Next Generation RFID Randomization Protocol

by

Jason LaValley

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Electrical Engineering
University of Ottawa

Abstract

Radio Frequency IDentification (RFID) is a wireless communications technology which allows companies to secure their assets and increase the portability of information. This research was motivated by the increased commercial use of RFID technology. Existing security protocols with high levels of security have high computation requirements, and less intensive protocols can allow a tag to be tracked. The techniques proposed in this thesis result in the increase of ciphertexts available without a significant increase in processing power or storage requirements. The addition of random inputs to the generation of ciphertexts will increase the number of possible results without requiring a more advanced encryption algorithm or an increased number of stored encryption keys. Four methods of altering the plaintext/ciphertext pair (random block, set pattern, random pattern, and indexed placement) are analyzed to determine the effectiveness of each method. The number of ciphertexts generated, generation time, and generation errors were recorded to determine which of the four proposed methods would be the most beneficial in a RFID system. The comparison of these method characteristics determined that the set pattern placement method provided the best solution. The thesis also discusses how RFID transmissions appear to attackers and explains how the random inputs reduce effectiveness of current system attacks. In addition to improving the anonymity of RFID tag transmissions, the concept of authenticating random inputs is also introduced in this thesis. These methods help prevent an adversary from easily associating a tag with its transmissions, thus increasing the security of the RFID system.

Acknowledgements

I would like to acknowledge those who have encouraged and supported me in completing my thesis and my Master's degree.

I would like to thank my supervisor Dr. Tet Yeap, from the School of Information Technology and Engineering at the University of Ottawa, for the support and guidance he has given me during the development of my thesis.

I would like to thank Dr. Hussein Mouftah, Mr. Peter MacKinnon, and the WiSense research group for giving me the opportunity to present my work and showing an interest in my research topic.

I would also like to thank my family and friends for their understanding and encouragement while I worked on this thesis.

Contents

List of Tables

List of Figures

List of Acronyms

RFID Radio Frequency IDentification

EPC Electronic Product Code

XOR exclusive OR

GPS Global Positioning System

TID Tag IDentifier

ID Identification

AES Advanced Encryption Standard

DES Data Encryption Standard

SHA Secure Hash Algorithm

MD Message Digest

FPGA Field-Programmable Gate Array

ASIC Application-Specific Integrated Circuit

DoS Denial of Service

Chapter 1

Introduction

1.1 What is Radio Frequency Identification?

Radio Frequency Identification, also known as RFID, is a system that allows the transmission of data between two points using radio frequencies. The system consists of tags, which transmit data upon request, and receiving stations which request that information on behalf of an application. These tags are commonly used for personal identification, club cards, or product tracking. A standard for information stored on RFID tags is the Electronic Product Code (EPC) set by EPCglobal, which contains data such as the manufacturer, stages of manufacturing/delivery, and expiry dates[24]. Additional information can also be stored on the tags such as authorization codes, identifiers, and balances when they are used as identification or club cards.

RFID tags can be grouped into passive, semi-passive/semi-active, or active categories. Passive tags solely use the power transmitted by the receiver to respond with its data, limiting the tag to basic functions. Semi-passive and active RFID tags can use the transmitted power as well as power from additional sources, such as power generated by solar or piezo-electric components, or stored in batteries. Tags within these categories usually contain microcontrollers that can provide additional functions and allow the tag to be reprogrammed. The categorization of RFID tags can also help to identify the range capability of individual tags. Passive tags can have a maximum range of up to three meters, while active tags have a range of 100 meters or more[38].

RFID receivers can range from small hand held readers to networked wall mounted stations. These receivers can be used on their own or with middleware which combines the readers into a meshed network. This network of receivers can be used to determine the location of the tags being read. Factors such as signal amplitudes and direction can be used to determine the general location of the tag. Materials between the tag and receiver can affect the way the signal is received, such as reducing the received signal power or reflecting the signal at an odd angle. Similar to GPS positioning, coordinates can be more accurate with a greater number of receivers reading the unique tag identifier.

When requested, the information stored on the tag can be encrypted prior to transmission. The complexity of the encryption can depend on the computation capabilities of the individual tags and the number of encryption keys stored within the tag's memory. The simplest form of encryption is to store the data on the tag in its encrypted state. Tags which provide its own encryption algorithm can also allow for the data to be changed or for new encryption algorithms to be used. To conserve power and response time, the most common encryption output is one ciphertext per encryption key.

1.2 Common Uses of Radio Frequency Identification

RFID tags and readers are commonly used to identify personnel and assets within a company. Some countries, such as Canada and the United States of America, are also introducing drivers licenses[32] and passports[32][33] which have RFID tags embedded within the identification. Governments are introducing RFID tags within official documents to make duplication more difficult for criminals. This step makes copying a passport with a new picture more difficult as the RFID tag embedded within will still have the original data stored on it. Companies can also use RFID tags to identify employees[28]. Gateways can be used to control locks on doors or to provide a passive security system. Rather than track an employee's movement, these gateways will replace the need for keys or keypads when accessing parts of the building restricted to the public.

When tracking company assets, such as store stock or warehouse shipments, the level of data required dictates the placement and the required number of RFID receivers. The most basic location data would be represented by a boolean response to the question "is the product in the building?". This basic level of information can be satisfied by placing receivers at the entrances to the warehouse. The placement of multiple receivers at each gateway will also allow the system to understand the direction that the tagged item is

traveling. This setup could also be used at loading bays to confirm the shipping manifests as the product is moved in and out of the warehouse. The next level would be the asset's location within the warehouse, which itself can be broken into many sub-levels ranging from coarse location to fine location. Coarse location levels could be satisfied by adding additional gateways along the aisles of the warehouse to split the warehouse into sections. Fine location levels would require a grid of receivers to be installed. The direction and power of the received transmission will narrow down the possible locations of the tag. System control can become more complex as the number of receivers increase within the same reading radius. Fine location also requires the control system to activate and deactivate receivers to avoid interference, thus requiring each tag to transmit multiple times to acquire a position. An example of this system would be a hospital monitoring the position of patients and staff to provide better service as well as the position of medicines and sensitive equipment to prevent theft[28].

The previous example used receivers (with known positions) to locate RFID tags (with unknown positions), the opposite can also be used to determine positioning. Examples of these positioning systems include navigation/wayfinding/proximity[40][34][31][27] and location based logistics[41] systems. These applications used grids of RFID tags (with known positions) to help navigate and collect local environmental data. This form of positioning is extremely useful when used in environments where GPS signals cannot reach, such as underground and within large buildings.

1.3 The current problem with Radio Frequency Identification security

While the information being transmitted is encrypted, adversaries eavesdropping on the wireless communications can still gain information about the tags. It is common for RFID tags to transmit the same ciphertext when it sends information to the receiver. That ciphertext can be tracked and this information can lead to assumptions about stock or personnel patterns depending on the application. The reason why most tags do not produce more than one ciphertext is cost. In a market where the cheapest tag is bought, manufacturers do not want to include randomized encryption algorithms, or additional encryption keys as it will increase the cost of the individual components used.

An example of an eavesdropping attack would be an adversary trying to gain access

to an office within a secure building. The attacker could record the ciphertexts of employees entering the building without coming into direct contact. What is done with this information, depends on the security system. If the system has a method of authenticating the tag (i.e. checking whether or not the transmission is from a valid tag or a copy), the adversary can use the information to determine when the building is empty, and depending on the number of eavesdropping attacks, what parts of the building have the fewest people in them at specific times. The break in will be easier for the attacker if the system is of a simpler design which does not authenticate transmissions. This means that the person can simply replay one of the recorded ciphertexts to gain entry to the building.

A solution to this problem would be to allow the tag to generate or store multiple ciphertexts which can be tracked by the security system. This would make it harder to identify tags by their ciphertexts and add another layer of security by tracking which ciphertexts are used to prevent duplication. As mentioned above, the cost of implementing this on the tag is too high to be considered by profit driven manufacturers. This leads to the motivation for the research done in this paper.

1.4 Motivation

Tag security could be increased by disassociating the tag's identity from the ciphertext it sends by increasing the number of ciphertexts it can generate. The ideal way to do this would not require additional components or the replacement of components with greater computational or storage capacities.

