC is given in Figure 5.2 on page 87. This code is an optimized version of the path through the input source code that the randomness found to be the best choice. The function operates on w -bit words as follows:

Inputs:

The number of data words: n_{data}
 The number of key words: n_{key}
 data words: $d_0, \dots, d_{n_{data}-1}$
 data words: $k_0, \dots, k_{n_{key}-1}$

Initialization:

Set the bit mask (which sets the highest key bit to 1): $m \leftarrow 2^{w/2} - 1$
 Copy data into state register: for $i = 0 \dots (n_{data} - 1) : x_i \leftarrow d_i$

Iteration:

for $r = 0 \dots (4 \cdot (3 \cdot n_{data} + n_{key}) - 1)$
 $x_r \leftarrow (x_r \oplus r)$
 if $n_{data} = 2 : g \leftarrow k_r \oplus (x_r \& m) \oplus x_{r+1}$
 if $n_{data} > 2 : g \leftarrow k_r \oplus (x_r \& m) \oplus x_{r+1} \oplus x_{r-1}$
 $x_r \leftarrow CRC(x_r, g, w/2)$

The operator $\&$ is bitwise AND. The function $CRC(d, g, c)$ stands for the CRC with data input d , generator polynomial g , executed for c clock cycles. All indexes into the data/key array are modulo n_{data} / modulo n_{key} (i.e., $x_{(r-1)} \equiv x_{((r-1) \bmod n_{data})}$).

Output:

When CRC-MAC is used as keyed one-way function or MAC: return x
 When CRC-MAC is used as a hash (with publicly known key): return $x \oplus d$

```

// compute generator polynomial and
// execute the 16-bit CRC function on state s for 8 clock cycles
u16 CRC_half(u16 s, u16 k) // s := CRC state, k:= key input
{
    u16 g, j;
    // compute generator from half of data word and key input and
    // set highest bit of generator to '1'
    g = ( s & 0xFF ) ^ ( k | ( 0x8000 ) );
    // shift for 8 clock cycles and
    // XOR state with generator when lowest state bit is '1'
    for( j = 0; j<8; j++)
        s = ( s & 1 ) ? ( s>>1 ) ^ g : ( s>>1 );
    return s;
}

// compute 16-bit CRC-MAC, hash output is left in x[]
CRC-MAC(u16* key, int n_key // key : key[n_key]
        u16* x, int n_data) // state: x[n_data]
{
    u16 r;
    for ( r = 0; r < 4*(3*n_data+n_key); r++)
    {
        x[r%n_data] ^= ~r;
        CRC_half(x[r%n_data],
                key[r%n_key] ^ x[(r+1)%n_data] ^
                (n_data>2 ? x[(r+n_data-1)%n_data] : 0));
    }
}

```

Figure 5.2: C implementation of CRC-MAC as keyed one-way function.

CRC-MAC can be instantiated for different data and key sizes and its hardware implementation can be parallelized for higher performance. Furthermore, CRC-MAC is flexible as to which underlying CRC function is available. Details of a minimum size hardware implementation are presented in Section 5.3.7.

5.3.6 Security analysis

In Section 5.2 we explain how our design methodology leads to constructions that resist all known statistical attacks by building a set of polynomials indistinguishable from a random function after a sufficient number of Feistel rounds. Resistance of the new construction against all known attacks is an important design goal, but does not prove its security as new attacks are typically discovered only after a vulnerable algorithm has been proposed. We can, however, exclude large classes of attacks that by design do not apply to CRC-MAC. Slide attacks, for instance, are prevented by the use of the round index making CRC-MAC a 2^w -round self-similar function.

Chosen plaintext or ciphertext attacks are not possible against authentication or private identification schemes because all the inputs to the hash are either random or secret. Furthermore, as we argued in Section 5.2, all functions that pass our randomness tests will most likely also resist linear, differential and any other statistical cryptanalysis.

CRC-MAC could be vulnerable to sophisticated algebraic attacks in which the attacker constructs the equations that express output bits in terms of (secret) input bits and solves these equations for the secret key. In order for the equations to be solvable with feasible effort, they need to be simplified [16]. For our hash, the generated equations are of high degree, very long, and very dense. According to our tests, they are even indistinguishable from randomly chosen equations after a quarter of the proposed number of rounds. These tests, therefore, give us a high level of confidence that no simplification of the equations can be found that will hold for a large number of keys, which would guarantee protection from algebraic attacks.

Collision-resistance is not required for CRC-MAC or any other private hash function, and since collision-resistance is a new and rapidly evolving area of cryptology, we will not speculate on the collision-resistance of CRC-MAC. We note however, that when CRC-MAC is used with variable

