This is to certify that the
thesis entitled

LIGHTWEIGHT RFID AUTHENTICATION PROTOCOLS

presented by

LEROY A. BAILEY

has been accepted towards fulfillment
of the requirements for the

M.S. degree in COMPUTER SCIENCE AND ENGINEERING

Major Professor's Signature

May 9, 2008

Date

MSU is an affirmative-action, equal-opportunity employer

LIGHTWEIGHT RFID AUTHENTICATION PROTOCOLS

By

LeRoy A. Bailey

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science and Engineering

2008

ABSTRACT

LIGHTWEIGHT
RFID AUTHENTICATION PROTOCOLS

By

LeRoy A. Bailey

Radio Frequency Identification (RFID) tags are cheap, simple devices that can store unique identification information and perform simple computation to keep better inventory of packages. This feature provides a significant advantage over barcodes, allowing their use in applications throughout various fields such as inventory tracking, supply-chain management, theft-prevention, and the like. However, unlike barcodes, these tags have a longer range in which they are allowed to be scanned, subjecting them to unauthorized scanning by malicious readers and to various attacks, including cloning. Therefore, a security protocol for RFID tags is needed to ensure privacy and authentication between each tag and their reader. This thesis provides three different protocols, RFIDGuard, PAP (A Privacy and Authentication Protocol for Passive RFID Tags), and HashTree, each providing security while separately implemented from one another. RFIDGuard and PAP protocols were developed for the supply-chain market while HashTree is an improvement upon a previously developed algorithm that ran in logarithmic time.

Table of Contents

List of Figures

Chapter 1

Introduction

Radio Frequency Identification (RFID) tags are small electronic components that are used to identify and track objects. They have applications in various fields such as inventory tracking, supply chain management, theft-prevention, and the like. An RFID system consists of an RFID tag (i.e. transponder), an RFID reader (i.e. transceiver), and a back-end database. An RFID reader consists of an RF transmitter and receiver, a control unit, and a memory unit. These instruments work together to transfer and receive information stored on radio waves between a reader and an antenna attached to an RFID tag. This information interacts with stored items upon a back-end database that some readers are able to connect to. Depending on the type of the tag, they also have the capability to perform different functions with the information transferred from a reader.

There are three broad categories of RFID tags: passive, semi-passive, and active. Passive tags are powered by the signal of an interrogating reader and can only work within short ranges (a few meters). Active tags maintain their internal state and power transmission using a battery. Semi-Passive tags are battery assisted tags that use some battery power to maintain their internal volatile memory but may still rely on the reader's signal to power their transmission. They can initiate communication

and operate over longer ranges (several meters), but are also more expensive and bulkier than passive tags. Passive tags, however, are also more popular and cheaper, making them more likely to be used within the broad range of applications stated earlier. Therefore, the protocols in this thesis have only been designed for passive tags.

RFID tags are able to uniquely identify individual items of a product type due to their elongated bit length, unlike barcodes, which only identify each product type. This advantage is particularly useful when the transaction history of each item needs to be maintained or when individual items need to be tracked. Furthermore, RFID tags do not require line-of-sight reading like barcodes, increasing the capabilities of the tag identification process of a reader significantly. Due to the these and other advantages that RFID tags have over barcodes, RFID is increasingly becoming more popular and is expected to replace the current barcode technology in the near future. However, there is also a growing concern among people about consumer privacy protection and other security loopholes that make RFID tags an easy target for malicious attacks. Passive RFID tags in their current form are vulnerable to various types of attacks and thus there is a pressing need to make this technology more secure before it is viable for mass deployment. Therefore, privacy and authentication are the two main security issues that need to be addressed for the RFID technology.

The two primary concerns of privacy with RFID tags are clandestine tracking and inventorying [12]. Clandestine tracking deals with issue of a nearby RFID reader being able to scan any RFID tag, since these tags respond to readers without discretion. Clandestine inventorying on the other hand is a method of gathering sensitive information from the tags, thus gaining knowledge about an organization's inventory. An organization called EPCGlobal [7] manages the development of the Electronic Product Code (EPC), a code in RFID tags that is equivalent to the code used to store information in a barcode. EPC compliant RFID tags have fields to store the

manufacturer code and the product code that makes it easy to follow the inventory patterns of a store [12] or the assignment of ID numbers to employees of a business, for example.

RFID privacy is already a concern in several areas of everyday life. Here are a few examples. Automated toll-payment transponders, small plaques positioned in windshield corners, are commonplace worldwide. In a recent judiciary, a court subpoenaed the data gathered from such a transponder for use in a divorce case, undercutting the alibi of the defendant [26]. Some libraries have even implemented RFID systems to facilitate book checkout and inventory control and to reduce repetitive stress injuries in librarians. Concerns about monitoring of book selections, stimulated in part by the USA Patriot Act, have fueled privacy concerns around RFID [20]. Lastly, an international organization known as the International Civil Aviation Organization (ICAO) has promulgated guidelines for RFID-enabled passports and other travel documents [10], [14]. The United States has mandated the adoption of these standards by 27 "visa waiver" countries as a condition of entry for their citizens. The mandate has seen delays due to its technical challenges and changes in its technical parameters, partly in response to lobbying by privacy advocates. One may see how verification of the information stored within the passport would also become an issue as well. This particular issue brings us to the other security threat in RFID, authentication.

Authentication is another major security issue for RFID tags. Privacy deals with authentic tags unknowingly being scanned by attacking readers, while authentication deals with valid readers being misled by deceptive tags. One example where authentication would play a useful role is when scanning counterfeit tags. It has been shown that one can rewrite what a tag emits onto another tag, effectively making a clone [12]. Therefore, authentication is as much of a concern as privacy is.

The key challenge in providing security mechanisms to passive RFID tags is that such tags have extremely weak computational power because they are designed

to be ubiquitous, low-cost (i.e., a few cents) devices [3]. Previous solutions have been developed to solve both security threats for RFID tags (such as [22], [4], [28], [8], [23], and [16]); however, these solutions are not suitable for passive RFID tags. For example, many protocols (such as [22], [4], and [28]) for RFID authentication use heavy duty cryptography. Some previous protocols (such as [8], [23], and [16]) address the privacy issues of RFID systems by requiring users to carry a large device on a daily basis, which seems to be impractical.

In order to deal with these issues, three protocols were developed: RFIDGuard, PAP (A Privacy and Authentication Protocol for Passive RFID tags), and HashTree. Though the goal of each protocol is to provide security for RFID tags, they were each developed to be executed within a different field of RFID technology. RFIDGuard and PAP were developed for use within the supply-chain market and HashTree was developed for use within a secure system. Therefore, each protocol contains a different strategy to achieve security within these different fields. RFIDGuard provides a kind of pseudonym challenge-response protocol, but one that is carefully interwoven with pseudonym rotation. Its security is based upon the fact that any third-party would not have access to any of the pseudonyms for a tag, therefore being labeled an unauthorized by an official tag or reader. The PAP protocol continues in a challenge-response like manner, but involves the use of hash-based functions instead of pseudonyms. Hash function algorithms such as the latest version of SHA have been widely accepted as a secure form of protection for transferring data within a limited computational range. The last protocol, HashTree, was developed from a previously developed protocol that also uses hash functions. Some previously developed hash-based protocol have improved the searching time for a tag within a reader's back-end database to *logarithm* complexity. HashTree continues to improve upon this method by dynamically updating the contents of a tag upon its authentication, allowing further security for a system if the tag were compromised.

This thesis will discuss each protocol mentioned in detail. Chapter 2 will give an overview of previous protocols developed to achieve similar goals, including a brief introduction into the protocol HashTree was developed from. Chapter 3 describes a system and threat model used specifically for the RFIDGuard and PAP protocols. The next three chapters provide a detailed explanation of each individual protocol, where the RFIDGuard protocol is explained in Chapter 4, the PAP protocol in Chapter 5, and the HashTree protocol in Chapter 6. In Chapter 7, we analyze the security of each protocol against a set of accepted standards for all RFID related protocols. Lastly, concluding remarks are given in Chapter 8.

Chapter 2

Related Work

Large organizations such as Wal-Mart, Procter and Gamble, and the U.S. Department of Defense have increased the focus upon RFID technology in recent years due to their need of using RFID as a tool for automated oversight of their supply chains [12]. However, the extended reading range of an RFID tag enables third-party users to eavesdrop upon transmissions between authenticated readers and tags, thus enabling a security breach. Therefore, many researchers have developed unique ways to combat this issue, including external devices to block outgoing and incoming signals between tags and readers to developing internal privacy and authentication protocols similar to those discussed more in detail within this thesis. This chapter introduces a few of the previously developed approaches to the RFID security issue. Some of them may not directly correlate with any of the protocols mentioned in this thesis, but they provide more insight into the research within RFID technology as a whole.

2.1 Tag 'killing' protocols

The first approach to dealing with consumer privacy was developed by the company that will oversee the barcode to RFID transfer, EPCGlobal Inc.. Their approach

is to just "kill" the tag [7]. In other words, the tag will be made inoperable, preventing it from being scanned by malicious readers. This procedure is preformed by the reader sending a special "kill" command to the tag (including a short 8-bit password). For example, after you roll your supermarket cart through an automated checkout kiosk and pay the resulting total, all of the associated RFID tags will be killed on the spot.

Though killing a tag may deal with consumer privacy, it eliminates all of the post-purchase benefits for the consumer. Examples of these types of post-purchase benefits include items being able to interact with what are being called "smart" machines. For example, some refrigerators in the future will interact with the RFID tags on food items. This benefit will allow the refrigerator to scan what items you normally buy, and once it notices that a certain number of items have been removed over a period of time, it will inform you of what items are missing so you may purchase more. Another example of a "smart" machine would be a microwave. The microwave would scan the RFID tag attached to the purchased item and automatically set the timer to the correct amount of time needed. From these examples, you can see that killing a tag would not be an appropriate approach to deal with consumer privacy.