Some skeptics may say the current system is less secure than a physical key since the key cannot be stolen without direct access by the attacker. Improving the security of the tag without increasing its cost will allow the trust and adoption of these technologies to rise. Once RFID's current applications are satisfied, this research could be implemented in future products.

1.5 Key Results

A method of plaintext modification was developed to provide a large number of modified plaintexts prior to encryption. This allowed a simple 1:1 encryption algorithm to be used while still generating multiple ciphertexts. The new security protocol helps to protect the encryption key used as well as disassociate the RFID key from its transmissions. This anonymity increases the overall security of the RFID network.

1.6 Main contribution

This research has developed a method which helps to avoid patterns in data transmissions between Radio Frequency IDentification (RFID) tags and receivers. The method, which involves removable randomly generated fill, helps to disassociate a single tag from its transmissions.

The motivation for this research was to develop a communication protocol that does not significantly affect the RFID tag's storage or processing requirements. This would allow the protocol to be added to existing tags which can be reprogrammed.

The main objective of the research is to determine whether or not string manipulation sufficiently hides RFID tag associations in combination with a pre-existing single key encryption algorithm when compared to tags which use multiple encryption keys. The evaluation of this research was done from the perspective of hindering eavesdropping attacks.

The first stage, seen in Chapter 3, in developing this protocol was to determine the best method in generating the highest number of recoverable randomizations with the least amount of input. This required the comparison of random number generation and placement techniques.

The second stage, seen in Chapter 4, created a simulated prototype of the top 3 reasonable randomization methods. The evaluation of these methods required a simulation of RFID communications between tags and receivers, which was also programmed for this research. Two of the main comparators were the time it took to complete a transmission, as well as the message similarities over a series of transmissions.

The third and final stage, seen in Chapter 5, created a firmware prototype which ran the best performing method, as determined in the second stage.

Chapter 2

Background and analysis of previous work

2.1 The Radio Frequency Identification tag

The Radio Frequency Identification tag, in conjunction with one or a series of receivers, allows a system to be created which tracks and protects assets and personnel or to maintain the security of information. Examples of asset or personnel protection include building security or inventory maintenance on behalf of a corporation. Examples of information protection include embedded passports or licenses on behalf of governments. There are several components and processes within the RFID tag architecture which can affect how it will react to any change proposed by this research. This section provides background material on the message being randomized and the processes which the message then relies on prior to transmission.

2.1.1 RFID Classes

The RFID tag can be split into multiple components which include an antenna, processing unit, and power storage. Additional components can be added to the tag to satisfy specialized applications. The antenna is used to gather power and communicate with a receiver using standardized commands over set frequencies. The most common frequencies used with RFID tags are 13.56 Mhz[1] and 860-960 MHz[23]. Depending on the application, the processing can either be completed by the transistors directly on the tag or by an attached microcontroller. The power source is dependent on two

factors, the processing power required and the ideal range of transmission. Low powered applications use the power gathered from the receiver's transmissions, these tags are called passive RFID tags. Active RFID tags require an independent power source, such as a battery, to satisfy higher power demands such as microcontrollers or transmitting over great distances. These characteristics can be used to separate RFID tags into four classes: identity tags, higher functionality tags, battery assisted / semi-passive tags, and active tags[22].

2.1.2 What is written on a RFID tag?

The message transmitted by the RFID tag can contain several pieces of information. This information can range from the tag's identifier, the information related to what the tag is attached to, and optional data fields which can be used in specialty applications. The following subsections will go into further detail on these different categories of information represented within the message.

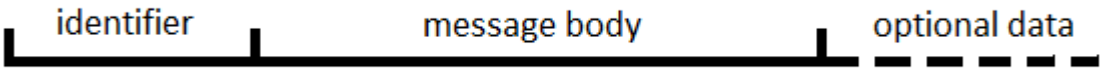

Figure 2.1: Message format

RFID tag identifier

Each RFID tag has a unique serial number assigned to it by the manufacturer called the Tag Identifier (TID)[23]. The standards set by EPCglobal allow the manufacturer to choose one of two TID formats, known as $E0_h$ and $E2_h$[23]. The $E0_h$ TID contains an 8 bit manufacturer identifier and a 48 bit tag serial number; the $E2_h$ TID contains a 12 bit tag mask-designer identifier and a 12 bit tag model number[23].

Main body message

While each tag is programmed with a unique identifier, that value is unique only within the local environment in which it was produced. An additional value known as the EPC URI is the preferred system to refer to physical objects as it is a ratified standard

which guarantees global uniqueness[24]. If this tag is used in a security application, the information would be used to identify the person assigned that tag. Data fields such as name, authentication codes, and permissions could be included within the message. This provides the application the information it needs to determine whether or not the tag holder is allowed to access certain information or physical locations. If the tag is used in an inventory application, the information would be used to identify product information such as the manufacturer, date of construction, and/or serial numbers. The majority of this information has been standardized in the form of the electronic product code, which is set by EPCglobal.

The data stored within the EPC depends on the tag's use, such as location tracking or identification cards. These different uses result in nine methods of EPC generation, which are summarized in the following table.

EPC Scheme	use	format
sgtin	trade item	urn:epc:id:sgtin:CompanyPrefix.ItemReference.SerialNumber
sscc	logistics unit	urn:epc:id:sscc:CompanyPrefix.SerialReference
sgln	location	urn:epc:id:sgln:CompanyPrefix.LocationReference.Extension
grai	returnable asset	urn:epc:id:grai:CompanyPrefix.AssetType.SerialNumber
giai	fixed asset	urn:epc:id:giai:CompanyPrefix.IndividulAssetReference
gdti	document	urn:epc:id:gsrn:CompanyPrefix.ServiceReference
gsrn	loyalty card	urn:epc:id:gdti:CompanyPrefix.DocumentType.SerialNumber
gid	unspecified	urn:epc:id:gid:ManagerNumber.ObjectClass.SerialNumber
usdod	USDOD supply chain	urn:epc:id:usdod:CAGEOrDODAAC.SerialNumber

Table 2.1: Description of EPC schemes[24]

The formats listed in the previous table help to provide a string representation of the data related to physical items, however this information must be placed within the memory of the RFID tag. If the data remains in its string form, the memory requirements, in addition to cost, will increase. Altering this string to a binary representation will help to minimize the space required on an individual tag. Each format has at least one binary coding scheme where its binary result can vary in length. To choose a scheme,

the manufacturer must consider the cost of memory as well as the number of tags used since the difference in binary length is a result of limiting the range of the data fields[24]. Table 2.2 lists the possible binary lengths of each EPC format.

EPC Scheme	binary length(s)
sgtin	96, 198
sscc	96
sgln	96, 195
grai	96, 170
giai	96, 202
gdti	96, 113
gsrn	96
gid	96
usdod	96

Table 2.2: Description of EPC scheme length[24]

The different EPC schemes each have unique methods of distributing its information over the 96+ bits. The first eight bits of any EPC is a header to aid a device or system in determining which method was used[24]. The remaining bits are reserved in segments for identifying filters used or any additional data related to each scheme[24]. A 96 bit Serialized Global Trade Item Number (sgtin) will be used to demonstrate the data fields represented by the EPC. The Global Trade Item Number is made up of subsections of varying length where the partition is used to identify the split between the company identifier prefix and item reference.[24] Each EPC scheme will have its unique segmentation of bits as well as distribution patterns within segments of varying length. Since the purpose of this research is to determine a universal method of message randomization, the additional EPC schemes will not be described in further detail within this report.

Optional message components

The previous subsection mentioned the two common uses of RFID tags, security and inventory, however some RFID tags have been used in alternative/research-based applications. One of these applications require tags to be manufactured such that sensors can be attached to its microcontroller.[30][16][8] This transforms the tag into a remote sensor platform which gives researchers a tool to collect data from a variety of locations without needing pre-existing communications infrastructure. The data collected is added to the

end of the message string prior to the modification stage. An example of this application would be to include a temperature sensor on a tag which identifies temperature sensitive products. This would allow the manufacturer to track the conditions of the product's local environment (immediate area) rather than the conditions of the global environment (building data).