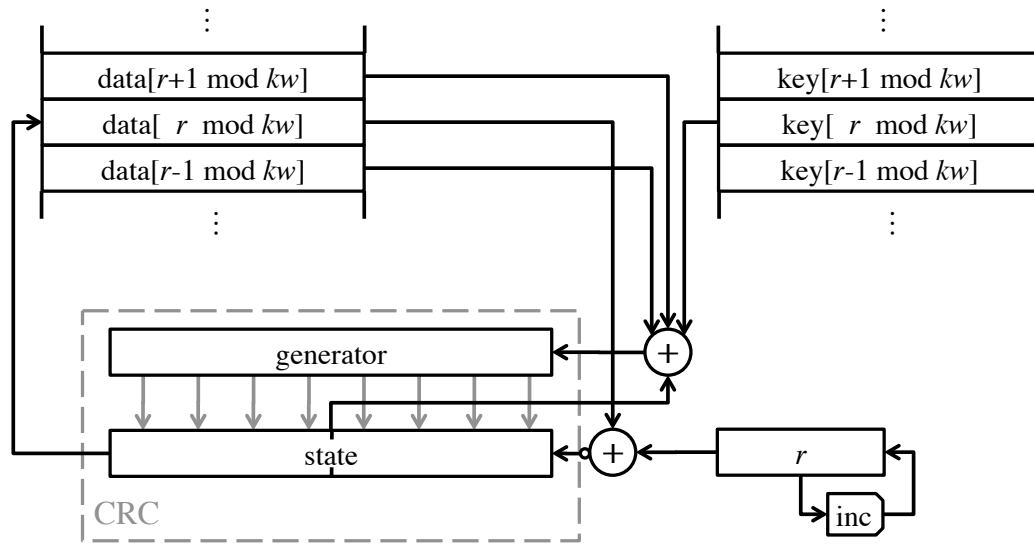


Figure 5.3: Structure of CRC-MAC.

secret key and equal size input and output, collisions are very rare.

5.3.7 Implementation

The design space in which we found CRC-MAC was specifically constrained in that it only allowed for solutions that can be implemented in small devices such as RFID tags with little hardware overhead. In this section we estimate this overhead by analyzing one possible implementation, which was manually derived from the cipher description. The implementation we are proposing is shown in Figure 5.3. The data words are stored in RAM and the key words are stored in RAM or ROM. In every round, one data word is loaded into the CRC data register while a combination of the neighboring data words, a key word, the round counter, and some bits of the data register are XORed and loaded into the generator register. Then, the data register is shifted by $w/2$ and XORed with the generator every time it has a **1** in its highest position. Finally, the data word is written back into RAM, the round counter is incremented, and the next round begins.

The CRC generator polynomial is typically constant when used for error detection. For our implementation, we adapt the CRC circuit to have a variable generator polynomial, which requires additional flip-flops and XORs. The only combinations of logic gates needed to build a variable generator CRC are flip-flops (FF) followed by NANDs and XORs followed by FFs. By building

more specific cells that combine NAND or XOR with a FF, area can be saved since fewer transistors and boundaries are needed and some of the transistors can be made smaller. The FF-NAND can be built in 83% of the area of separate FF and NAND, while the XOR-FF can be built in 81% of the area of XOR and FF. The size estimates for the adapted FFs were generated using Cadence layout tools and the size of the implementation presented next was estimated from synthesized Verilog code. The standard cell library we use comes from a 500m process for RFID tags by NXP [45].

The vanilla 16-bit CRC occupies a total of 101 gate equivalents (GE), while our CRC variable generator polynomial occupies 188 GE (or 87 GE overhead where CRC is already used). The variable generator CRC can be used for normal CRC as well as for CRC-MAC. The cipher further needs a small non-linear feedback shift register (NLFSR) that is used as round counter—a component often neglected when estimating the implementation footprint of ciphers. This round counter requires 7 FFs and 4 other logic gates and occupies a total of 47 GEs. Note that the counter does not produce values as an increment-sequence, but instead in a different sequence of unique values, which can be generated with less hardware. This sequence is at least as effective as an increment sequence in preventing slide attacks and might be stronger in other functions because the different counter values are derived through a non-linear process.

Lastly, four more XOR cells are needed to combine the different values that form the generator polynomial. The total gate footprint the 16-bit CRC-MAC implementation is 262 GEs (or 161 GEs more than a plain CRC). CRC-MAC is hence smaller than the smallest currently proposed cryptographic function for RFIDs by a factor of four [10]. The estimates neither include the hardware needed for key storage (which can be very small ROM) and data storage (which can be RAM shared with functions other than hashing), nor the overhead of routing, placing, and interface logic.

Closely related to the implementation size is the power consumption of a function. Power consumption will arguably become the more important constraint for RFID tags after a few more process generations when more transistors can be built on a tag than can be supplied with power. The power consumption scales with the number of transistors and the frequency with which the transistors switch, called *activity factor*. The activity factor is typically relatively high for area-optimized implementations of cryptography because circuits are reused as much as possible. Our

implementation of CRC-MAC has a similarly high activity factor. Compared to the larger ciphers, CRC-MAC consumes relatively little power since it operates in much fewer transistors. Compared to the smallest implementation of AES, CRC-MAC consumes roughly $\frac{1}{14}^{th}$ the power, since the AES implementation is 14 times its size [24] (assuming similar activity factors and clock frequencies).

Chapter 6

Building Private Identification Systems

The protocols and hash function presented in this dissertation provide a number of tradeoffs between desirable properties of an RFID privacy protection scheme. Most of the tradeoffs are between server cost and privacy. When designing a privacy system, each of the tradeoffs needs to be parameterized. An optimal design method finds a set of parameters that maximizes privacy while staying within some given cost budget. This section presents such a design method that finds the optimal parameterization of the tradeoffs presented in Chapter 3 for a given cost budget.