2.2 Cryptography protocols

Due to the invention of "smart" machines and other devices that will need to reuse a tag's information repeatedly, researchers began developing ideas that will insure one's privacy and mutually authenticate a tag and a reader to one another without actually killing the tag upon its scanning. One approach developed to fit this criterion borrowed the concept of public key cryptography [25]. This cryptographic approach is a simple and well defined algorithm to be implemented. An example of this type of protocol would be as follows. Consider a matching pair of public and

private keys for both a reader R and a tag T Each tag would initially be embedded with its reader's public key, R_{pu}, and its own unique private key, T_{pr}. Upon query from a reader, both a tag and the reader may mutually authenticate each other using the tag's embedded keys. Tag T would encrypt an arbitrary nonce n with its private key within an additional encryption using the reader's public key $R_{pu}(T_{pr}(n))$ and send it to the reader. Once the reader has decoded the outer encryption using its private key R_{pr}, it will search its back-end database and obtain the tag's matching public key T_{pu}, decrypting the inner lock to retrieve the nonce from the tag. The reader would then re-encrypt this nonce with its own private key, and similar to the tag, it would establish a second encryption around the previous one with the tag's public key, $T_{pu}(R_{pr}(n))$. Before establishing this encryption, the reader would use its keys in combination with the current tag's to form a temporary key T_k, which would be sent along with the nonce to the tag. Upon retrieval and decryption of the message sent from the reader, the tag would use this T_k to send any further information between itself and the reader, such as a person's job title to enable them access to a certain location of a building. Figure 2.1 illustrates such a protocol. For a protocol that would require a faster authentication process, any important information from the tag would be embedded within the first message sent, such as a product's EPC code [7] for a supply-chain marketing system. This cryptographic approach prevents the unwanted listening of the transmitted messages from a tag unless a hacker was to retrieve the private key of that tag, which is very unlikely in a given short period of time.

Despite the security strength of this approach, public key cryptography (especially for execution within RFID tags) requires a large amount of computational power in order to encrypt transferring messages. This approach not only increases the size of the tag due to the extra amount of circuitry a tag would need to hold, but it also significantly increases the cost of each tag, causing this approach to be used within

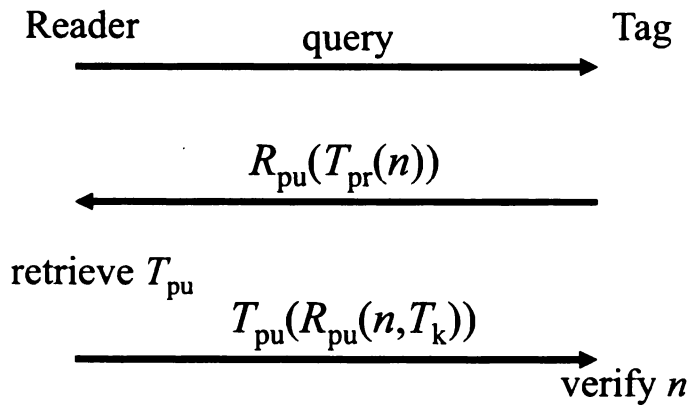

Figure 2.1: An example cryptography protocol

very limited sectors of RFID security.

2.3 Faraday's Cage

One of the first approaches in dealing with consumer privacy involves what is called a Faraday Cage [12]. A Faraday Cage is container made of metal mesh or foil that is designed to block certain radio frequencies. In fact, this method of shielding an item with certain metal material is known to be used by thieves when trying to decept shoplifting detection systems. Recently, the U.S. State Department has even indicated that U.S. passport covers will include metallic material to limit RF penetration, thus preventing long-range scanning of closed passports [9, 1]. This approach, however, has disadvantages The main disadvantage is that most cages are not designed to fit around certain items, such as wrist-watches, containers, and bigger items such as televisions or computers. This disadvantage limits the use of this approach, restricting it from more commercial enviornments such as the supply-chain market.

2.4 Active Jamming Device

Another approach dealing with consumer privacy is called the active jamming approach [12]. This approach will allow an individual to carry a device that would block nearby RFID readers by transmitting or broadcasting signals from the device. However, this approach could be illegal if the broadcast signaling power of this device is too high. This brute-force method might cause the jammer to interfere with surrounding legitimate RFID readers. Therefore, this approach is not a suitable solution for the privacy protection of RFID.

2.5 Blocker Tag

The protocol behind the Blocker Tag helped towards the development of RFID-Guard. The approach Juels uses in [15] is similar to that of the "jamming" approach described earlier; however, its effect is not as strong upon operation, cleverly interacting with the RFID "singulation" protocol to disrupt only certain operations. The singulation protocol for RFID tags is a tree-based method upon which a reader may tell apart multiple tags scanned at the same time. This process works by repeatedly querying all present tags within the area, distinguishing each separate tag by keeping a count of the number of collisions a reader has received. For a more detailed description of this procedure, consider the example given in Figure 2.2. This figure represents a tree with a depth of 3 containing $2^3 = 8$ tag serial numbers represented at each of its leaves. Assume we want to distinguish between tags '001' and '011'. A reader would first query for all tags with '0' prefix. This query would return a collision since both tags have this criterion. Upon querying for a '1' prefix, no signals would be returned; therefore, the reader would not continue to query for any tags beginning with that prefix. The singulation protocol would then follow the recursion upon the collision points of the tree as before to eventually reach the corresponding

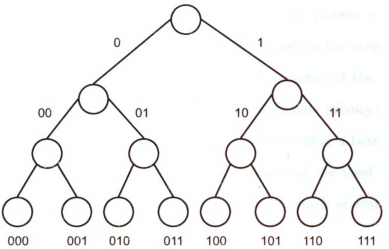

Figure 2.2: A singulation protocol tree

leaves '001' and '011', receiving only one response back denoting their presence.

In order to interact with this protocol, in [15], Juels uses a special bit in tags called a "privacy bit" that can take a value of 0 or 1 and can be easily toggled by a reader after authenticating with a unique pin for that tag. The tag bears a value of 0 when inside a store, indicating that it has public access and on checkout the tag is moved to a "private" zone by flipping the privacy bit to 1. But doing just this does not ensure security of the tag. An additional specialized tag called the "blocker" tag must accompany the tag in question in order to secure it [15]. The blocker tag confuses malicious readers into thinking that tags with all possible values are present with the bearer. The tag specifically achieves this by sending the corresponding value a reader is currently querying for within the singulation protocol. For example, if a reader is asking for a serial number beginning with '11', the blocker tag would send this value in order to collide with any tags that may actually contain this value, confusing the reader upon reaching the leaves of its tree. This "confusing act" can either be done to overwhelm the reader (full blocker) or in a "soft" or polite way (partial or selective blocker) [13]. A soft blocker, however, would only interact with leaves that are within a "privacy zone". For example, as described earlier, the privacy bit of a tag would

change from 0 to 1 upon authentication of tag. The soft blocker would then only protect tags with a prefix of '1', allowing other items within the area to be scanned if necessary. Either way, the tag is secure only in the presence of the blocker tag.

This concept is specifically designed to promote consumer privacy; however, given the difference in signaling strength of each tag, they may hold a better place within different retail environments. A normal blocker tag may be used within cellular phones to disrupt malicious transmissions to trying to attack or retain information from different calls or text messaging. A soft blocker would be more suited for the supply-chain market, and thus would be embedded within a grocery or shopping bag. This temporary method of security provides the advantage of allowing "smart" machines as described earlier access to certain RFID related items without any additional processing. However, as the previous two devices, this concept has its flaws as well. Even a well-positioned blocker tag has a chance of failing given the unreliable transmission of RFID tags [12]. Also, readers may eventually evolve in exploiting weaknesses to blocker tags and overtake their signal strength [19]. In order to fully understand the attacks and defenses upon this approach, research and evaluation will have to continue before any deployment is considered.

2.6 Hash Lock: A hash function based protocol

Upon the failure of separate mechanisms devised to provide efficient authentication to RFID tags, researchers began developing schemes to provide such security within the tag themselves. One of the first internally designed protocols is known as *hash lock* [27]. This hash-function based approach deals with unlocking a tag value through hash based key values to gather its secret information. The tag is initially set to a 'locked' state, where a reader sends a lock value to the tag, $lock = hash(key)$, where the key is a random value. This random value is stored within the tag's re-

served memory location (i.e. a Meta-ID value), which upon entering the locked state, the tag will not transfer any private information about itself other than what will need to be given during the authentication process. To unlock the tag, the reader must send the tag the original key that was used to make its Meta-ID value. Upon receiving Meta-ID value, the tag performs a hash function using the key and compares the function's result to its Meta-ID value. If this result matches the Meta-ID, the tag unlocks itself, allowing its EPC Code [7] to be transferred to readers upon future queries. This protocol is very simple and straightforward in providing security against the breach of secret information held within the tag, i.e. its EPC Code. Since only authorized readers know the original key value of each tag, only they are able to unlock the tag for its information. To secure a tag, authorized readers may lock the tag again after reading the code.

Despite its simplicity, this protocol provides a heavy breach in security. It fails to provide mutual authentication between the tag and reader, only authenticating the reader since it must provide the key value that should be unknown to the public. This breach can simply be exploited by a malicious third party scanning the tag for its Meta-ID value. This value in turn would be randomly broadcast to nearby readers where eventually the key value for that particular tag is returned. The party may then use this information to send to the original tag, obtaining its EPC code and other sensitive information.

To combat the above mentioned flaws, the same authors of the paper [27] devised a new approach, dealing with randomization. The emphasis here was to disguise the Meta-Id value with a random number each query, thus the tag and its value could not easily be traced. Therefore, an additional pseudo-random number generator is embedded into the tag for this approach. Similar to the original hash lock protocol, a tag will pre-store a key value (known as an ID) to place itself in a locked state. However, this approach will not store the hashed result of this ID, but rather upon

each query, the tag will hash its ID with a random value given by its pseudo-random generator, resulting in hash(ID_k, r), where k represents the kth tag among a number of tags within the system, ID_1, ID_2, ..., ID_k, ..., ID_n.

Upon the reader's query of a tag, the reader will obtain two values. These values included the random number generated by the tag and the hash function result value generated by the hash against the ID value of the tag and its current random number at that time. To unlock the tag, a reader must send the tag its original ID value. Therefore, the reader will begin to search its back-end database containing all ID values for all tags, where it must repeatedly perform a hash function against each separate value with the random number given from the tag. This will allow the reader to compare each hashed result against the one sent from the tag, wherein if they match, the reader will obtain the matching kth ID value, returning it to the tag to unlock itself. Once the tag is in an unlocked state, any reader may perform a query to gain the tag's EPC information.