2.1.3 Communication timing requirements

In addition to standardizing the message material, the EPC standard also sets limits and restrictions on the communication patterns between a RFID tag and receiver; an important example of these restrictions is the timing limit between transmissions. Since the randomization of any tag message will increase the tag's processing time, the time restrictions will also restrict the complexity of the randomization protocol.

The most important time period, from the perspective of tag security, is the amount of time the tag has to respond to a tag's query for information[23]. In addition to standardizing legitimate tag responses, set time periods between transmissions can also reduce the impact of those attackers attempting to hijack the communication. One attack which is affected by this is one where the attacker acts as an intermediary between a receiver and a RFID tag, allowing access to adversary as it mimics a legitimate tag. Set times reduces the attacker's chance of transmitting, receiving, and rebroadcasting tag responses to reader challenges.

Another time constraint set by the tag is the available encryption time. The example set by Feldhofer and Wolkerstorfer[13], shows how AES encryption hardware has a maximum of 25 milliseconds to perform a 128 bit block AES encryption on a UHF tag. To reduce the strain on the encryption hardware, the researchers proposed an interleaved communication scheme, which would ask for a tag's authentication and return after a set time to collect the information[13]. This frees the receiver to initiate the authentication process with additional tags. The delay in authentication can be set to be as long as necessary, however it is believed that a delay of "tens of milliseconds" is appropriate[13].

2.1.4 Random number generator

Any EPC compliant RFID tag will already contain a random or pseudo-random number generator to satisfy the EPC standards set by EPCglobal. This generator is currently used with the anti-collision schemes prior to direct communication.[23] The minimum requirements of this generator are as follows[23]:

- RFID tag contains a 16 bit random or pseudo-random number generator
- The probability that any number generated is greater than 0.8/65536 and less than 1.25/65536
- The probability that any two tags out of a pool of 10000 will generate the same number is less than 0.1 percent
- The probability that the number generated can be predicted is less than 0.025 percent when the previously generated values are known

Random number generators can be developed in either the software or as a piece of hardware[29]. Those developed within the software of the tag are usually considered to be pseudo-random number generators while hardware implementations (using naturally occurring randomness such as atmospheric or thermal noise) are considered to be true-random number generators[29]. Several versions of true random number generators have been developed as integrated circuits [19][37][18][4] and involve the use of chaotic signals or physical noise. The paper written by Balachandran and Barnett[4] describes the three common sources of the true-random number generator:

- "direct amplification of noise using a wideband high-gain amplifier or a regenerative latch"
- "sampling of a high-frequency oscillator with a low-frequency jittery oscillator"
- "discrete time chaotic systems using analog signal processing techniques whose output is spectrally flat and the numbers are uniformly distributed"

In addition to true-randomness, another benefit to the hardware random number generators is the lower voltage required to operate the generator [37] when compared to the voltage required to operate a microcontroller. This makes it more feasible (in terms of

power efficiency) to implement this design on a passive tag. While true-random number generators require less power compared to pseudo-random number generators they are also slower[2]. The reduced speed of the true-random number generator may affect a tag's ability to generate enough random numbers on demand. This would result in the tag generating and storing the random numbers before communication with a receiver is initiated, thus increasing memory requirements.

The paper written by Liu, Huang, and Zhu [29] proposes that it is "unrealistic to configure the hardware devices for every user who have to generate random numbers". Pseudo-random number generators follow an algorithm that will eventually repeat itself, Liu et al. proposes that this method provides suitable randomness if the period of non-repetition is far greater than the period that transmissions are collected[29]. This means that a pseudo-random number generator could be used as long as it met the previously mentioned requirements[23].

2.1.5 Encryption algorithms

To protect the information transmitted by the RFID tag, encryption is used to mask the actual message. Terminology used with the encryption of data includes the plaintext, ciphertext, and secret key. The plaintext is the true value of the data stored on the tag which is encrypted using an encryption algorithm, resulting in the masked value known as the ciphertext. Many algorithms use a parameter known as a secret key, a value which is used to encrypt and decrypt the messages sent between the reader and tag. To operate the most secure encryption algorithms, the length of the secret key is required to be the same length as the plaintext. The following sections will discuss the difference between symmetric and asymmetric encryption as well as describe cipher based and hash function based encryption.

Symmetric vs Asymmetric vs Hybrid

Encryption algorithms can be broadly split into two categories, symmetric and asymmetric encryption[9]. Symmetric encryption involves the use of a secret value, known as the key, which is shared between all legitimate parties of the encryption/decryption process[9][39]. This key is used to encrypt and later decrypt the transmitted data[9]. If this key was to be stolen or generated independently by an attacker, the system is no longer secure. Asymmetric encryption involves the use of both public and private

keys[9][3]. Asymmetric algorithms work in such a manner that a message can be encrypted using the public key and later decrypted by the private key[9]. These algorithms work in such a manner that the private key cannot be derived from the public key[9]. This also solves the symmetric encryption's problem of key distribution as well as increases the overall security. The disadvantage to the asymmetric algorithm is the length of time it takes to encrypt a message[9].

Since the asymmetric algorithms can take longer to complete than symmetric algorithms, hybrid systems have been proposed to utilize the benefits of each encryption type[10]. Hybrid systems use the security of asymmetric encryption to distribute randomized keys to all legitimate parties to the encryption/decryption process[15]. Those parties then use the symmetric key and faster algorithms to encrypt any further messages[9]. This method seems especially beneficial to RFID tags when considering the short period of time available to encrypt data prior to transmission.

Ciphers

The cipher is a form of encryption which combines the plaintext with a secret key such that the mathematical algorithm produces a ciphertext which can only be reverted back to the plaintext with the same secret key[9]. Encryption ciphers can be broken into two categories, block ciphers and stream ciphers. The major difference between the two cipher categories is that stream ciphers operate with time-varying transformations on each separate bit of the plaintext and block ciphers operate on large blocks of the plaintext with a fixed transformation[35].

The simplest form of stream ciphers is the exclusive OR (XOR) logic function. This function acts a logic gate comparing two streams of data: the plaintext and the secret key. The resulting ciphertext bit follows the logic function described below:

Plaintext bit	Key bit	Ciphertext bit
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.3: XOR logic output

The order in which separate streams are run through the XOR function is irrelevant since the operation is associative. An additional property of the XOR function is the fact

that any stream which XOR's itself will create a stream of zeros. These two properties result in the fact that the encryption process is identical to the decryption process, operating the XOR function with the secret key and the plaintext or ciphertext (depending on encryption/decryption). This simplicity in design allows for a low cost encryption process with a short duration, however it also allows the ciphertext to be vulnerable to a known plaintext attack. If the attacker knows which plaintext is going to be transmitted, the same process which allows the reader to decrypt the ciphertext will allow the attacker to derive the secret key. It is for this reason that it is often recommended to implement the XOR function as a one-time pad encryption algorithm when encrypting static messages. The research described in this thesis proposes an XOR function which uses the same key with a varying plaintext.

Figure 2.2: Examples of the XOR function

A popular form of block ciphers is the Advanced Encryption Standard (AES). AES was developed to replace the Data Encryption Standard (DES) such that it could support 128, 192, and 256 bit block ciphers and be more efficient than DES[9]. To secure the plaintext, the AES algorithm repeats a series of four steps (SubByte, ShiftRows, MixColumns, and AddRoundKey) for a total of 9, 11, or 13 (respective to the key length) iterations[9]. Figure 2.3 demonstrates the order and repetition of these steps.