Our design method for privacy systems considers only choices that can be implemented on RFID tags, but provides a large spread of alternatives in terms of server cost, since the maximally acceptable server cost varies widely among different applications. In logistics applications, for instance, reading time is a critical performance measure and privacy protection must not add considerable work to the reading device. In other applications, such as contact-less payment, a somewhat higher server workload is acceptable, since transactions already involve extensive computation. While we argue that somewhat higher workloads are often reasonable, we still believe that the computational overhead of public key cryptography would be prohibitive for both device and server. Our design space, therefore, only considers symmetric key cryptography and only considers the tree protocol.

6.1 Picking a Hash Function

The first and perhaps most difficult decision in designing a symmetric-key privacy system is choosing the underlying hash function. All previously proposed functions are too expensive to implement on RFID tags, which leaves the two hash functions discussed in this dissertation: adapted HB protocols and our private hash functions, *CRC-MAC*.

Both functions have similar performance by most measurable metrics: they have similar communication overheads; about the same implementation cost on RFID tags (when a CRC function is already present); and they can be used in the same privacy protocols. HB functions can be implemented more efficiently on the server side in software or dedicated hardware. But since both solutions are relatively small compared to key storage and key lookup, the difference in implementation size does not have a large effect on the overall server cost. When using HB functions, the level of added noise has to exceed some threshold, while private hash functions operate independent of the level of noise.

Both LPN-based functions and automated cipher design, which lead to *CRC-MAC*, are too new to make conclusive claims about their security. Automated cipher design, however, comprises defenses against statistical attacks, which have been thoroughly investigated over the last decades. Unless new attack ideas are found, *CRC-MAC* resists cryptanalysis as argued in Chapter 5. The security of HB protocols and LPN in general, on the other hand, is an area of active research. Recent improvements in LPN solving (e.g., [28]) suggest that the solving of LPN problems might be much less hard on average than previously believed [28]. We therefore suggest using *CRC-MAC* until more certainty exists about the eventual security of HB protocols. The following discussion on protocol parameters and key sizes apply equally to private hash functions, HB functions, and any other secure one-way function that can be implemented on RFID tags.

For the design method described in this chapter, we assume that HB functions and *CRC-MAC* (with or without randomization) can be implemented on RFID tags. Therefore, we concentrate on maximizing privacy while not exceeding a given number of hashing operations on the server side.

6.2 Tweaking Protocol Parameters

This dissertation provides four tradeoffs between privacy and cost on the protocol layer: periodically updating secrets, adding noise to user responses, decreasing the tree depth, and increasing the spreading factor on the last tree level. Finding the right balance between these options is crucial to achieving the best possible privacy at a certain cost.

As defined in Section 2.3, the level of privacy grows with the amount of entropy in a system as seen by the attacker. Assume a tree with N tags, output size n , spreading factor k , spreading factor on the last level k_{last} , tree depth d . The entropy of the tree protocol varies with the randomization p :

$$r(p) = \sum_{i=0}^n \frac{\text{Binom}(i, n, p)^2}{\text{Binom}(i, n, \frac{1}{2})}$$

$$\begin{aligned} \text{Privacy}(k, k_{last}, d, p) = (d-1) \cdot & \left(\log \left(\frac{N^{1(d-1)}}{k} \right) + \log(k + r(p) - 1) - \frac{r}{k+r(p)-1} \log(r(p)) \right) \\ & + \log(k_{last} - 1). \end{aligned}$$

The first summand is the entropy of each of the randomized levels, as derived in Equation 3.3, multiplied by the number of levels. The second summand is the entropy of the groups on the last level, which is not randomized and from which only one secret of each branch is known to the attacker. The parameter n is the output size, which does not influence the result very much in the range of reasonable choices.

The average server cost for the simple search strategy described in Section 3.4.4 is:

$$\text{Cost}(k, k_{last}, d, p) = \left(\sum_{i=0}^{np_1} \left(\text{Binom}_{PDF}(i, np_1, p_2) \cdot k \cdot \left(1 - \text{Binom}_{CDF} \left(i, n, \frac{1}{2} \right) \right) \right) \right)^{(d-1)} \cdot k_{last}.$$

The cost is the expected number of branches that need to be evaluated before the right tag is found. This includes the right branch and those branches that seem more likely to be the right branch as a

result of the randomization, which was derived in Equation 3.4.

All design choices experience diminishing returns; earlier steps always provide a better privacy/cost ratio than latter steps. A simple iterative algorithm that compares the current privacy/cost ratio of the next possible choices can, therefore, find the optimal parameterization within the design space described by the cost-privacy tradeoffs. This iterated algorithm is depicted in Figure 6.1 on page 99.

The algorithm takes as parameters the number of users in the system, N , and an upper bound on the average server cost of each identification that must not be exceeded, C_{max} . The output is a set of protocol parameters that maximize privacy within this cost budget. The algorithm starts with the lowest-cost tree, which is a binary tree with $k = 2$, and no randomization.

First, a designer needs to decide whether periodically updating secrets is a viable privacy measure for a system. This measure is orthogonal to the other design choices and impacts the other measures only in that the tree needs to be slightly increased in size to accommodate for the tags that are being updated. The decision about whether to update secrets is therefore made first. As explained in Section 3.3.1, updating secrets provides highly increased privacy for applications in which tags are monitored closely enough to know when one is compromised, such as in certain retail scenarios. If the decision is made in favor of updating, the secret tree needs to be grown to accommodate for tags that are currently being updated. The maximum number of concurrently outstanding updates will vary widely across different systems and calculating its exact value is therefore outside of the scope of this dissertation. For lack of a better estimate, we chose to increase the tree by 10%, since this size provides plenty of room for updates while not having a large impact on privacy at the system sizes we consider. After this setup step, the algorithm iteratively finds the best balance between the remaining three design choices.