In addition to successfully achieving security on RFID tags, this approach also provides location privacy. In the previously developed protocol Hash Lock, each tag still reveals its Meta-ID. However, this approach only discloses a random number and a hashed value based from that number. Therefore, a malicious third party can not trace a specific tag (i.e. a product of a store) based from its Meta-ID. In this case, the randomized hash lock protocol is able to provide location privacy.

Despite the vast improvements that this randomized protocol presents over the original hash lock scheme, this approach may not be suitable for all cases. Due to the fact that the reader must search through as many ID values as possible to find the matching hash result, its running time is that of $O(n)$, for n tags within the system. Therefore, this approach would have a vast scalability problem when the number of tags for a system increases enormously. The additional cost of producing a pseudo-random generator for a tag also presents another problem for a system of

that scale.

2.7 Tree-based approaches

As explained earlier, the randomized hash based approach in [27] fails to maintain a less than optimal running time and low-cost tags against a system whose number of tags increases expeditiously. However, as this type of system's popularity continues to expand over time, the search to find an efficient algorithm grows larger. In an effort to decrease the running time of a hash based function while maintaining its security, authors of [6, 5, 21] developed an approach which improves the key search efficiency from linear complexity to logarithmic complexity. The key in achieving this is based upon the structure of the back-end database, which is set up as a tree based graph. Every node will then require $O(\log N)$ search time, as proven in multiple mathematical theories. One of the first groups to use a tree-based approach was Molar et. al [21], who in their approach used a challenge-response protocol. This procedure required multiple rounds to identify a tag, each round consisting of three messages between both parties. Due to the tree-based nature of the back-end database, however, this caused each round to have a $O(\log N)$ running time, incurring relatively large communication overhead between each message. Therefore, another paper [6] made an improvement upon this algorithm that shortened the length of a round to only one message from the tag, providing no further interaction between the tag and the reader. Despite improving the algorithm however, all tree-based approaches to this point still have a similar problem: since all tags share keys with other tags within the tree, compromising one tag would result in compromising other tags as well. This problem is known as the *compromising* attack.

In order to resolve this issue, the keys of both the reader and the tag must be simultaneously updated. To our knowledge, [18] is the first paper to discover this

flaw and develop its own dynamic key-updating approach to this problem. In [18], the back-end database for a tree contains a set of keys k in a similar binary tree structure from that of [6]. In order to update the tree however, each node contains an additional temporary key tk. Initially, each temporary key equals the value of the current key for each node, $tk = k$. The authentication procedure for a tag is essentially the same as a normal tree-based approach, where the tree is repeatedly searched for a path leading to a matching leaf node that represents a tag. Upon authentication of a tag, the reader performs a hash function upon k and sets its results as its current key value, $k = \mathrm{h}(k)$. The temporary key is then always updated to the current key's value beforehand. This is to insure that every search for a key's path to its leaf node is correctly calculated, since the protocol will check both the current key value k and the temporary tk value for a node during authentication of a tag. Though this protocol prevents many of the problems with [6], it is only suitable for systems that plan on never scanning the same tag twice. This is due to the value of the temporary key only being capable of holding one key value at a time. If a tag were to be scanned multiple times, the temporary tk value of a node would only hold the ith - 1 current key value of a node of i scans. This would not only allow paths sharing nodes with this tag's path to be wrongfully unauthorized, but would eventually misauthorize the repeatedly scanning tag as well. This flaw provides the background for HashTree protocol discussed in this thesis. The HashTree protocol will simultaneously update the keys of a reader and tag upon each authentication, while still allowing non-previously queried tags to continually authenticate themselves with ease.

2.8 Minimalist Cryptography

Many active tags contain the ability to re-label themselves in a fashion that is indistinguishable to third-party malicious attackers, but may still be authorized by certified readers. Since passive tags to do not contain enough power to withhold this ability, many researchers began developing additional items to somehow block or distort the transmission of the tag from hackers, as described earlier within this section. However, Juels has developed a protocol called the "minimalist" system [11], allowing the re-labeling of a passive tag within its limited computational capabilities. Within this system, each tag contains a small set of pseudonyms; a different pseudonym is then given to a reader upon query since each tag will rotate to through its list of pseudonyms per each scan. The security holds in that only an authorized reader will contain the entire set of pseudonyms. An unauthorized reader does not contain each pseudonym for a tag, and thus will not be able to gain any secure information from the tag given its different appearances. A more thorough example of this scheme is provided below.

As previously stated, each tag contains an array of pseudonyms α_i, where i denotes the current pseudonym of a tag for m pseudonyms, $1 \leq i \leq m$. However, the protocol would not be secure from these pseudonyms alone. If that were the case, the minimalist system would be vulnerable to the cloning attack, an attack of which many static designed protocols for RFID tags suffer from [24]. In this case, an attacker would query the tag, thus obtaining its current pseudonym α_i and replay this value to the reader, allowing itself to be recognized as the currently scanned tag. To combat this issue, a tag only verifies itself to a reader after the reader has authenticated itself to the tag. To accomplish this, for every pseudonym α_i, each tag contains two additional key values, β_i and γ_i, for which the tag and reader will authenticate themselves with. Upon query by a reader, a tag will respond with its current α_i pseudonym value. Assuming the value is valid, the reader will retrieve

the tag's corresponding β_i and γ_i values from its back-end database and send the tag β_i. Upon authentication of the reader, the tag will respond with its corresponding γ_i value. As you can see, this protocol mimics that of a simple challenge-response protocol, but one that is designed upon pseudonym rotation.

In order to successfully achieve long-term security for a tag, one must continually update its α_i, β_i, and γ_i values. Logically, this update method would need to occur after each mutual authentication between a tag and a reader to maintain the low-cost of not periodically updating a large amount of tags at once. However, updating the tags also presents a new problem: an attacker may still eavesdrop or tamper with the updating process. To address this issue, Juels proposed using one-time pads that have been used across multiple authentication protocols to update the current values of a tag and a reader. Using this method, a malicious third-party who only eavesdrops periodically will be unlikely to gain the updated α_i, β_i, and γ_i values.

One-time pads [2] may be thought of as a simpler form of encryption than that used within cryptography; thus, they are able to be used within tags that have smaller computational capabilities (i.e. passive tags). One-time pads are essentially a random bit string of length l. If two parties then share a secret one-time pad δ, it has been proven that a message M may be sent secretly via cipher text $M \oplus \delta$ between both parties, where $\oplus$ denotes the XOR operation. Thus, after mutual authentication between both the tag and the reader, the reader uses these one-time pads to update the α_i, β_i, and γ_i values noted earlier and sends these pads to the tag so that it may update its values as well. Provided a malicious third-party doesn't eavesdrop upon a reader transmitting the message and obtain these pads, they achieve no knowledge of the newly updated tag values. However, in case a third-party were to obtain one of the pads, this scheme also has an additional spin on what is consider to be the normal use of a one-time pad. This involves using the one-time encryption across multiple authentication sessions. To achieve this, pads from two different

authentication sessions are XORed with a given tag value w to update it, where $w \in \alpha_i \cup \beta_i \cup \gamma_i$. Therefore, even if a third-party successfully obtains a pad used in a prior session, it is seen that they will still not be able to obtain any information about the updated values of w.

The minimalist approach offers resistance against spying upon corporate businesses, such as the clandestine scanning of product stocks in a supply-chain market. Since our first developed protocol was made for retail environments, we borrowed this idea of pseudonym rotation along with Juels soft-blocking technique [13] to develop the RFIDGuard protocol discussed later in this thesis.

Chapter 3

Modeling

Upon developing the RFIDGuard and PAP protocols, similar modeling techniques were used in the limitation of security due to each system being developed for a retail environment. This chapter discusses those limitations, separately listing tags requirements for both protocols and stating what attacks are assumed for a third-party to try and carry out upon both systems. This chapter does not however apply to the HashTree protocol, due to its broader range of scale within most fields in RFID technology.

3.1 System Modeling

This section models the security properties that RFIDGuard and PAP are expected to achieve. It begins by describing our assumptions concerning the readers and tags being used. It then discusses the security and privacy properties that both protocols are expected to achieve.

Readers and Tags

The two principal parties involved in both protocols are *readers* and *tags*. It is not considered how the reader will distinguish between multiple tags because this is

handled by singulation protocols [17] and it is out of the scope of this thesis. The existence of a malicious reader is also considered; however, since both an authorized reader and a malicious reader will receive the same information in both protocols once privacy is established to the tag, they are both simultaneously referred to as readers.

There are three types of authorized readers in both protocols: *inventory readers, checkout readers, and return readers.* An inventory reader is the most basic reader of these three, only allowing the ability to query the tag. A checkout reader contains all of the functions of an inventory reader as well as the ability of connecting to a back-end database. The information retrieved from the back-end database could be used by the checkout reader to authenticate itself to a tag. A return reader has the same functionality as a checkout reader.

Class 1 Generation 2 tags are considered to be the standard tags for use within both protocols, where these tags were standardized by EPCglobal [7] in 2004 for passive RFID tags. This global standardization has been adopted by US Department of Defense, Wal-Mart, Metro AG, etc [3]. Class 1 Generation 2 tags have four memory banks: Reserved Memory Bank (which as at least 32 bits for storing information such as the password for killing a tag), EPC Memory Bank (which as at least 496 bits for storing EPC information), TID Memory Bank (which as at least 32 bits for storing tag identifier), and User Memory Bank (for storing information related to the tag's application). Note that the upper limit of the user memory bank in a tag is not specified in the standard. In other words, the size of the user memory bank of a tag depends on how much memory the manufacture puts on the tag. Both protocols only require a small amount of memory, which could be allocated from the user memory bank of a tag. The RFIDGuard protocol only requires a tag to perform four simple operations: comparing two numbers, calculating modular addition, storing and retrieving a number in user memory bank, and flipping a bit. The PAP protocol only requires a tag to perform three operations: comparing two numbers, execute a

hash function, and storing and retrieving a number in user memory bank, and flipping a bit. These operations could be easily implemented on Class 1 Generation 2 tags.