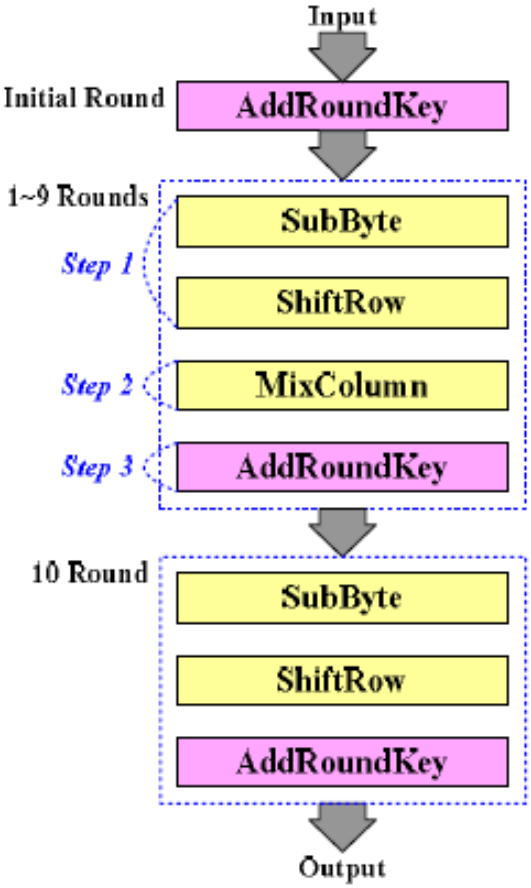

Figure 2.3: Advanced Encryption Standard encryption algorithm [26]

If an attacker can only receive the traditional plaintext and ciphertext mappings, their only method of breaking the encryption is by a brute force attack against a full AES encrypted ciphertext[9]. There have been several attacks which have been able to deduce the secret key used, however the common theme among these attacks is that the cracked ciphertext was produced without iterating all the rounds required by the definition of each AES model[7][17][5][6]. One method in which the attacker can derive some information about the plaintext is called a side channel attack which uses information such as the process duration and power drain[9]. It has been theorized that the adversary can correlate this information with the knowledge of the AES encryption procedure to determine characteristics of the secret key[9].

Hash functions

Unlike the cipher, hash functions do not rely on secret information to process plaintexts. Hash functions are primarily designed to be one-way functions that are often used within password based authenticators[9]. These functions can take any message and turn it into a fixed length message digest which represents the original plaintext[9]. For a hash function to be considered secure, an adversary cannot derive the original plaintext from the digest within a reasonable amount of time[9].

One feature of a hash function is collision resistance. This resistance implies that an attacker cannot generate an input to a hash function that will result in the same output[9]. Two tests are available to determine a hash function's collision resistance. The first test states that finding a hash input that results in the same output as a given output is nontrivial[9]. The second test states that finding two messages, a given message and a randomly chosen message, that will result in the same hash value is nontrivial[9].

Popular hash functions in encryption history include MD4, MD5, SHA-0, SHA-1, and SHA-2[9]. MD4 and MD5 are not considered collision resistant, therefore the functions were deemed insecure and retired[9]. The SHA (Secure Hash Algorithm) family of hash functions, numerated by generation, have survived longer than the MD functions due to their collision resistance. While the current full SHA-1 algorithm has not been cracked, attacks have been successfully completed by researchers who have reduced the number of rounds the algorithm processes[9].

2.2 RFID tag vulnerabilities

Much like the classes of RFID tags, the types of attacks can be split into separate categories.

2.2.1 Passive Attacks:

Passive attacks involve an adversary eavesdropping on the messages sent between RFID tags and receivers[42]. While most of the newer tags use security layers to protect the data stored within, the attacker can still gain relevant information. This type of attack can gather position data over time and allow the attacker to determine possible relations between groups of tags and their identities[42]. These relations can provide insight into the stocking procedures of a competitor or the daily routines of tag holders. It is

also important to note that first generation contactless bank cards and passports have been found to use weak security measures[20]. While this protects the information from immediate use, the attacker can store the data to be cracked at a later time. Increasing the anonymity by varying the appearance of the ciphertext will help prevent an attacker from gathering information about the relationships between transmissions. The purpose of this thesis is to develop a method to disrupt this type of attack.

2.2.2 Active Attacks:

Attacks which degrade or halt messages between RFID tags and receivers are known as active attacks and include the replication, modification and deletion of these messages[42].

Denial of Service Attacks:

Denial of service (DoS) attacks continuously send random information to the victim receivers such that the processing of legitimate communications will be either delayed or impossible[42][14]. To prevent the attack from being blocked, the attacker can also randomize the return address and use multiple nodes to transmit the randomized data[42]. Denial of service attacks can also be implemented indirectly by interfering with the RFID tag rather than the receiver[36]. Attackers can attempt to crack the password protecting the tag's kill command, desynchronize the tag, or jam communications[36]. The paper written by Tagra et al.[36] proposes an alteration to the Gossamer protocol to protect the system from DoS attacks. The Gossamer protocol suggests that RFID tags and receivers share subkeys (portions of a single key) to mutually authenticate each other[36]. While this protects the system from accepting commands which would damage the tag or receiver, signal jammers and flooding receivers with fraudulent tags will still reduce the efficiency of communication between authenticated tags and receivers.

Replay Attacks:

There are two forms of replay attacks: tag cloning and man-in-the-middle attacks[14]. Tag cloning relies on the recording of transmissions being sent between the tag and receiver[14]. Adversaries can reuse the information they have gained using passive attacks to either mimic a legitimate tag or receiver[42]. An adversary can retransmit receiver instructions to RFID tags, fooling these tags into rebroadcasting their information more frequently than required. The resulting tag data can help the attacker gain information

about what the tags are attached to as seen in the section referring to passive attacks. Similarly the adversary can rebroadcast tag transmissions to convince a receiver that it is a legitimate tag. This can give the attacker access to data or physical locations that would otherwise be secure.

2.3 Basic modifications to current protocols

Some of the simpler modifications involve increasing the number of encryption keys or stored ciphertexts on the tag. Other modifications involve substituting one method of encryption for another. This section will describe how these types of modifications affect the tag and its security problem.

2.3.1 Directly increasing the number of ciphertexts

The majority of tags have a one-to-one ratio when comparing stored encryption data fields to transmitted ciphertexts. These data fields can either be a pre-encrypted ciphertext generated during the manufacturing of the tag or encryption keys which can be used by an on-tag encryption engine. The reason for the one-to-one ratio when storing pre-generated ciphertexts is due to the fact that the tag does not have the capability to generate others. Tags with an on-tag encryption algorithm could in some cases have a higher ratio of keys to ciphertexts, however most tags do not have such an engine built into them.

Adding more ciphertexts or encryption keys would solve this problem, however the number of ciphertexts needed to be effective would not likely fit on the memory set aside for that information. One or two additional ciphertexts may stop an eavesdropper listening once, however a more patient adversary would wait for the tag to transmit several times, generating a list of ciphertexts associated to an individual tag. The cost of increasing storage to store a sufficient number of ciphertexts would make the tags too expensive to keep them at their current prices.

In addition to the cost increase, additional ciphertexts generated by multiple keys will also affect the tag's security as well as the speed at which a receiver can decrypt. To decrypt the ciphertext with a one-to-one ratio encryption, the receiver must use the same ciphertext as the RFID tag used to encrypt. There are two solutions to this problem, either the tag and server can follow a pattern of keys, or the receiver can

attempt to decrypt the ciphertext with every key available in its memory. A pattern could be determined over time such that the attempt to disassociate the tag would fail. The pattern that the tag and receiver would follow would also require them to be in sync. The second method, which involves the receiver decrypting the message with every encryption key would increase the time for a successful communication to grow by the number of keys stored in the receiver‘s memory.

2.3.2 Indirectly increasing the number of ciphertexts

The fact that the majority of tags generate one ciphertext per encryption key was mentioned in the previous section. The encryption algorithms used within these tags do not need to be as advanced as those used for communications between servers. The reason that this simplicity is allowed to be run on RFID tags is that the adversary has no way of directly testing the algorithm being used. Examples of common simple encryption breaking attempts include the adversary using the encryption engine to encrypt its own plaintexts to determine a pattern. These attempts cannot be made on the tags since the adversary has no way of communicating with the encryption engine. While the information being transmitted to the receiver is generally safe within the encryption, the problem statement refers to the fact that this ciphertext is usually the same for each transmission. One way of adding available ciphertexts without directly increasing the number of keys stored, would be to add some form of randomization to the tags. Several encryption algorithms use a randomization engine to jumble the ciphertext each time it is generated. This method results in unique ciphertexts being generated while still using one encryption key. These algorithms require greater computation power, resulting in either an increase in processing time, or a faster processor being added to the tag. Similar to increasing a tag's storage, replacing the microprocessor with a faster unit would increase the cost of manufacturing a tag, therefore the goal of these functions is to run within their time restrictions.