In each iteration, the ratio between privacy and cost is computed for the three different tweaking variables; tree depth, spreading factor on the last level, and noise. If the most advantageous of the options does not exceed the cost budget, it is applied and the next iteration begins. Otherwise, the iterated phase ends. Since all three measures can only be applied in discrete steps, the order in which they are compared influences the result. A larger-grained measure might compare unfavorably

to a finer-grained measure when only the next step is considered, but might actually achieve a better result when one increment of the first measure is compared to several increments of the second measure that both achieve the same privacy gain. Therefore, larger-grained measures must be applied with preference to find the optimal choice among the available design choices. The different tweaks provide different granularities: key depth and spreading factor can only change discretely in relatively large steps, while randomization can be increased by a very small amount, Δp , whose lowest possible value is determined by the resolution of the available random numbers. To give preference to the larger-grained tweaks, the finer-grained tweaks are reset each time a less fine-grained tweak is applied. In particular, randomization is reset to zero when one of the other tweaks is applied, and the spreading factor is reset to a balanced tree when the tree depth changes. The resolution of the randomization, Δp , should consequently always be chosen to provide a smaller privacy increase than the discrete choices for the other measures, which is achieved by $\Delta p = 10^{-5}$ for the system sizes we consider.

After the iterative phase cannot find any more improvements, randomization is increased so as to exhaust the available cost budget. This can simply be achieved by increasing the randomization at ever-smaller steps until the resolution of the random number generator on the RFID tag is reached. Finally, the algorithm terminates with the best possible parameters for the three design choices that maximize privacy at a given cost.

6.3 Choosing Key Sizes

All protocol layer measures rely on the one-way properties of hash function, which is only achievable when sufficiently long cryptographic keys are used. Key storage, however, is one of the cost factors on the server and tag that competes for resources with other measures. Keys should, hence, only be chosen to be as strong and expensive as necessary. Finding the key through brute force should be comparable in cost to finding the key by other means. The optimal key length depends on the type of attacker and varies for different layers of the tree and for different hash functions. HB hash functions, for instance, should never be used with keys smaller than 64 bits, since the LPN

problem can be inverted for smaller keys, as explained in Section 4.1.2, using attacks much faster than brute force. When other hash functions such as CRC-MAC are used, brute force should be assumed as the most efficient cryptographic attacks and the key size should hence be chosen so that a brute force attack is not economical.

For the higher-level secrets in the tree that are shared across several tags, keys can be extracted from one tag to compromise the privacy of others. We estimate that extracting the keys from a non-tamper-resistant tag costs about \$2,000 (and less when done for many tags)¹. For the same investment, a 56-bit key can be found through brute force. Note that when randomization is used, the brute-force time increases significantly since many samples have to be hashed with each key to find the key that matches best on average. Hence, 52-56 bit keys are sufficient for the higher levels of a randomized tree.

The optimal key size on the lower level depends on the likely attacker. Profiling attackers and corporate espionage attackers want to extract as many traces as possible. Each secret key that is broken adds one more trace. The cost of breaking the key should therefore scale with the value of the traces. When assuming that RFID traces, like Internet traces, will at most be worth a few dollars on average, key sizes of 40-48 bits should suffice in defeating the profiling attacker. The surveillance attacker, on the other hand, is very determined to track one or few individuals. Keys of 60-64 bits should be used to make breaking even a single key fairly costly.

To prevent over-engineering, it is important to consider the motivation of likely attackers and the cost of different ways to disclose secret keys. Most RFID privacy systems will optimally use two level trees and need less than $2 \cdot 64 = 128$ key bits total per tag. Larger keys usually do not improve the overall security.

¹We estimate that extracting the secret keys from one tag takes about one day (at \$1,000) and requires access to about \$50,000 worth of equipment for a few hours, plus some inexpensive chemicals. When keys are extracted from larger numbers of tags, economies of scale decrease the cost per tag.

6.4 Determining Output and Nonce Length

The optimal output length, similar to the key size, varies for different tree levels. Outputs must be large enough to prevent false identifications. For randomized tree protocols, only ambiguity on the last level can lead to false identifications, while ambiguity on the other levels increases privacy.

The outputs for the last level must be large enough to make the probability of a collision between different tags negligible, which requires that the number of possible outputs on the last level be much larger than the spreading factor on that level $2^{n_{last}} \gg k_{last}$.

The size of the outputs on the randomized levels is optimal when the expected level of collisions is low, but not necessarily negligible. Large numbers of collisions must be avoided since they interfere with the randomization and increase the server workload (collisions also increase privacy, but less than increased randomization would at the same cost). The optimal output size depends on the cost ratio between processing more data and computing slightly more hashes. Without a way to calculate these costs, we suggest setting the output size on the randomized levels to $n = 2^k + 4$, as this setting achieves a low frequency of collisions while not increasing the communication cost significantly at the system sizes we consider.

Nonces should be chosen so that the probability that an attacker sees the same random nonce twice is small. Nonces of length k lead to a repetition after an average of $2^{k/2}$ readings. Assuming $k = 32$ and that tags emit a fresh nonce at most once a minute, an attacker would have to read tags from one group constantly for more than six weeks, and even longer when randomization is used. Hence, nonces of length 32 bits are sufficient in the scenarios we consider.