Another important idea that needs to be discussed involves the general protection of messages traversed between readers and tags. This involves the *cover-coding* mechanism that has been standardized for Class 1 Generation 2 tags. As stated earlier, the signal transferred by a Class 1 Generation 2 tag is only up to few meters; however, the signal from a reader could travel as far as one kilometer [3], allowing the information sent from a reader to a tag to be eavesdropped by an attacker who may be out of sight. In order to prevent this, each Class 1 Generation 2 tag incorporates the mechanism of *cover-coding*. In this procedure, when a reader queries a tag, the tag first generates a 16-bit random number and sends it to the reader. Note that this random number only travels a few meters. In the subsequent communication between the reader and the tag, all messages are XORed with the random number. Therefore, as long as attackers are not physically within a few meters, they cannot decode the messages sent out from a reader. In essence, the cover coding mechanism uses the widely known concept of the one-time pad discussed in Chapter 2.

Security and Privacy Requirements

Each protocol (RFIDGuard and PAP) strives to achieve two requirements: authentication and privacy. In terms of authentication, a tag and a reader should be able to achieve mutual authentication, that is, a tag should be able to authenticate a reader and a reader should be able to authenticate a tag. In terms of privacy, a tag should only give out private information to authorized readers.

3.2 Threat Modeling

Previous research has some assumptions on practical attacks on RFID systems, some of which we entail into the first two protocols. First, a malicious reader cannot eavesdrop upon a reply from a tag to an authorized reader unless the malicious reader is a short distance (i.e., several meters) from the tag. This is because the backscatter (reply) channel between the tag and the reader is assumed to have a relatively short range (i.e., several meters) compared to that of the forward channel due to low cost of passive tags. Second, it is not easy for an attacker to hide himself between a legitimate reader and a tag in an active session. Third, it is not easy to intercept a message and modify the message over the air in real time. These three assumptions are made due to the fact that all authentication procedures will take place inside a retail store. Therefore, it is assumed that a retail store will have some security mechanisms that prevent unauthorized readers from entering the store. This can be easily achieved by installing detection devices near the entrance of the store to detect unauthorized readers [17]. The last assumption states that the limitations of a retail tag itself can not be modified, due to it being a passive tag.

After making a list of assumptions as to what an attacker can not perform, it is assumed that attackers have two primary abilities. First, they have the ability to query a tag as any normal reader would. Second, they have the ability to clone a tag.

Chapter 4

RFIDGuard

This chapter discusses the structure of the RFIDGuard protocol. This protocol is broken into four parts, each denoting a heightened or limited level of security denoting where a current tag is placed within the process of a retail environment.

Section 4.1 provides an overall look of the protocol, while the other four sections describe the protocol in detail. Section 4.1.1 describes the protocol as if a tag were inside the store. Section 4.2 describes the protocol once the tag has arrived to the checkout counter. During a product's use outside the retail area is where Section 4.3 denotes the protocol. Lastly, if a product needs to be returned to the store, Section 4.4 will describe that procedure.

4.1 Basic Design

The basic idea of the RFIDGuard protocol is as follows. Each tag attached to a product holds (1) a list of pseudonyms, which is essentially a list of random numbers associated with the product, (2) a pointer (i.e., an index number) representing the current index in the list of pseudonyms (initially set to 0), (3) a privacy bit, which indicates the tag is in the privacy state or in non-privacy state, (4) a number that represents the generic name of the product, and (5) some private information (i.e.,

ID, EPC code, etc.). In order to establish authentication between a tag and a reader, the tag sends its generic name and current pointer to the reader. The reader uses this information to determine the next pseudonym that the pointer points to in the tag and sends that pseudonym to the tag. The tag then checks whether the received pseudonym is correct. If so, the tag authenticates the reader. Upon reader authentication, the tag sends the next pseudonym back to the reader. The reader then checks whether the received pseudonym is correct. If so, the reader authenticates the tag.

Briefly stated, we propose a kind of pseudonym challenge-response protocol, but one that is carefully interwoven with pseudonym rotation. In order to establish privacy, upon checkout, the privacy bit of a tag is changed to one. At any point that the tag's privacy bit is one and a reader attempts to scan it, the tag will only return enough information for a trusted reader to perform the authentication procedure mentioned above, which include its pseudonym pointer and the number to represent its generic name. Since an unauthorized reader would not contain the list of pseudonyms, the tag will not give out its private information.

One might notice that both the authentication and privacy levels of this tag are not needed at each scan, depending on the location of the tag. Therefore, we have established four different subset protocols to represent the level of assumed security based upon these locations, including the inside of the store, during checkout, the outside of the store, and the process of returning an item. These protocols are explained in detail as follows.

4.1.1 The In-store Protocol

The in-store protocol concerns querying a tag located inside a store. We assume there is an established level of security that does not allow unauthorized RFID readers within a scanning range of these tags; therefore, the in-store protocol is designed to provide no authentication and privacy protection for efficiency purposes. Each tag

when delivered to the store will have its privacy bit set to zero, denoting a location containing only authorized readers. Upon a reader querying a tag, the tag will send the reader its ID, its generic name, and an index that points to its current pseudonyms location. Figure 4.1 illustrates this in-store protocol.

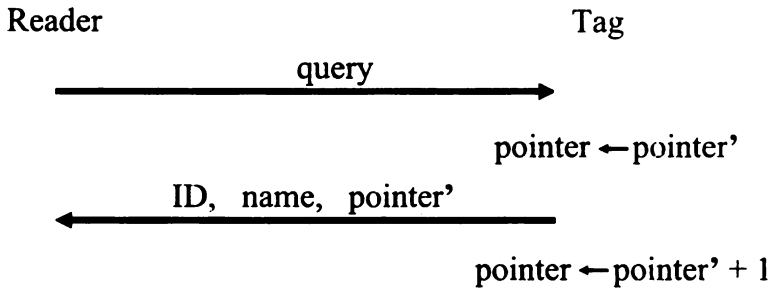

Figure 4.1: The in-store protocol

Though the reader would not need any more information beyond the tag's ID at this time, its generic name and pseudonym location are sent by default to lessen the cost of the tag. If it were to just send its ID, the tag would have to be programmed to know when to send additional information, further increasing the cost of the tag.

After sending the above information, the tag will increase its pointer value by one. This increases the level of security for the tag, allowing the pointer value to not always begin with its first pseudonym upon leaving the store. For example, if the pointer value were to only change during the checkout procedure, it would always begin at the first pseudonym. If an unauthorized reader was brought into the store, each pseudonym could be correctly aligned with the pointer number and a duplicate tag could be easily made. However, since the pointer increases each time, even a person with an unauthorized reader could not figure out which pseudonym to connect to which pointer value when making the malicious tag. This theory of increasing the pointer each time is applied to the following protocols as well.

4.2 The Checkout Protocol

The checkout protocol concerns querying a tag during a checkout procedure. To prevent the use of cloned tags, the checkout protocol allows the reader to authenticate the tag. To ensure that the proper type of reader is used during the checkout procedure, the checkout protocol also allows the tag to authenticate the reader as well. Note that the type of readers used in the in-store protocol and those used in the checkout protocol have different capabilities. The readers used in the in-store protocol only perform simple querying and does not need to connect to back-end inventory databases. In contrast, the readers used in the checkout protocol not only have the ability to query a tag but also can connect to back-end inventory databases. This is a reasonable assumption because the readers used in the in-store protocol are typically hand held mobile devices and the readers used in the checkout protocol are typically hooked to computers. Therefore, in the checkout protocol, a tag also needs to authenticate the reader; otherwise, any employee with a hand-held reader could checkout any product and steal from the store.

The checkout protocol works as follows. The first two steps are the same as the two steps in the in-store protocol. In the third step, the reader retrieves $pseudonym[pointer' + 1]$ from its back-end inventory database using the *name* received in the second step. The reader also generates a random number m, and sends it and the pseudonym to the tag. The tag will then use the sent pseudonym to authenticate the reader because an unauthorized reader does not have access to the back-end inventory database to obtain $pseudonym[pointer' + 1]$. If the tag successfully authenticates the reader, the tag sets its privacy bit from zero to one, denoting the tag's traversal to a location that may contain unauthorized readers, and sends $pseudonym[m]$ to the reader. Similarly, the reader can use this to authenticate the tag because a counterfeit tag does not know $pseudonym[m]$. Figure 4.2 illustrates this checkout protocol.

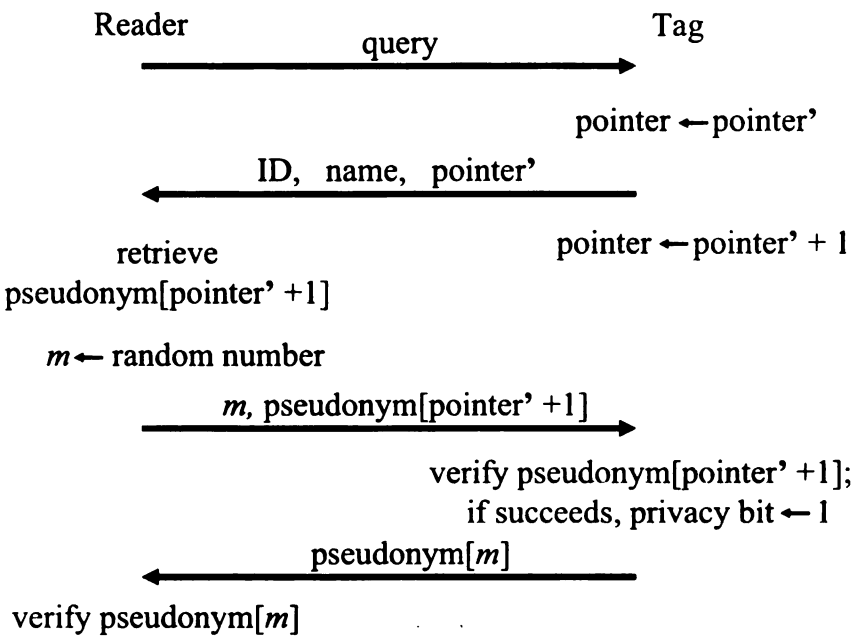

Figure 4.2: The checkout protocol

There are different levels of security obtained by using this checkout procedure. Only an authentic tag and an authorized reader would know all the pseudonyms of a tag. Therefore, if a cloned tag does not contain all of the pseudonyms, it would not send the correct pseudonym to the reader at the end of the procedure, causing the reader to not verify that tag. If the reader does not verify a tag within a time limit, the system will timeout and not allow the customer to finish the checkout procedure. Another level of security denotes the increase in pointers every time the tag is scanned within the store. If the cloned tag were to contain some pseudonyms but they were not aligned with their correct pointer values, the reader would know this because it would not only check that the last sent pseudonym was in the array of the generic name used, but the location of that pseudonym as well. For example, if cloned tag labeled a pseudonym as the pointer value 3 but its actual value within the list should be 5, the reader would catch this and mark the tag as counterfeit. The invariant of increasing the pointer value each time it is scanned still holds even if its privacy bit

is one, as seen in the next section. The last level of security given by this protocol is held within the act of the reader sending a pseudonym to authenticate itself. Since an authorized tag at this point is assuming it is in a location free of unauthorized readers (i.e. its privacy bit is zero), the point of sending the pseudonym is not to authenticate the reader itself, but the type of reader within the store. If all the readers had access to change the privacy bit to 1 (i.e. an inventory scanner), an employee could register items as if they have been sold and steal them from within the store.