2.4 Previous research on this topic

2.4.1 Towards lightweight secure communication protocols for passive RFIDs - Huang and Kapoor[21]

Dijiang Huang and Harsh Kapoor focused their research on reducing the computation complexity of tag based encryption. Rather than using hash functions, the method proposed uses XOR and modulo addition operations to create a secure one time pad encryption algorithm. In addition to the new encryption algorithm, the researchers also propose a communication protocol which provides secure key lookups, key transportation, reader-tag mutual authentication, and data confidentiality[21]. The purpose of this protocol development was to reduce the cost of the tag by reducing the number of gates required to encrypt the tag data. The researchers citing additional research stated that encryption algorithms which include hash functions (such as MD4, MD5, SHA-1, and SHA-256) require 7350 to 10868 gates [11][21] and that the AES-128 encryption algorithm requires 3400 gates [12][21]. The method proposed by Huang and Kapoor resulted in a basic scheme (1850 gates) and an advanced scheme (2850) which met their set security and performance requirements[21].

The security requirements set by the researchers were to prevent unauthorized reading of RFID tags, prevent tracking of RFID tags, and to prevent replay attacks[21]. To satisfy the authorization requirement, the researchers used a trusted third party (such as a secure networked device) to distribute the encryption keys to the reader and tags[21]. The tracking requirement was solved by using a different key for each tag transmission[21]. To prevent a replay attack, Huang and Kapoor used a challenge response scheme which ensured that only legitimate readers and tags could initiate the transfer of information[21].

Huang and Kapoor used the manipulation of random data to authenticate the tag with the reader. To manipulate this data the tag and reader must share two secret values Y and Z which would be added to and subtracted from the random values transmitted[21]. The results of this manipulation would ensure that the tag and reader could confirm the other's identity[21]. The use of the random values also hides the authentication process and secret values from any adversary trying to gain information by passively listening to the transmissions[21]. In addition to preventing tag tracking, the combined use of random values and secret value manipulation forms a challenge/response protocol which will prevent the reuse of transmissions to gain access via replay attacks[21].

To encrypt the tag data, the researchers used an exclusive OR (XOR) function with a unique key per transmission[21]. This key was generated by running the previously generated random strings through the XOR function[21]. The benefit of XOR encryption is its low cost in terms of logic gates required to implement the function. The disadvantage with XOR encryptions is that the encryption key must be regenerated when encrypting static data. If the key is reused with this static data the ciphertext will also remain static, which would fail the security requirement of preventing tag tracking.

Algorithm 1 Basic Scheme.

1: $R \rightarrow T : A_1 = n_i - Z_j$
$T : n_i' = A_1 + Z_j$
2: $R \leftarrow T : challenge = Y_j \oplus n_j$
$R : n_j' = challenge \oplus Y_j$
$R : n_{ij} = n_i \oplus n_j'$
3: $R \rightarrow T : A_2 = n_{ij} - Z_j$
$T : n_{ij}' = A_2 + Z_j$
$T : n_i'' = n_{ij}' \oplus n_j$
4: **if** $n_i'' == n_i'$ **then**
$R \leftarrow T : (Data||h(Data)) \oplus n_{ij}'$
R : R performs the following verifications:
$(Data||h(Data)) \oplus n_{ij}' \oplus n_{ij} \Rightarrow Data'||h(Data)'$
if $h(Data') == h(Data)'$
accept
else
discard
end if
5: **else**
$R \leftarrow T : n_{rand}$
R : Data verification fails
end if

Algorithm 2 Extended Scheme.

1: $R \longrightarrow T_j : s1||n_i$
$T_j : x = g(T_j, n_j)$
$T_j : B = (Y_j + n_i) \oplus n_j$
$T_j : challenge_j = B||x$
2: $R \longleftarrow T_j : \forall T_j \in L, s2||challenge_j$
R : for $j = 1, \ldots, N$ recovers T_j and n_j from x
R : uses Y_j and n_i to recover n_j' from B, and checks
if $n_j == n_j'$ **then**
for $j = 1, \ldots, N$ computes:
$A_j = (Z_j + n_i) \oplus (Y_j + n_j)$
$n_{ij} = n_i + n_j + Z_j$
$nonce_{ij} = g(T_j, n_{ij})$
3: $R \longrightarrow T_j : \forall T_j \in L, s3||nonce_{ij}||A_j$
else
stop
end if
$T_j : \forall T_j \in L$ checks $nonce_{ij}$ and then computes
$n_i' = [n_{ij} \oplus (Y_j + n_j)] - Z_j$
4: **if** $n_i' == n_i$ **then**
$T_j : nonce_{ji} = g(T_j, n_{ij} + 1)$
$R \longleftarrow T_j : \forall T_j \in L$
$s4||nonce_{ji}||(Data_j||h(Data_j)) \oplus n_{ij}$
R : based on $nonce_{ji}$, R recovers $Data_j'$ and its integrity checking code $h(Data_j)'$ using n_{ij} and then checks:
if $h(Data_j') == h(Data_j)'$ **then**
accept
else
discard
end if
5: **else**
$R \longleftarrow T_j : \forall T_j \in L, s4||n_{rand}$
R : Data verification fails
end if

Figure 2.4: Basic and Enhanced RFID schemes [21]

Huang and Kapoor concluded that they had developed a lightweight protocol to secure RFID data without resorting to traditional encryption techniques. The researchers also listed improvements which could be researched in the future. This list included the development of methods to prevent the adversary from learning the secret values stored on the key as well as methods to update the secret keys stored in a tag prior to the next transmission[21].

2.4.2 Low-cost Cryptographic Circuits for Authentication in Radio Frequency Identification Systems - Kim et al.[26]

This paper introduces a low cost AES cryptographic circuit which improves the speed and decreases the complexity of previously designed FPGA and ASIC AES circuits. Improvements to the existing AES design included the combination and reduction of arithmetic steps as well as the size of the function's architecture[26].

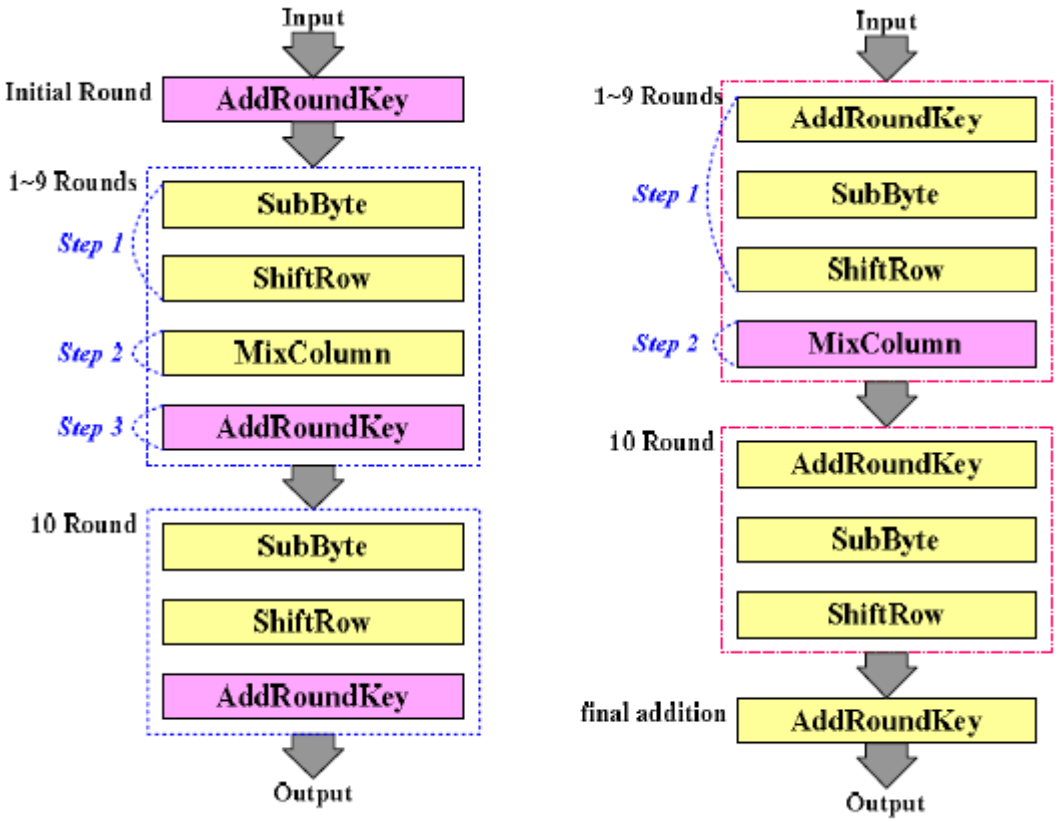

Figure 2.5: Comparison of the standard and modified AES algorithms [26]