Parameters:

N // maximal number of tags in the tree
 C_{\max} // server cost budget

$$Adv(k, k_{last}, d, p) = \frac{Privacy(k, k_{last}, d, p)}{Cost(k, k_{last}, d, p)}$$

Initialization:

$k = 2$ // spreading factor
 $k_{last} = 2$ // ... on last tree level
 $d = \sqrt{N}$ // tree depth
 $p = 0$ // randomization level

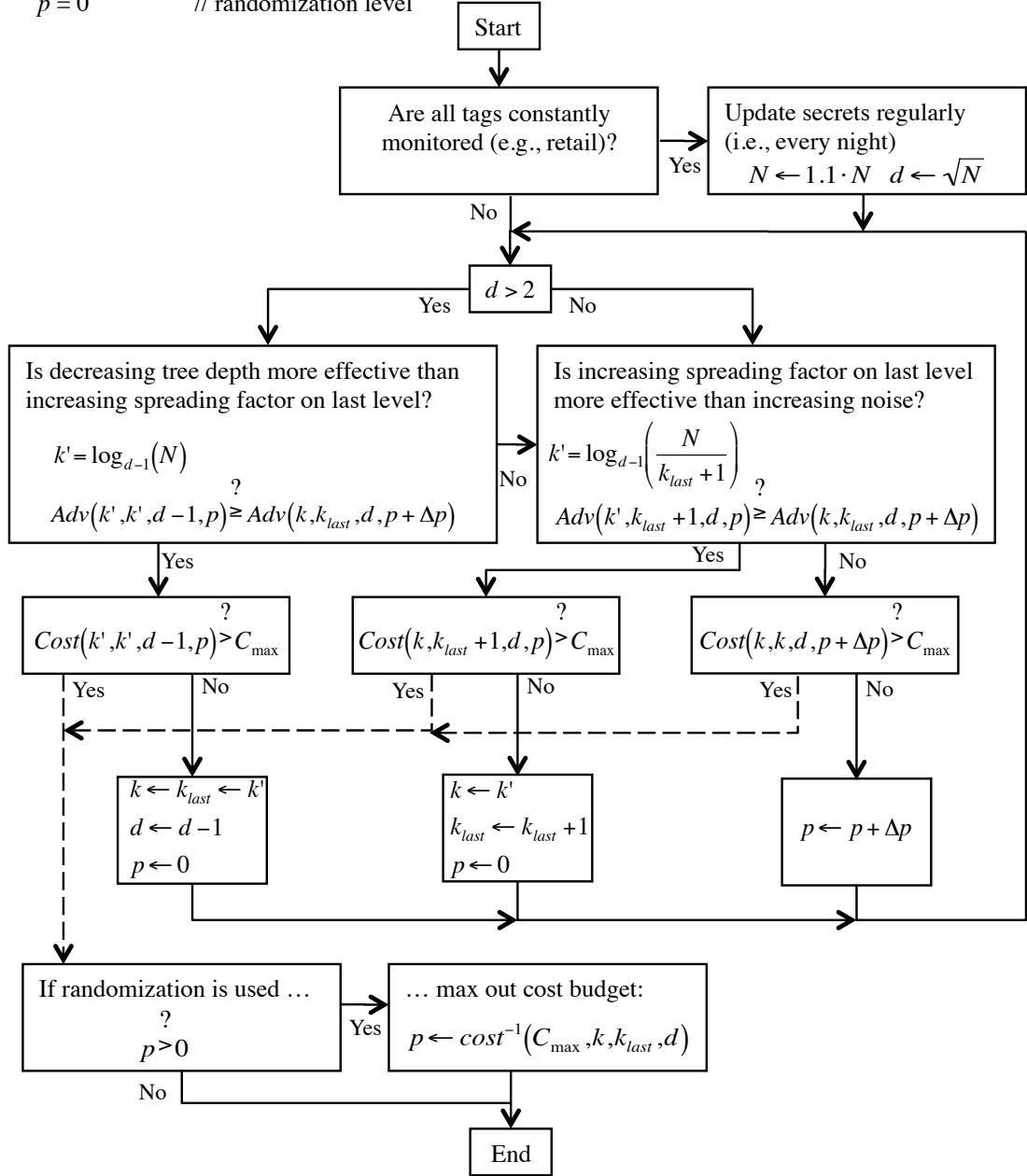


Figure 6.1: Design method for finding the optimal protocol parameters for a given system size and server cost budget.

Chapter 7

Conclusions

Strong privacy protection is possible at low cost. To achieve the best protection against realistic attacks within a limited cost budget, all components of a privacy system must be designed so that the overall system provides the necessary protection, but is not over-designed. To find the necessary level of protection, the incentives of likely attackers must be considered. This dissertation provides the building blocks needed to implement privacy systems at low cost for RFID tags and servers, as well as a framework for evaluating a system's privacy in light of attackers' abilities and incentives.

7.1 Summary of Contributions

This dissertation contributes to progress in understanding how information leakage impacts privacy, designing more economical privacy protection, and implementing privacy protection on RFID tags.