4.3 The Out-store Protocol

The out-store protocol resembles a tag's behavior once it leaves the store. At this point, various readers with different levels of security are assumed to be able to access the tag. Therefore, only enough information about the tag is given to allow authenticated readers access in order to flip the tag's privacy bit back to zero. This information includes the generic name and the pointer to its current pseudonym. Since the generic name is represented by a number, an unauthorized reader will not know what items are currently being read. Also, the tag's privacy bit will continue to increase by one on each scan, as it will when the privacy bit is off. This continues the level of security explained in the previous protocol for protection against cloned tags. The next section will explain how the information being given above will allow an authenticated reader to turn the privacy bit of a tag back to zero. Figure 4.3 illustrates this checkout protocol.

Note that there are many reasons that an attacker would want to retrieve a tag's private information. For example, an attacker may want to know what people shop for in certain stores to develop spam or other similar shopping techniques. Also, an attacker may want access to the private information of a tag to gather secrets about the product's producer or the store in general. This type of attack is prevented in

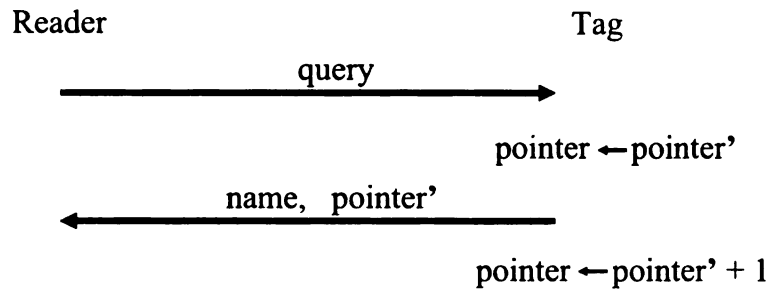

Figure 4.3: The out-store protocol

our out-store protocol.

4.4 The Return Protocol

The return protocol deals with the returning of an item to where it was sold. Many stores have returned items that they are still able to resell. Therefore, these RFID tags need to be set up to enable the in-store protocol. The return protocol requires mutual authentication between a tag and a reader. To prevent unauthorized readers from flipping the privacy bit of a tag from one to zero, the tag needs to authenticate the reader. Though this concept may appear clear, it may not be as easy to understand why the tag needs to be authenticated. If the tag were not authenticated, a person could create a counterfeit tag to indulge the price value of an item. This in turn would allow a customer to increase the price of an item, enabling them to receive a higher amount of money back or exchange the item for a higher valued one.

The return protocol works as follows. The first two steps are the same as the two steps in the out-store protocol. In the third step, the reader retrieves $pseudonym[pointer' + 1]$ from its back-end inventory database using the $name$ received in the second step. The reader will also generate a random number m and send this value along with the psuedonym to the tag. The tag can use the sent

pseudonym to authenticate the reader because an unauthorized reader does not have access to the back-end inventory database to obtain *pseudonym*[*pointer'* + 1]. If the tag successfully authenticates the reader, the tag set its privacy bit from one to zero, denoting the tag's traversal to a location that is free of unauthorized readers, and sends its ID along with *pseudonym*[*m*] to the reader. Changing the privacy bit back to zero allows for its reuse in the store (if needed). Similarly, the reader can use *pseudonym*[*m*] to authenticate the tag because a counterfeit tag does not know it. Similar to the check-out protocol, in the last two steps of our return protocol, each pseudonym sent between the reader and the tag are XORed with the random number that they established for that session using the cover-coding mechanism. Figure 4.4 illustrates this checkout protocol.

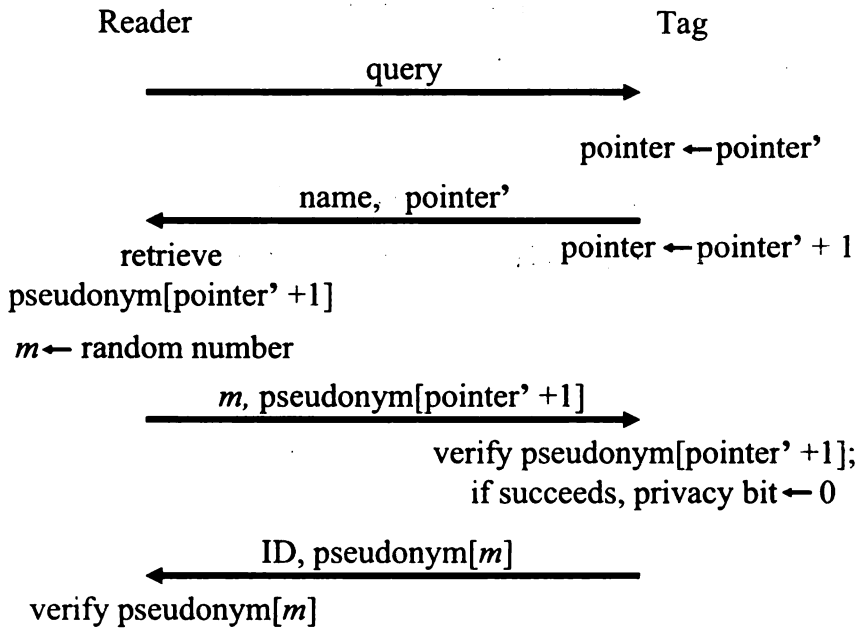

Figure 4.4: The return protocol

Chapter 5

PAP

This chapter discusses the structure of the PAP protocol. This protocol is broken into four parts, each denoting a heightened or limited level of security denoting where a current tag is placed within the process of a retail environment.

Section 5.1 provides an overall look of the protocol, while the other four sections describe the protocol in detail. Section 5.2 describes the protocol as if a tag were inside the store. Section 5.3 describes the protocol once the tag has arrived to the checkout counter. During a product's use outside the retail area is where Section 5.4 denotes the protocol. Lastly, if a product needs to be returned to the store, Section 5.5 will describe that procedure.

5.1 Basic Design

In the PAP protocol, each tag attached to a product stores (1) a secret key k shared by both the reader and the tag, (2) a generic name (i.e., the numeric representation of the product type), (3) an ID (i.e., the EPC code, which is the numeric representation of the individual item), and (4) a privacy bit, where value 0 indicates that the tag is in the non-privacy state (i.e., in store) and value 1 indicates that the tag is in the privacy state (i.e., out store). In order to achieve authentication between

the tag and the reader, the tag first sends its ID (or generic name) and a random nonce to the reader upon query. The reader uses this information to determine the secret key k of the tag and applies a one-way hash function upon it, sending both the hashed result and another random nonce to the tag. The tag verifies the reader by performing the same hash function using its secret key k with the nonce sent to the reader. If this value matches the hashed result sent from the reader, the tag authenticates the reader. The tag will then perform another hash function using its secret key k with the nonce received from the reader and send this hashed value to the reader. The reader then performs the same hash function with its secret key k. If the result matches, the reader authenticates the tag.

In order to establish privacy, upon checkout, the privacy bit of a tag is changed from 0 to 1. At any point that the tag's privacy bit is 1 and a reader attempts to scan it, the tag will only return enough information for a trusted reader to perform the authentication procedure mentioned above, which only includes a number to represent its generic name. Since an unauthorized reader would not contain the secret secret key k, the tag will not give out its private information.

Next, we present the PAP protocol based on four different locations: inside a store, at a checkout counter, at a return counter, and outside a store.

5.2 In-store protocol

The in-store protocol concerns querying a tag located inside a store. We assume there is an established level of security that does not allow unauthorized RFID readers within a scanning range of these tags; therefore, the in-store protocol is designed to provide no authentication and privacy protection for efficiency purposes. Each tag when delivered to the store will have its privacy bit set to zero, denoting a location containing only authorized readers. Upon a reader querying a tag, the tag will send

the reader its ID and a random nonce n_t. Figure 5.1 illustrates this in-store protocol.

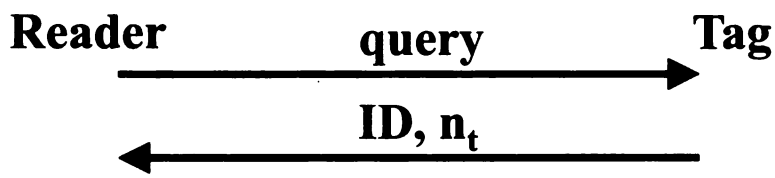

Figure 5.1: The in-store protocol

Though the reader would not need any more information beyond the tag's ID at this time, the random nonce generated by the tag is sent by default to lessen the cost of the tag. If it were to just send its ID, the tag would have to be programmed to know when to send additional information (i.e., the difference between a checkout reader and an inventory or price checking scanner), further increasing the cost of the tag.

5.3 Checkout protocol

The checkout protocol concerns querying a tag during a checkout procedure. To prevent the use of cloned tags, the checkout protocol allows the reader to authenticate the tag. To ensure that the proper type of reader is used during the checkout procedure, the checkout protocol also allows the tag to authenticate the reader as well. As previously mentioned in Section 3, different types of readers exist in the store; therefore, a tag always sends the random nonce n_t in the in-store protocol to save cost. Other readers beyond the checkout console should not have the ability to connect to the database that contains the secret key k associated with the product in order to fulfill the authentication requirements for this protocol. If a tag does not authenticate the reader, an employee with a hand-held reader could checkout any product and steal from the store.