To optimize the throughput of the AES algorithm, the researchers reordered the AES round operation steps such that steps 1 and 3 were combined into a single step[26]. This was possible since SubByte, ShiftRow, and AddRoundKey can all operate on one byte and that ShiftRow does not change the data when it reorders it[26]. This reordering of functional steps also removes the need of the initial AddRoundKey, saving the clock cycles required for its operation. The data path design allows for the SubByte, ShiftRow, and AddRoundKey to be executed at the same time before being saved to an eight bit register[26]. This path results in one operation round to be completed in 42 clock cycles[26]. The researchers also chose to restrict the tag architecture to 8 bits to meet the RFID tag's strict circuit design environment as well as to save the silicon area otherwise used for higher numbers of Sbox[26]. The Sbox is used for the SubByte operation as well as to generate the next round key during the MixColumn operation[26].

Through the various adaptations to the original AES algorithm and circuit design, the researchers created an AES encryption circuit which can encrypt a 128 bit block of data within 870 clock cycles, using less than 4000 gates[26].

2.4.3 Spacing based Authentication Protocol for Low-Cost RFID - Jin et al.[25]

It is understood that RFID tags provide lower processing capabilities than larger computer systems, and that this limitation can lead to less secure systems. To improve the security and reduce the number of times encrypted data is transmitted, it is common practice to authenticate the reader and tag taking part in the communication process. This paper proposes an efficient authentication process which utilizes XOR operations and a spacing algorithm to provide forward security and location privacy[25].

The proposed authentication relies on the trusted agent and the tag sharing a secret identifier which is separate from the RFID tag's unique identifier[25]. The tag's spacing algorithm splits the secret identifier into segments and sorts them into two strings based upon whether the segment identifier is even or odd[25]. The length of the segments used is randomized and sent from the receiver as part of the query[25].

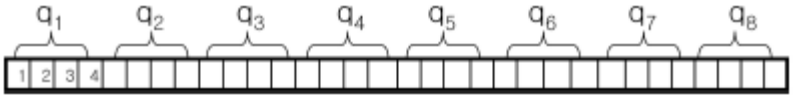

Figure 2.6: Secret value segmentation with segment lengths of 4 [25]

These two strings are then run through a XOR function and passed along with the number of segments to the trusted agent via the receiver and is referred to as the partial identifier[25]. The agent then looks through its database to find a secret value which would return the same value as it has received[25]. If the agent can find this value, the tag is then considered authentic[25]. To authenticate the receiver, the agent chooses a new segment value and re-segments the same secret value[25]. That value and the length of the segments is passed back to the tag. If the tag can recreate the segmentation with the new number of segments, then the receiver is considered authentic[25].

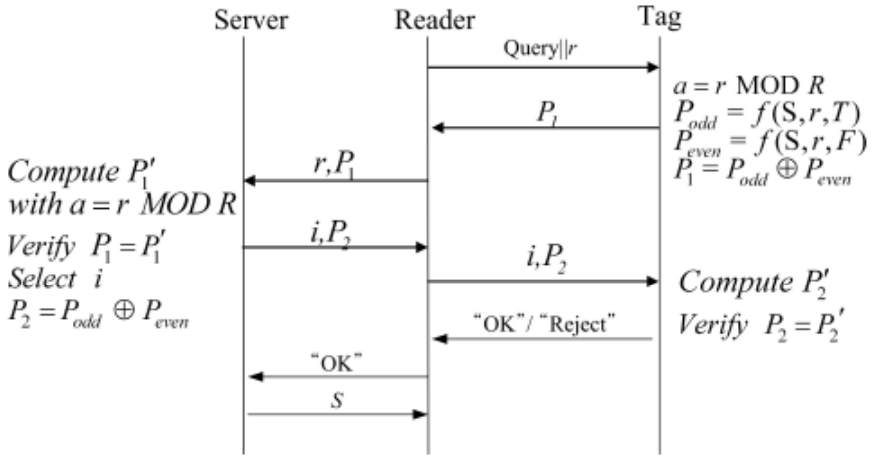

Figure 2.7: Basic spacing algorithm [25]

To ensure the security of the authentication, and to prevent an attacker from building a library of responses, the secret value can be recalculated by resegmenting the original secret value with the new segment length being the sum of the previous two lengths[25].

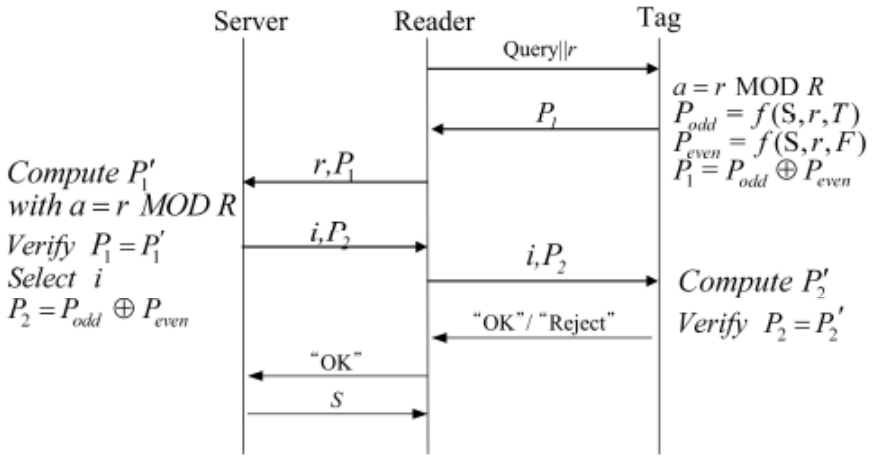

Figure 2.8: Advanced spacing algorithm [25]

The researchers determined that, with this advanced algorithm switching the secret value, the probability that an attacker could guess the correct partial identifier can be represented by the following equation:

$$Pr[P = P'] = (\frac{1}{2})^{(96/r)} \tag{2.1}$$

where 96 is the length of the secret value and r is the segment length sent by the receiver in its initial query[25]. The researchers considered this value to be negligible, therefore they considered this method to meet their security requirements[25].

2.4.4 General comments on the previous research

While some protocols utilize advance encryption algorithms and others use a combination of advanced authentication and less complex XOR functions to secure the information transmitted within, the commonality between them is the length of the randomized strings. The method proposed by Huang and Kapoor requires two random strings to be generated, each with the same length of the message[21]. Once those values are run through the XOR function, it results in one encryption key with the same length as the random strings. If it were not for both parties calculating the secret key, one string of the same length could be randomized and have the same chance of being guessed by an attacker.

$$Pr[P = XOR(R1, R2)] = PR[P' = R] = 2^l \tag{2.2}$$

The method developed by Kim et al. takes this randomization further by requiring new keys for each round of the AES encryption[26]. In addition to the number of times the random number generator is used, the number of gates required is also a concern. Advanced encryption algorithms can be replaced by XOR encryption if the method of generating a key has a higher impact than that of the method developed by Huang and Kapoor[21].

The authentication method described in the paper written by Jin et al.[25] provides an excellent way to include mutual authentication in the RFID communication protocol. Since the probability of successful attacks is dependent on a ratio of the randomized length and segment length it is also flexible to work with messages of any length provided that the segment length is properly restricted. This method of authentication could be altered to provide randomization instructions to the plaintext prior to encryption.