The main contributions of the dissertation include:

1. Introducing information leakage as an intuitive metric that measures privacy in terms of lost entropy. Information leakage can be used to quantify the privacy impact of various information sources. Furthermore, we show that distributions of information leakage capture the different levels of privacy experienced by different users. These privacy distributions are important for estimating the privacy of a system when facing realistic attackers (Section 2.3).
2. Identifying three realistic attackers who aim to extract traces from RFID data. The attackers

are described by two functions: the attacker strategy function that captures how well an attacker can apply data mining, and the attacker value function that encodes the incentives of the attacker. By combining this attacker model with distributions of information leakage, which we call binning functions, we can estimate the expected value of an attack. The three types of attackers differ in their incentives so that certain protection measures are effective against some attackers but not against others (Section 2.4).

3. Describing and analyzing several improvements to probabilistic privacy protocols such as the tree protocol that trade higher server cost for lower attack value. Privacy is gained by restructuring the tree so that no tags fall in small groups and by adding noise to tag responses, which makes different tags less distinguishable. The combination of these two measures provides design choices with very low information leakage at acceptable cost (Sections 3.3 and 3.4).
4. Introducing two options for implementing privacy protection on resource-constrained devices. The first option is to parametrize HB hash functions to forestall easy inversion, have low communication costs and, at the same time, provide satisfactory privacy in the randomized tree protocol. These improvements are achieved by limiting the number of samples an attacker can collect and by deriving all random number from a small seed (Section 4.1). The second option is the design of private hash functions, that avoid expensive properties not needed for privacy protection such as collision-resistance (Section 4.2).
5. Finding one such private hash function, CRC-MAC, by testing the monomial structure of candidate functions for whether the function's structure is distinguishable from random. If a function's structure is indistinguishable from random, a cipher that resists known cryptanalysis can be built from the function. We use the automated tests to find the strongest construct from a large set of (mostly weak) designs, all of which achieve a very small implementation overhead by reusing the CRC circuitry that is already present on RFID tags (Chapter 5).
6. Providing a design method that guides designers in choosing the best balance between the different tradeoffs to maximize privacy within a given server cost budget (Chapter 6).

7.2 Limitations

The techniques described in this dissertation rely on several assumptions. The assumptions were chosen to describe the constraints of current and future RFID systems as accurately as possible and predict the privacy threats that may arise from RFID technology to the best of our knowledge. Note that the discussed protocols aim to achieve privacy, but not authentication, which would expose them to a different set of possible attacks.

Limits of Threat Model. We assume that the secret keys can be stored securely on the back-end server and that the communication channel between the server and legitimate readers can be secured. We are considering rogue readers that do not know all of the keys on any of the tags they track. Our model covers neither tracking through tags that the attacker placed on the victim, nor tracking by former owners in cases where the secret keys are not updated. Furthermore, our threat model assumes that RFID scanning incurs costs. Attackers are assumed to act rationally in that the expected return of an attack must exceed its cost. Irrational attackers, should they exist, might not be defeated by our privacy measures.

This dissertation is only concerned with information leakage on the radio link between RFID tags and readers. Other important privacy hazards created by RFID technology are not considered. These additional privacy threats include the insecure storing of RFID readings and information side-channels of RFID transactions on the Internet (i.e., through DNS caches when the ONS lookup service is used [22]).

Emphasis on RFID protocol information. We acknowledge that information is leaked through many channels, and therefore provide a privacy metric for integrating information from different sources. However, we focus our analysis on the protocol layer because we believe that the magnitude of RFID protocol information outweighs the magnitude of all other information sources that can be collected cheaply. Our estimate for the trade-off between privacy and cost will need to be revised should other easily-collectable information sources be found that enable tracking.

Cryptographic assumptions. All protocols discussed in this dissertation rely on an RFID tag's ability to generate unpredictable random numbers and the irreversibility of the used hash function.

HB-based privacy functions rely on the LPN problem, which needs to be shown to be hard on average to be a good foundation for privacy. Cheap one-way hash functions such as CRC-MAC—the second alternative for implementing privacy protocols on RFID tags—rely on the ability of automated randomness tests to detect structural weaknesses of cryptographic constructs. The lack of conclusive randomness tests and the prohibitive cost of exhaustive tests, however, mean that we must resort to heuristics.

7.3 Open Questions

The results of this dissertation open several avenues for future research.

Quantifying information sources. In designing a privacy system, the right balance between information disclosure and privacy needs to be found. A level of protection that discourages all potential attackers is sufficient. Towards this end, more research is needed that measures the amount of information leaked through various side channels and estimates the cost of collecting the information.

Estimating the value of information. We present techniques for increasing the cost of an attack beyond the expected value of the information learned in the attack, but do not quantify this value. To choose optimal design parameters, a better understanding of the true value of data such as customer profiles and inventory statistics is needed. The value of private information could be estimated from the market price of available data such as credit card transactions and Internet browsing traces.

Modeling realistic attackers. Further research is needed to understand how attackers can use data mining techniques to overcome the noise caused by fuzzy information sources, swapped tags, and environmental influences. The attack value will depend on the attacker's ability to employ data mining, which is encoded in the attacker strategy function. Furthermore, better estimates are needed on the cost of alternative tracking technologies in order to better understand the added privacy risk of RFIDs.