The checkout protocol works as follows. The first two steps are the same as the two steps in the in-store protocol. In the third step, the reader retrieves the secret key

k of the tag from its back-end inventory database using the EPC Code, *ID*, received in the second step. The reader will then perform a one-way hash function on this k and the random nonce, n_t, received from the tag. The reader then generates its own random nonce, n_r, and sends it along with the hash result, $h(n_t, k)$, to the tag. Because the tag knows key k, it can verify whether the hash result received from the reader is valid. Note that an unauthorized reader does not know the value of key k associate with the tag, and is not be able to compute $h(n_t, k)$. If the tag successfully authenticates the reader, the tag sets its privacy bit from zero to one, denoting the tag's traversal to a location that may contain unauthorized readers. The tag then computes $h(n_r, k)$ and sends the result back to the reader. The reader authenticates the tag by verifying the validity of the hash result $h(n_r, k)$ received from the tag. Figure 5.2 illustrates this checkout protocol.

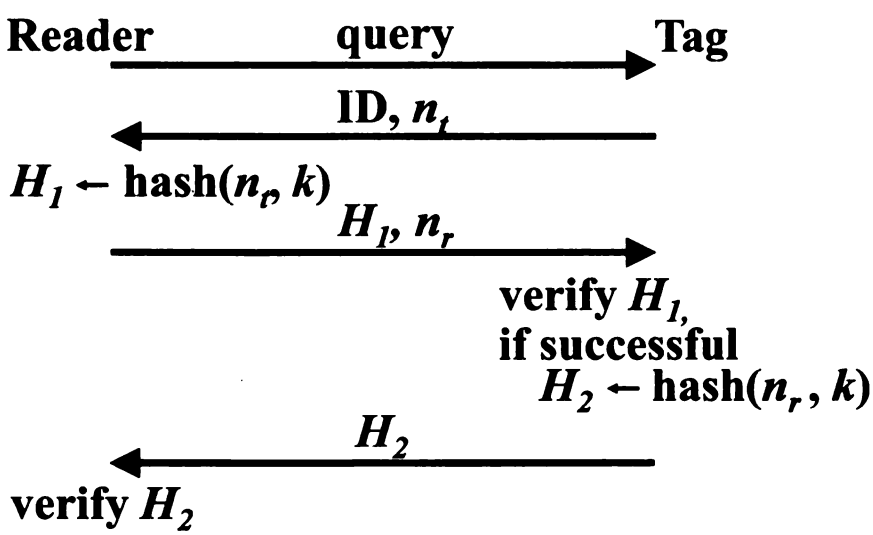

Figure 5.2: The checkout protocol

There are different levels of security obtained by using this checkout procedure. Only an authentic tag and an authorized reader would know the value of the secret k for a tag. Therefore, if a cloned tag does not contain the correct value for k, it would not send the correct hash result to the reader at the end of the procedure, causing

the reader fails to authenticate that tag. If the reader does not verify a tag within a time limit, the system will timeout and not allow the customer to finish the checkout procedure. The second level of security deals with the random nonce sent by both the reader and the tag in this process. In order to reduce the chances of a replay attack, random numbers are hashed along with the value of the secret key k.

5.4 Out-store protocol

The out-store protocol resembles a tag's behavior once it leaves the store. At this point, various readers with different levels of security are assumed to be able to access the tag. Therefore, only enough information about the tag is given to allow authenticated readers access in order to flip the tag's privacy bit back to zero, which includes the tag's generic name and a random nonce. Since the tag's generic name is represented by a number, an unauthorized reader will not know what items are currently being read. The next section will explain how the information being given above will allow an authenticated reader to turn the privacy bit of a tag back to zero. Figure 5.3 illustrates this checkout protocol.

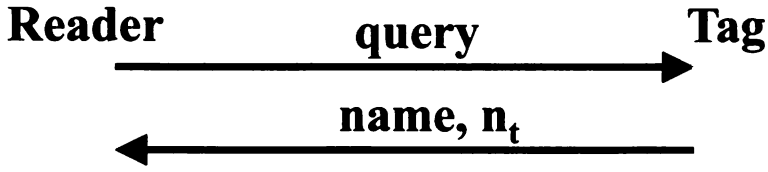

Figure 5.3: The out-store protocol

Note that there are many reasons that an attacker would want to retrieve a tag's private information. For example, an attacker may want to know what people shop for in certain stores to develop spam or other similar shopping techniques. Also, an attacker may want access to the private information of a tag to gather secrets about the product's producer or the store in general. This type of attack is prevented in

our out-store protocol.

5.5 Return protocol

The return protocol deals with the returning of an item to where it was sold. Many stores have returned items that they are still able to resell; therefore, these RFID tags need to be reset for resale. The return protocol requires mutual authentication between a tag and a reader as well. To prevent unauthorized readers from flipping the privacy bit of a tag from one to zero, the tag needs to authenticate the reader. Though this concept may appear clear, it may not be as easy to understand why the tag needs to be authenticated. If the tag were not authenticated, a person could create a counterfeit tag to indulge the price value of an item. This in turn would allow a customer to increase the price of an item, enabling them to receive a higher amount of money back or exchange the item for a higher valued one.

The return protocol works as follows. The first two steps are the same as the two steps in the out-store protocol. In the third step, the reader retrieves the secret key k of the tag from its back-end inventory database using the *name* received from the tag. The reader will then perform a one-way hash function on this k and the random nonce, n_t, received from the tag. The reader then generates its own random nonce, n_r, and sends it along with the hash result, $h(n_t, k)$, to the tag. Because the tag knows key k, it can verify whether the hash result received from the reader is valid. Note that an unauthorized reader does not know the value of key k associate with the tag, and is not be able to compute $h(n_t, k)$. If the tag successfully authenticates the reader, the tag sets its privacy bit from zero to one, denoting the tag's traversal to a location that may contain unauthorized readers. The tag then computes $h(n_r, k)$ and sends the result back to the reader. The reader authenticates the tag by verifying the validity of the hash result $h(n_r, k)$ received from the tag. Figure 5.4 illustrates this

checkout protocol.

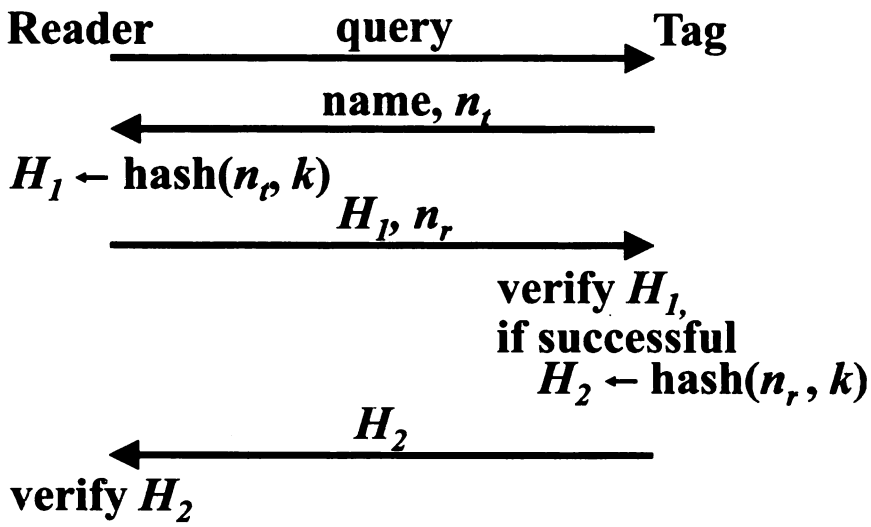

Figure 5.4: The return protocol

Chapter 6

HashTree

This chapter discusses the structure of the HashTree protocol. It first discusses the issues facing a normal static tree based authentication approach. The chapter will then give details of the proposed dynamic approach to improve on these issues.

Section 6.1 discusses the details of a static tree-based approach. Section 6.2 provides an overall look of the protocol. The next three sections discusses each function within the protocol. Section 6.3 goes over the intial details for setting up the reader and the tag. Section 6.4 discusses the actual security measures of the protocol. Lastly, Section 6.5 discusses what procedures will take place if additional tags needed to be added (or deleted) from the system.

6.1 Challenges of Static Tree Based Approaches

In order to understand the challenges a static tree based authentication approach [6] entails, one must first understand how such an approach works. Consider the tree in Figure 6.1. Each node contains a distinct key and each tag is denoted to each leaf node. Therefore, there exists a unique path of keys from the root to each leaf node, whereas these set of keys are assigned to each leaf node, which is used for authentication. For example, tag T_8 contains keys k_0, $k_{1,2}$, $k_{2,4}$, $k_{3,8}$. When the reader

R authenticates T_8, it first sends a nonce n to the tag. The tag T_8 then encrypts its set of keys with n (usually with the use of hash functions) and sends the result back to the tag. The reader then searches for the leaf node corresponding to the tag by repeatedly hashing the sent nonce n with its binary tree of keys and matching those results to the sent list of encrypted results from the reader. If such a path exists to a leaf node, T_8 is identified and reader R regards this tag as valid.