Chapter 3

Protocol Generation

The modification method proposed involves inserting random characters into the EPC prior to encryption. This modification will provide a greater number of messages such that one encryption key will still result in different ciphertexts for each iteration of the algorithm. This chapter discusses the potential alteration methods and their benefits.

3.1 Modified identifier characteristics for anti-collision protocols

Each RFID tag has its own system-unique static identifier. While this identifier can be used in communication with the receiver, the fact that the identifier does not change can affect the privacy of the tag. Since the identifier is static, transmitting this value before each modified message will undo any attempt to disassociate the data being sent. A solution to this would be to generate a random identifier for the tag to use during any anti-collision stage of communication.

EPC capable RFID tags are required to contain a random number generator with specific performance characteristics, as seen in chapter 2. One of these characteristics is that the chance that any two tags, out of a pool of 10000, will generate the same 16 bit random number is less than 0.1 percent[23]. This means that the chance that an anti-collision process will select two tags with the same temporary identifier is negligible. Once the tag has passed through the anti-collision process, it can then properly identify itself via the authentication stage.

3.2 Plaintext modification

Once communication has been arranged, the fill must be generated and placed within the message. This action can be referred to as the modification of the plaintext.

3.2.1 Generation of fill

The type of fill chosen can affect the method of generation. The character restrictions do have a direct impact on the randomly generated fill. When generating a random string, the range will be restricted to the numeric range (0-9), the alpha-numeric range (0-9, a-Z), or the ASCII range which can represent 256 different numbers, letters, and symbols. The benefit of using fill with a greater generation range is that it can provide a greater number of modified plaintexts within a shorter fill length.

range	numeric	alpha-numeric	ASCII
number of possible values per character	10	62	256

Table 3.1: Number of variations per character

In addition to these character ranges, subcategories can further restrict the range of available characters. Examples of these subcategories include case-sensitive vs non-case-sensitive restrictions on the alpha-numeric and ASCII ranges and the binary restriction (0-1) on the numeric range. While the case sensitivity would be arbitrarily chosen by the program designer, the numeric restriction would be mandated by the format of the message (integer/binary) during the implementation of the message modification.

When a random character is generated, the result is numerical and a second process must be used to translate that number into a letter. If a non-numerical range is selected, then multiple random characters would be needed to represent one fill character, therefore the length of the random number generated would be greater than the length of the fill. This means that for each character, the algorithm must first identify the set which represents the next fill character. The increase in character set lengths could also require the random number generator to be used more than the numerical range scenario, since the length of the random string must increase to match the number of sets. If the random number was generated once, the random string would first have to be broken up into individual sets and then translated. The identification step would not be necessary while generating fill characters as needed, however it would still require the translation step.

The type of fill used will also depend on the characteristics of the encryption algorithm. If the algorithm only considers strings with a numeric range, then characters or symbols would throw an exception.

The fill can be generated either as one string that will be split during its placement, or as one character at a time during the same placement procedure. While completing the generation in one step would be preferred, it may not be possible depending on the length of the fill. Since the random number generator can only produce a number up to the MAX value set in its language, the fill generator would have to generate multiple random numbers if the fill length is greater than that of the MAX value.

The main goal of this research is to make this new protocol as modular as possible, such that it can be used with the majority of existing re-programmable RFID tags. For the reasons mentioned in this section, the remainder of the research used a fill generated with a numeric range. The following sections will show how the numeric range (0-1 or 0-9) was chosen to be included in the final modification protocol.

With the method of fill generation chosen, the next step in the transformation is to place the characters within the plaintext. Each method will include discussion on how the fill is inserted into the plaintext as well as how the modified plaintext will guide the receiver to locate and remove the fill. The following sections discuss the different methods of placement along with their respective effectiveness. The effectiveness of a method can be determined by comparing the number of random plaintexts generated per random fill character as well as the ease of fill insertion and removal.

3.2.2 Comparing random character placement methods

This research is meant to be applicable to any form of message being sent, therefore the content of each message is irrelevant to the generation of a new protocol; however generic characteristics such as message length and character restrictions will have an effect. The message's length is made by adding the length of the EPC and the length of the total random segment. If the identifier is required, that length is then added to find the final message length.

$$M = EPC + F \tag{3.1}$$

While the length of the message does not affect the number of random characters used, the increase in EPC length as well as identifier length provide a greater number of available spaces to place single random fill characters. A single random fill character can be defined as a random character surrounded either by characters from the original message or additional random characters. This results in a characteristic which can increase the total number of possible combinations without affecting the generated random string. An example of how the length can vary with standardized information is the implementation of the modification. Since a common length of the EPC is 96 bits, it would appear that the length would be static. However, whether the modification is made in the EPC's integer form or binary representation can affect the number of spaces available to place a single character (one without additional random characters surrounding it). The remainder of the paper will refer to the EPC as 96 bits in length in its binary form and will refer to the integer form by its maximum length of 29 characters.

Four methods of random character placement will be discussed in this section: random block placement, set pattern placement, random pattern placement, and indexed placement. One of the simplest tests used to compare these methods will be ranking the placement methods by the number of modified plaintexts it can generate. Three of these methods contain operational and potential plaintext values, this will be further explained in each placement description.

Ideally the method that generates the highest number of modified plaintexts will be chosen, however the computation requirements must also be taken into consideration. The goal of this chapter is to narrow down the potential modification methods which will then be simulated in the next chapter. That simulation will determine if the extra plaintexts generated will greatly affect the selection and encryption of messages prior to transmission. If that time is greater than the set limit, then the method must either be modified or replaced with one of the other suitable placement methods mentioned below.

Random block placement

The random block method adds the generated random fill as a single string within the plaintext. This method either requires the receiver to know the length and position of the fill, limiting those characteristics to one value, or to understand characteristic variables added to the modified plaintext.

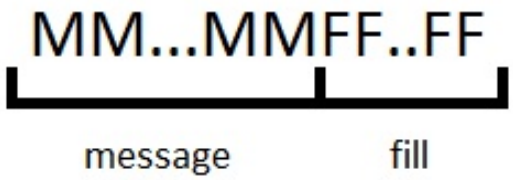

Figure 3.1: Set random block format

The simplest approach to randomizing a message would be to have a set length and position of the random fill, however it is also the method that provides the fewest modified plaintexts. An example of this would be to always place the fill block at the end of the message. While this approach limits the available (operational) modifications to the maximum value of the random fill, the total (potential) number of modifications relies on the length of the original plaintext. Since an eavesdropper has no prior knowledge of how the placement is set for each system, the block can potentially be anywhere in the message. This means that the formula to determine the number of operational modifications per character is

$$R^F \tag{3.2}$$

while the formula to determine the potential modifications per character is

$$(EPC + 1) \times R^F \tag{3.3}$$

where EPC is the length of the electronic product code, R is the range of one character, and F is the length of the fill.

To increase the number of operational plaintexts, the position and length of the fill itself can be randomized throughout the message. The implementation of this random block method would require additional message formatting, adding a length and position variable to the modified message.

Figure 3.2: Variable random block format

Both length and position variables have their own requirements when added to a plaintext. The number of characters reserved for the position variable must be large enough such that any position can be listed. While the size of the fill length variable cannot be directly set by the length of the plaintext, the reader and tag can be programmed with a maximum fill length. This maximum value allows the correct number of characters to be reserved.

maximum fill length	message length	fill length	fill position	format string
9	10	3	5	03005
19	100	5	9	05009
19	150	12	45	12045

Table 3.2: Examples of modified plaintext formatting strings

The reservation of these variable lengths allows the receiver to easily split the message into its separate components. These variables are the most effective when the random fill remains as a single block within the modified plaintext and are not feasible when considering multiple blocks as the message would grow too large. Since it is now possible for the block to vary its position with each modified plaintext generation, the number of operational modified plaintexts matches the potential number created by the first style of the random block method. This also allows the previous style's potential formula to be used to calculate the number of operational plaintexts

$$(EPC + 1) \times R^F \tag{3.4}$$

The performance of both styles can be compared with the following charts which varies the fill length from one to twenty-nine.