Cipher design. Designing cryptographic primitives specifically for privacy is a new idea that

necessitates further study. The heuristics we used in testing CRC-MAC show its resistance against all known classes of statistical attacks, such as differential and linear cryptography, but do not exclude the possibility that new attacks can be found. If new attacks can be found through further research, we will gain insights into which heuristics should be applied in the test of a cryptographic function. Hence, we encourage cryptanalysis of our construct to find better tests that will lead to better constructs.

7.4 Summary

Privacy protection cannot be achieved economically by simply extending known authentication protocols. On the one hand, achieving perfect privacy for large RFID systems is not possible. Despite this, sufficient protection from all likely threats can be achieved economically. Protocols that achieve very low levels of information leakage are orders of magnitude cheaper than protocols that achieve zero information leakage. Similarly, cryptographic functions custom-built for privacy are an order of magnitude cheaper than general-purpose primitives.

We have shown that a more realistic and comprehensive privacy metric leads to improved protocols and new cryptographic primitives that can provide sufficient privacy within the economic and physical constraints of RFIDs. A privacy protection scheme for RFID tags can be realized economically when all its building blocks are designed to provide only those properties that are essential for privacy.

Bibliography

- [1] Gildas Avoine, Etienne Dysli, and Philippe Oechslin and. Reducing time complexity in RFID systems. In *Selected Areas in Cryptography*, volume 3897/2006 of *Lecture Notes in Computer Science*, pages 291–306. Springer, 2006.
- [2] Gildas Avoine and Philippe Oechslin. RFID traceability: A multilayer problem. In *Financial Cryptography and Data Security*, volume 3570/2005 of *Lecture Notes in Computer Science*, pages 125–140. Springer, 2005.
- [3] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-tags. In *Pervasive Computing and Communications Workshops, 2007*, pages 217–222, White Plains, NY, March 2007.
- [4] Matthias Bauer, Benjamin Fabian, Matthias Fischmann, and Seda Gürses. Emerging markets for rfid traces, June 2006.
- [5] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *Advances in Cryptology CRYPTO 06, Lecture Notes in Computer Science*. Springer, 2006.
- [6] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In *Advances in Cryptology-CRYPTO 90*, volume 537/1991 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1991.
- [7] Alex Biryukov and David Wagner. Slide attacks. In *Fast Software Encryption*, volume 1636/1999 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

- [8] Avrim Blum, Adam Kalai, Hal Wasserman, and Carnegie Mellon University. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50:2003, 2000.
- [9] Manuel Blum. A secure human-computer authentication scheme. Technical report, 2000.
- [10] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727/2007 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
- [11] Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Avi Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled RFID device. In *14th USENIX Security Symposium*, pages 1–16. USENIX, 2005.
- [12] Shekhar Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, 1999.
- [13] Levente Buttyán, Tams Holczer, and István Vajda. Optimal key-trees for tree-based private authentication. In *Privacy Enhancing Technologies Workshop PET*, pages 332–350. Springer, 2006.
- [14] Claude Castelluccia and Mate Soos. Secret shuffling: A novel approach to RFID private identification. In *Conference on RFID Security*, pages 169–180, Malaga, Spain, July 2007.
- [15] F. Cate and M. Staten. The value of information-sharing. Council of Better Business Bureau White Paper, 2000.
- [16] Nicolas T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - CRYPTO 2003*, volume 2729/2003 of *Lecture Notes in Computer Science*, pages 176–194. Springer Berlin / Heidelberg, 2003.
- [17] Nicolas T. Courtois, Karsten Nohl, and Sean O’Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *Cryptology ePrint Archive*, Report 2008/166, 2008.
- [18] Ivan Damgård and Michael Østergaard. Rfid security: Tradeoffs between security and efficiency. *Cryptology ePrint Archive*, Report 2006/234, 2006.

- [19] Dang N. Duc, Jaemin Park, Hyunrok Lee, and Kwangjo Kim. Enhancing security of EPC-global Gen-2 RFID tag against traceability and cloning. 2006.
- [20] Claudia Daz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, pages 54–68. Springer, 2002.
- [21] EPCGlobal. Class 1 generation 2 UHF air interface protocol standard v1.0.9., 2005. www.epcglobalinc.org/standards.
- [22] Benjamin Fabian, Oliver Günther, and Sarah Spiekermann. Security analysis of the Object Name Service for RFID. In *International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing – SecPerU’05*, Santorini Island, Greece, July 2005. IEEE, IEEE Computer Society Press.
- [23] H. Feistel. Block cipher cryptographic system. US Patent 3,798,359, 1971.
- [24] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156/2004 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.
- [25] Martin Feldhofer and Christian Rechberger. A case against currently used hash functions in RFID protocols. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4277/2006 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 2006.
- [26] Eric Filiol. A new statistical testing for symmetric ciphers and hash functions. In *Information and Communications Security*, volume 2513/2002 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2002.
- [27] Henri Gilbert, Matt Robshaw, and Herve Sibert. An active attack against HB+ - a provably secure lightweight authentication protocol. *Cryptology ePrint Archive*, Report 2005/237, 2005.