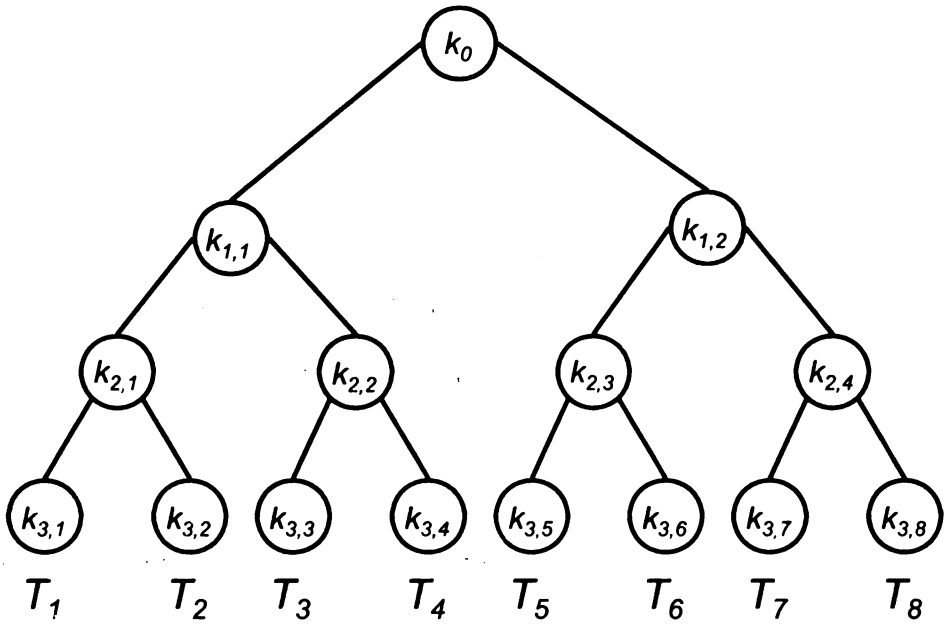

Figure 6.1: A static binary tree with eight tags

From the above procedure, we can tell that the path to a tag will end up sharing certain keys with other similar paths to different tags. For example, tags T_8 and T_7 share key $k_{2,4}$, and of course all tags will share the root key, k_0. A positive in using this static architecture is that its running time is logarithmic. For example, in Figure 6.1, any identification of a tag only needs $\log_2(8) = 3$ search steps. This structure, however, also presents a security flaw in that if a tag is compromised, an adversary would obtain multiple paths from the root node to a leaf node, including the keys on those paths. Since the keys of the architecture are never updated, the captured keys will still be used by the uncompromised tags, allowing further knowledge of what key

combinations an uncompromised tag may contain.

A practical solution to solving the above mentioned problem is to update the keys after every authentication; however, the aforementioned static tree architecture can not handle such a task due to its complexity. For example, if one were to update tag T_1, they would have to change keys k_0, $k_{1,1}$, $k_{2,1}$, and $k_{3,1}$ partially or totally. This in turn would single-handedly affect every path of tree, not allowing the other non-authenticated tags to update themselves in order to authenticate correctly with a reader. Therefore, using this architecture, every tag would need to be updated periodically and simultaneously (along with static tree used by the reader) to continue the synchronization of the approach. Unfortunately, this solution is not practical in large scale systems with hundreds or millions of tags. An alternative solution would be to collect only those tags effect by each authentication of another tag's path periodically; however, this idea would be more cumbersome than the first in trying to collect a number of tags only affected by one tag changing's it's keys. Therefore, we propose our HashTree protocol, a dynamic key-updating algorithm for private authentication in RFID systems. Our protocol is explained in the next few subsections.

6.2 Basic Design

Our HashTree protocol is compromised of three components: *system initialization, tag identification*, and *system maintenance*. The first two components are very similar to the static tree based approaches presented in [6] and performs the basic identification functions. However, unlike the previously described dynamic-key approach [18] in Section 2, our protocol only has one simple step in order to update the reader and the tag, and therefore doesn't need an additional step for its explanation. Lastly, the fourth component is used to direct the joining and disjoining of tags to and from the system.

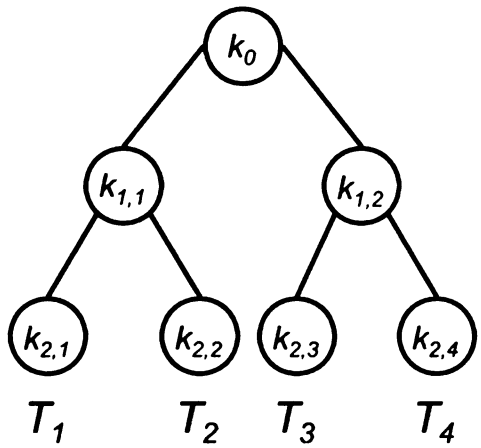

Figure 6.2: A small binary tree with four tags

6.3 System Initialization

To explain this process, consider a short and balanced binary tree as shown by an example in Figure 6.2. Assuming that there are N tags T_i (where $1 \leq i \leq N$) and a reader R in the RFID system, reader R will assign the N tags to N leaf nodes of the balanced binary tree S. Each node in S will be assigned with a key k. Initially, each key is randomly and independently generated by the reader.

Upon the introduction of a tag T_i into the system, the reader distributes an array of $(\lceil \log N \rceil + 1)$ to T_i. These keys correspond to the path from the root to tag T_i. For example consider Figure 6.2: the keys stored in tag T_2 are k_0, $k_1, 1$ and $k_2, 1$. However, these keys will not be stored in their current state. The reader R will also generate a random number (a nonce) n such that each key k will be used to generate a hash function result in the form $h(k, n)$. Therefore, the sequence of hash function results would be $(h(k_0, n), h(k_1, n), ..., h(k_d, n))$, where d denotes the keys distributed to T_i and depth of how many keys are within the tree. The nonce n generated by the reader will also be included within the tag. The usage of the hash function will be explained in the later portion of subsection 4.4.

6.4 Tag Identification

The tag identification procedure is compromised of a basic three round process. The first round consist of the reader R querying the tag, T_i to send its data to the reader for authentication. In the second round, the tag releases its information stored to the reader. This includes the nonce n generated from earlier and the hash functions results from the nonce and the list of keys given to the tag.

During the step between the second and third round is when the actual authentication of tag T_i takes place. In short, the reader begins to mimic its action in the initialization process by performing a hash function with the root key k_0 and the nonce given by the tag. The reader then begins to authenticate the tag by checking the first hash function result $h(n, k_0)$ with the one generated by the root note. Assuming the hashed result of this function matches the previously generated one, the reader continues to authenticate the tag in a recursive like manner. At this point, each key k from the children of the root node would be used to generate another hash function with the given nonce n. Once one of these function results are matched, the reader may know what subtree to continue its recursive path along the tree until the authentication procedure is complete. For example, consider tag T_3 from Figure 6.2. The reader would need to perform a hash function using keys k_0, $k_1, 2$ and $k_2, 3$ in combination with the nonce given by the tag in order to successfully authenticate that tag. Upon authentication of a tag, a signal is sent to the back-end database connected to the reader where as it will retrieve the tag's EPC [7] and proceed perform whatever calculations are necessary for that reader (i.e. unlock a secure door, show a person's registered information, etc.).

As stated in subsection 4.2, the method of updating our tags is simple enough to be held within our identification procedure, which is what our third round is compromised of. After authentication of the tag, only the tag's keys will be updated, not the reader's. To achieve this, the reader will generate another random nonce

n_2 different from the one given by the tag. This nonce will be then be used by the reader to perform another set of hash functions in accordance with the tag's T_i keys as recorded in the reader key tree. The new list of hash function results and the new nonce n_2 generated will then sent to the tag, thereby updating its key values.

Before explaining the system maintenance portion of this section, let's take a moment and explain the theory behind only updating the tag's values and not the reader's. The key behind paper [18] is that it assumes an attacker may gain access to the secret values of a tag, therefore compromising the reader's key tree system since each key within the static tree [6] paper connected to some other tag's set of keys. Our algorithm handles this compromisation in several ways. First, by updating a tag's set of key values, it can no longer be traced to other tags held within the secure holding area. For example, if this protocol were used for supply-chain management within a retail store, all tags bought from the customer could not be traced to items held within the store. More importantly, the process in which we assign the tag initially and after authentication (i.e. using hash results instead of original key data) blocks the attacker from knowing the actual key values held within the key tree of the reader. This is one of the main reasons we do not have to update the key tree of the reader. The other reason deals with the theory behind two key locations not being associated with each other. To explain this, consider tag's T_1 and T_2 in Figure 6.2. Both tag's in this tree structure share the key $k_1, 1$. In a static tree, compromising one of these tags would compromise the secret of the other since each key is given to the tag in its normal form. However, our algorithm produces a different hash result of this value for both tags since each tag would have a different nonce (i.e. $h(n_1, k_{1,1}) \neq h(n_2, k_{1,1})$); therefore, the attacker would have no way of relating these two values together.

6.5 System Maintenance

In a real-time execution of this protocol, users may need to simultaneously remove and add additional tags to the key tree. Therefore we have enabled an easy way of performing such a task.

Let's assume a new tag T_i needs to connect to the current tree, S. This task may be performed in one of the two ways. First, the reader R will search each leaf node of the current tree. If there is an empty space available, T_i will be placed there. This space may have been made available due to the process in which a node is deleted from the tree. If a tag T_i needs to be deleted from the system, its data (including its EPC and any other system information needed) will be erased from the current leaf node pointer to that tag, leaving the above mentioned space. However, if all the leaf nodes are pointing to a current tag, a new subtree will have to be established. This subtree will be made in the same fashion in which the entire tree is made as described in the subsection 4.2. Each node (except for the root node) will be assigned a randomly generated key and the tag T_i will be assigned these values with a random nonce as described in the fashion before. An example of the addition process can be found in Figure 6.3.

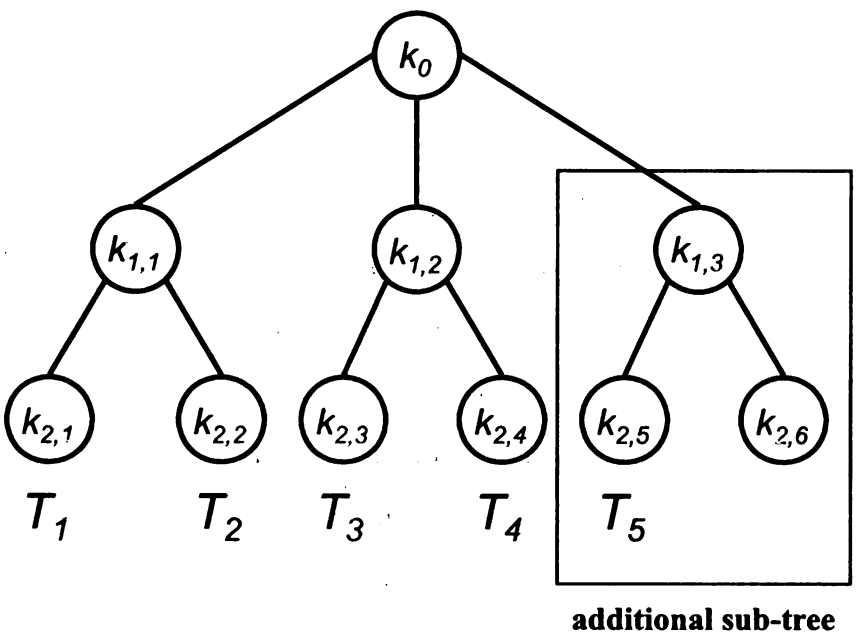

Figure 6.3: Example of an additional subtree

Chapter 7

Security Analysis

Due to lack of resources, a thorough examination of each of the previously described protocols was not able to be performed upon actual RFID readers and tags. However, due to the current cost of a reader, many papers [6, 18] before this thesis have been in this current situation and have established a widely accepted list of security requirements for which most RFID protocols should abide to. This chapter goes through this list, discussing the pros and cons of each protocol for each requirement.