- [28] Zbigniew Golebiewski, Krzysztof Majcher, Filip Zagorski, and Marcin Zawada. Practical attacks on HB and HB+ protocols. Cryptology ePrint Archive, Report 2008/241, 2008.
- [29] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *Proceedings of the Conference on RFID Security*, July 2007.
- [30] P. Israsena. Securing ubiquitous and low-cost RFID using Tiny Encryption Algorithm. In *1st International Symposium on Wireless Pervasive Computing*, January 2006.
- [31] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, February 2006.
- [32] Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In *Advances in Cryptology CRYPTO 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
- [33] Ari Juels and Stephen A. Weis. Defining strong privacy for RFID. Technical report, 2006.
- [34] Chae Hoon Lim and Taekyoung Kwon. Strong and robust RFID authentication enabling perfect ownership transfer. In *International Conference on Information and Communications Security*, pages 1–20, 2006.
- [35] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing*, 17(2):373–386, 1988.
- [36] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology EUROCRYPT 93*, volume 765/1994 of *Lecture Notes in Computer Science*, pages 386–397. Springer Berlin / Heidelberg, 1994.
- [37] Ralf C. Merkle. Secrecy, authentication, and public key systems. Stanford Ph.D. thesis, 1979.
- [38] David Molnar, Andrea Soppera, and David Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Selected Areas in Cryptography*, volume 3897/2006 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2006.

- [39] David Molnar and David Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 210–219, New York, NY, 2004. ACM Press.
- [40] J. Munilla and A. Peinado. HB-MP: A further step in the HB-family of lightweight authentication protocols. *Computer Networks*, 51(9):2262–2267, June 2007.
- [41] Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 12(1):29–66, 1999.
- [42] Yasunobu Nohara, Sozo Inoue, and Hiroto Yasuura. Unlinkability and real world constraints in RFID systems. In *Pervasive Computing and Communications Workshops, 2007*, pages 371–376, White Plains, NY, March 2007.
- [43] Karsten Nohl and David Evans. Quantifying information leakage in tree-based hash protocols (short paper). In *Information and Communications Security*, volume 4307/2006 of *Lecture Notes in Computer Science*, pages 228–237. Springer, 2006.
- [44] Karsten Nohl and David Evans. Hiding in groups: On the expressiveness of privacy distributions. In *Proceedings of The IFIP TC 11 23*, volume 278/2008 of *IFIP International Federation for Information Processing*, pages 1–15. Springer, 2008.
- [45] Karsten Nohl, David Evans, Starbug, and Henryk Plotz. Reverse-engineering a cryptographic RFID tag. In *17th USENIX Security Symposium*, pages 185–194, San Jose, CA, July 2008.
- [46] Andrew Odlyzko. Privacy, economics, and price discrimination on the internet. In *ICEC2003: Fifth International Conference on Electronic Commerce*, pages 355–366. ACM Press, 2003.
- [47] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to privacy-friendly tags. In *RFID Privacy Workshop*, 2003.
- [48] S. O’Neil. EnRUPT—first all-in-one symmetric cryptographic primitive. In *The State of the Art of Stream Ciphers (SACS)*, 2008.

- [49] Sean O’Neil. Algebraic structure defectoscopy. Cryptology ePrint Archive, Report 2007/378, 2007.
- [50] Jacques Patarin and Audrey Montreuil. Benes and butterfly schemes revisited. In *8th International Conference on Information Security and Cryptology - ICISC 2005*, pages 92–116, 2005.
- [51] Bart Preneel, Werner Van Leekwijck, Luc Van Linden, Ren Govaerts, and Joos Vandewalle. Propagation characteristics of boolean functions. In *Advances in Cryptology EUROCRYPT 90*, volume 473/1991 of *Lecture Notes in Computer Science*, pages 161–173. Springer, 1991.
- [52] Melanie R. Rieback, Bruno Crispo, and Andrew S. Tanenbaum. RFID Guardian: A battery-powered mobile device for RFID privacy management. In *Information Security and Privacy*, volume 3574/2005 of *Lecture Notes in Computer Science*, pages 184–194. Springer, 2005.
- [53] M.-J. O. Saarinen. Chosen-IV statistical attacks on estream stream ciphers. In *eSTREAM, ECRYPT Stream Cipher Project, Report 2006/013*, 2006.
- [54] A. Satoh and T. Inoue. ASIC hardware focused comparison for hash functions MD5, RIPEMD-160, and SHS. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 1, pages 532–537, April 2005.
- [55] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block cipher design. In *Fast Software Encryption, 3rd International Workshop Proceedings*, pages 121–144. Springer, 1996.
- [56] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *Privacy Enhancing Technologies*, volume 2482/2003 of *Lecture Notes in Computer Science*, pages 259–263. Springer, 2003.
- [57] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

- [58] Juan Soto and Lawrence Bassham. Randomness testing of the advanced encryption standard finalist candidates. In *NIST IR 6483, National Institute of Standards and Technology*, 2000.
- [59] Thorsten Staake, Frédéric Thiesse, and Elgar Fleisch. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1607–1612, New York, NY, 2005. ACM.
- [60] Helion Technology. High performance AES (rijndael) cores for Xilinx FPGA, 2007. www.heliontech.com/downloads/aes_xilinx_helioncore.pdf.
- [61] Helion Technology. RSA and modular exponentiation cores, 2007. www.heliontech.com/modexp.htm.
- [62] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology CRYPTO 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
- [63] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology EUROCRYPT 2005*, volume 3494/2005 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.
- [64] Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing*, volume 2802 of *Lecture Notes in Computer Science*, pages 201–212, 2004.
- [65] J. Wolkerstorfer. Is elliptic-curve cryptography suitable to secure RFID tags? In *Workshop on RFID and Lightweight Crypto*, 2005.