7.1 Privacy

This requirement states that any private information of tag, such as a product's EPC code or the name of a person from an identification badge, should not be disclosed to a third-party during authentication. As far as the RFIDGuard and the PAP protocols are concerned, only the checkout and return protocols will need to be considered, since the in-store and out-store protocols are just part of the former ones. In both return protocols, the ID of a product is not returned until a reader has authenticated itself, thus adhering to the requirement. The checkout protocol on the other hand releases the ID before authentication; however, as previously stated, it is assumed a protection device will be placed near all exits of a store when executing

this protocol, thus protecting it from third-party readers until a tag's privacy bit is turned on.

The HashTree protocol is different case. You may have noticed that no private information was discussed within the protocol itself. This is because the protocol was made general enough to implement a numerous amount of ways. Without going into too much detail upon those numbers, any private information in the current form of the protocol would be contained within the back-end of the database, linked to the leaf node that is associated to a tag. Therefore, when a tag is authenticated, the reader will know what type of information is needed for that item or person(i.e. when authenticating a person to enter a building or lab).

7.2 Cloning resistance

This requirement states that an attacker should be impersonate a valid tag in any form. This also includes performing what is known as the *replay* attack, where an attacker takes a reply from a valid tag and replays it to a reader in order to authenticate itself. The replay attack in the case of both RFIDGuard and PAP would consider scanning the tag outside the contents of a store, which in this case the tag is protected since the only information given to any reader will only allow an authorized reader to unlock the tag for its information. The HashTree protocol however is subtle to this attack. Since its private information is assumed to be within the back-end database of the reader, it only gives its hashed results and the pre-generated randomized number in order to hash the path of keys within the tree correlating to the tag, which any third-party attacker can access if they were to scan an unprotected tag.

7.3 Untraceability

The requirement states that a tag should be able to be correlated with its output authentication messages; otherwise, that tag may be tracked by attackers. In order to understand why the RFIDGuard and PAP protocols are not affected by this security requirement, one must understand the meaning of a tag's generic *name*. This value represent the product's type, not that individual item itself. For example, in the retail envoirment, the generic name for a product would have a number presenting all 20oz Pepsi products, not an individual bottle of Pepsi itself. If this were not true, not only would these protocols be subjected to the above security requirement, but the number of pseudonyms (or keys) needed for storage in the back-end database would be ambigious (i.e. establishing pseduonyms/keys for items that are less likely to be returned). The HashTree protocol however is subtle to this, but the updating of the tag lessens its degree. As stated before, this protocol is meant for a more secure tag (i.e. identification badges, etc), so one would exepect the tag to be used daily. Since the tag is updated so often, it would be difficult to track a tag over a long time.

7.4 Forward secrecy

This requirement assumes that an attacker may copromise of a tag; therefore, your tag should not reveal any previous outputs from an earlier authentication session. By looking at the protocol of the HashTree, it should obvious to tell that it does not suffer from this requirement, seeing as how each tag's contents is updated after every authentication session. During the research for RFIDGuard and PAP however, this requirement was not considered, and therefore is subtile to this attack. However, the pseudonym is rotated upon each scan, so the tag will not have the same exact experience as before.

7.5 Compromising resistance

This requirements also assumes that an attacker may gain the contents of a tag; however, it suggest that the contents of one tag should not correlate with the contents of another. This flaw can be clearly seen in a static tree-based tag and is exactly what the HashTree based tag was designed for. Since the hash result of a tag's original set of keys is built in the tag beforehand, a compromised tag will not relay any secrets from the reader's tree nor will it correlate to hash results from any keys shared with another tag. Per the last requirement, the RFIDGuard and PAP protocols were developed without this requirement being taken into consideration, therefore, they are also subtjected. However, since each set of psuedonyms (or a key) deals with a product type, the RFID system will not be as breached as possible. For example, if a tag with a product's line of clothing material is compromised, only that line of clothing will be compromised, not the entire store

Chapter 8

Conclusion

RFID is a promising technology that can revolutionize the way we lead our lives. However, before this becomes a reality, certain security issues like consumer privacy protection and fraud prevention and detection must be addressed. This thesis provides three different protocols to combat these issues: RFIDGuard, PAP, and HashTree, which all run differently in their own respects. This is due to RFIDGuard and PAP being more for a retail enviornment and HashTree being developed in a more broader perspective. Despite the flaws mentioned within the security analysis, each protocol has the potential of securing a large RFID system with further research. There is an ample scope in the field of RFID security to improve and innovate in order to allow RFID technology to be incorporated into our daily lives; the steps followed in this thesis is only the continuation of many steps to come.

Bibliography

[1] D. Molnar A. Juels and D. Wagner. Security and privacy issues in e-passports. In *IEEE/Create Net Secure Communications*, 2005.

[2] P.C. van Oorschot A.J. Menezes and S.A. Vanstone. Handbook of applied cryptography. *CRC Press*, 1996.

[3] G. Barber, E. Tsibertzopoulos, and B.A. Hamilton. An analysis of using epcglobal class-1 generation-2 rfid technology for wireless asset management. In *Military Communications COnference*, volume 1, pages 245–251, October 2005.

[4] S. Bono, M. Green, A. Stubblefield, A. Juels, A. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In *USENIX Security Symposium*, pages 1–16, Baltimore, Maryland, USA, July-August 2005. USENIX.

[5] A. Soppera D. Molnar and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of rfid tags. *Cryptology ePrint Archive, Report*, 2005/315, 2005.

[6] Tassos Dimitriou. A secure and efficient rfid protocol that could make big brother (partially) obsolete. In *PERCOM '06: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 269–275, Washington, DC, USA, 2006. IEEE Computer Society.

[7] EPCglobal. Epcglobal website. http://www.EPCglobalinc.org/, 2007.

[8] C. Floerkemeier, R. Schneider, and M. Langheinrich. Scanning with a purpose: Supporting the fair information principles in rfid protocols. In *Proceedings of the Second International Symposium on Ubiquitous Computing Systems*, 2004.

[9] International Civil Aviation Organization ICAO. Document 9303, machine readable travel documents (mrtd), part i. *Machine readable passports*, 2005.

[10] International Civil Aviation Organization ICAO. Document 9303, machine readable travel documents (mrtd), part i. *Machine readable passports*, 2005.

[11] A. Juels. Minimalist cryptography for low-cost rfid tags. *Proc. 4th Int. Conf. Security Commun. Netw.*, 3352:149–164, 2004.

[12] A. Juels. Rfid security and privacy: A research survey. *IEEE Journals on Selected Areas in Communications*, 24(2):381–394, 2006.

[13] A. Juels and J. Brainard. Soft blocking: Flexible blocker tags on the cheap. *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 1–7, 2004.

[14] A. Juels, D. Molnar, and D. Wagner. Security and privacy issues in e-passports. In *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm)*, pages 74–88, September 2005.

[15] A. Juels, R. L. Rivest, and M. Szydlo. The blocker tag: Selective blocking of rfid tags for consumer privacy. *In Proceedings of the 10th ACM conference on Computer and communication security*, pages 103–111, 2003.

[16] Ari Juels, Paul Syverson, and Dan Bailey. High-power proxies for enhancing RFID privacy and utility. In *Workshop on Privacy Enhancing Technologies - PET 2005*, Dubrovnik, Croatia, May-June 2005.

[17] T. Li and R. Deng. Vulnerability analysis of emap-an efficient rfid mutual authentication protocol. *International Conference on Availability, Reliability and Security*, 2007.

[18] Li Lu, Jinsong Han, Lei Hu, Yunhao Liu, and Lionel M. Ni. Dynamic key-updating: Privacy-preserving authentication for rfid systems. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 13–22, Washington, DC, USA, 2007. IEEE Computer Society.

[19] B. Crispo M. Rieback and A. Tanenbaum. Rfid guardian: A battery-powered mobile device for rfid privacy management. In C. Boyd and Eds. J. M. Gonzlez Nieto, editors, *Proc. Australasian Conf. Inf. Security and Privacy*, volume 3574, page 184194. Springer-Verlag, 2005.

[20] D. Molnar and D. Wagner. Privacy and security in library rfid: Issues, practices, and architectures. In B. Pfitzmann and P. McDaniel, editors, *Proc. ACM Conf. Commun. Comput. Security*, pages 210–219, 2004.

[21] D. Molnar and D. 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*, page 210219, New York, NY, USA, 2004. ACM.

[22] M. Ohkubo, K. Suzuki, and S. Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.

[23] M. Rieback, B. Crispo, and A. Tanenbaum. Rfid guardian: A battery-powered mobile device for rfid privacy management. *Proc. Australasian Conf. Inf. Security and Privacy*, 3574:184–194, 2005.

[24] S.E. Sarma. Towards the five-cent tag. http://www.autoidlabs.org/, MIT-AUTOID-WH-006, 2001.

[25] D.W. Engels S.E. Sarma, S.A. Weis. Rfid systems and security and privacy implications. In LNCS, editor, *Workshop on Cryptographic Hardware and Embedded Systems*, volume 2523, page 454 469, 2003.

[26] S. Stern. Security trumps privacy. *Christian Science Monitor*, 2001.

[27] 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.

[28] Johannes Wolkerstorfer. Is elliptic-curve cryptography suitable to secure RFID tags? Handout of the Ecrypt Workshop on RFID and Lightweight Crypto, July 2005.