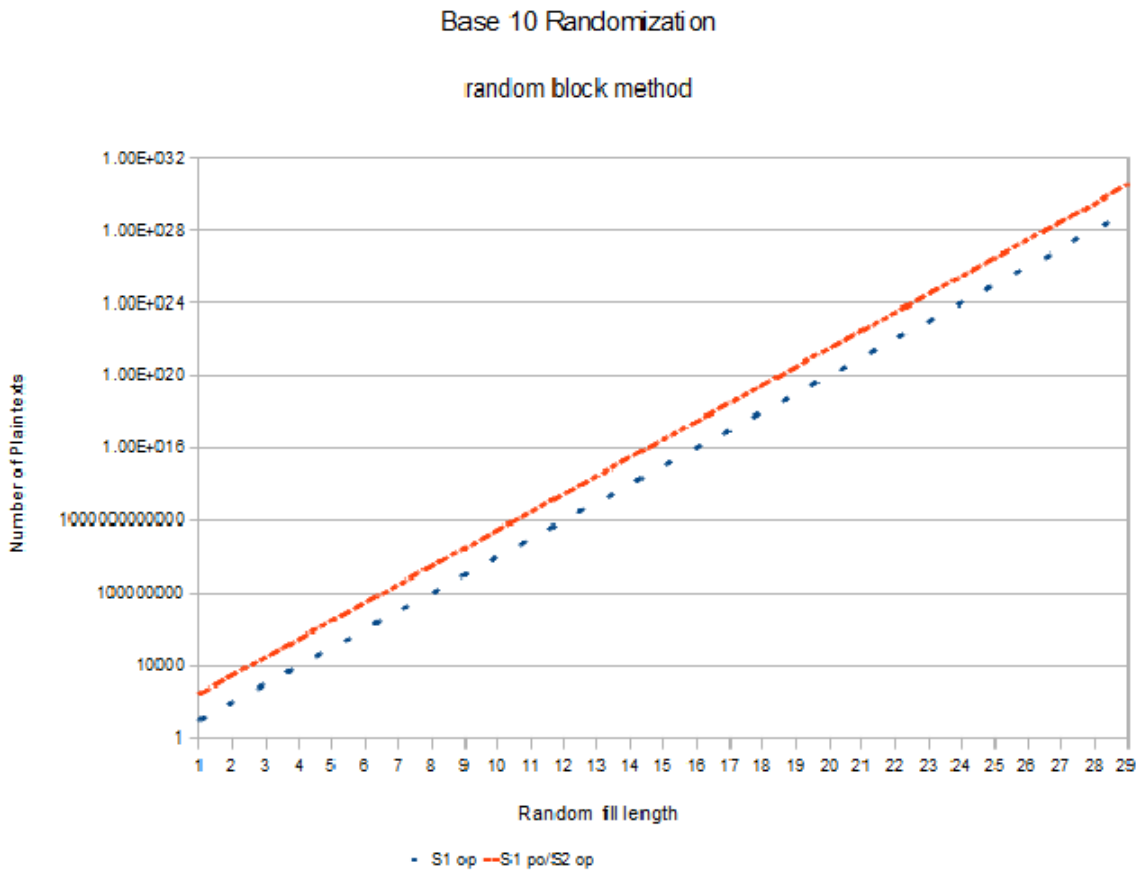

Figure 3.3: Random block plaintexts with base 10 implementation

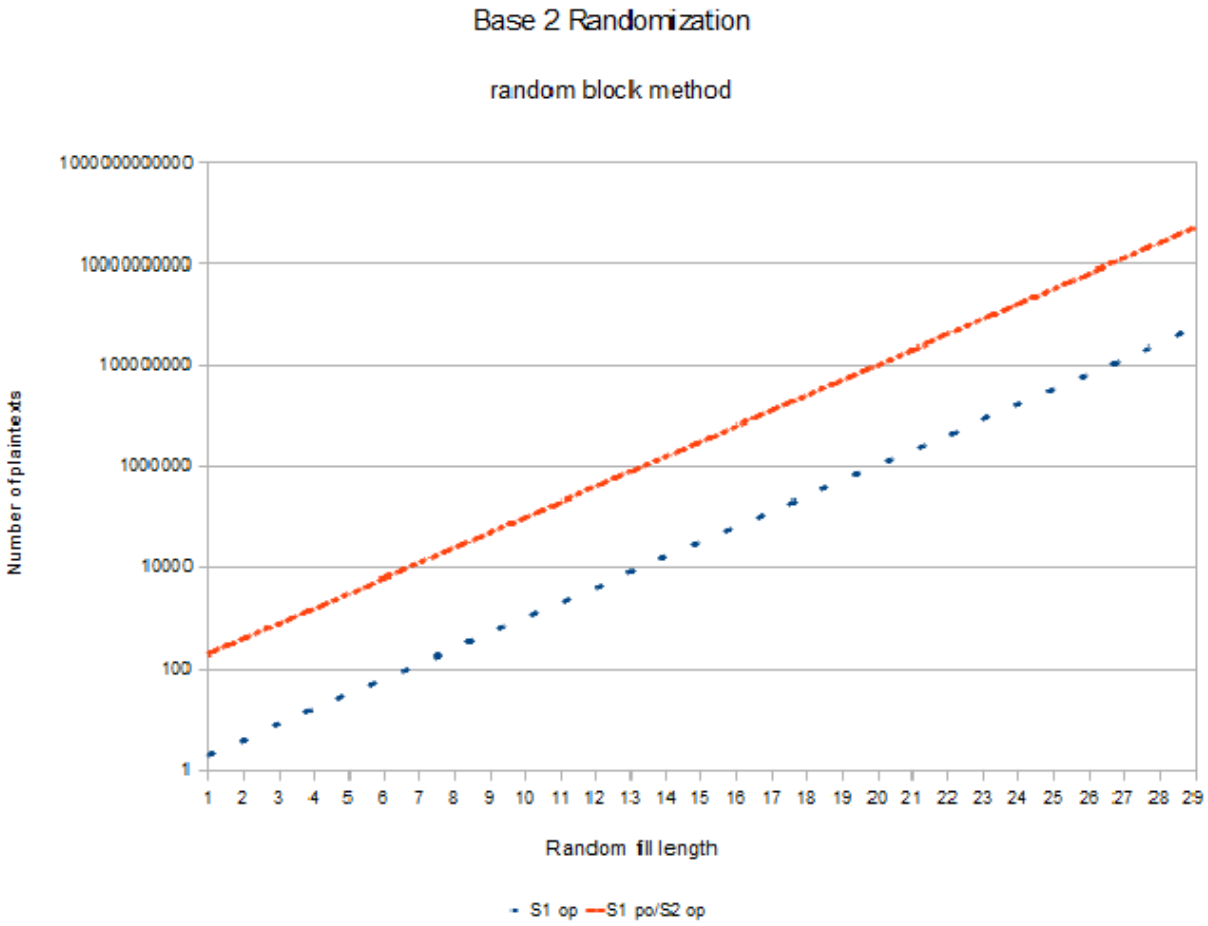

Figure 3.4: Random block plaintexts with base 2 implementation

The charts show that there is a difference between the potential and operational number of plaintexts that this method can generate. While this second style of the random block method adds more operational modified plaintexts, there are additional methods which will provide a greater number of modified plaintexts. The base 2 implementation would not be suitable due to the low number of plaintexts it would generate as well as the fact that one block of random characters will not obscure the original pattern.

Set pattern placement

The set pattern method can be considered a modification to the previous method involving the separation of the random block into separate characters. A pattern can be generated such that a system's tags and receivers share the same pattern. While this does fix the position of the random fill characters, the length of the fill varies with the length of the message. This pattern can be set during the tag's programming and represents the number of original characters to copy prior to inserting a random character.

Figure 3.5: Pattern generation

Once the pattern has been set, the generation of plaintexts repeats the pattern stored within the tag, replacing the non-fill characters with values from the original plaintext and fill characters from the newly generated fill string. This pattern will continue to repeat until all message characters have been used, ending the modified plaintext with a shortened version, or stub, of the pattern.

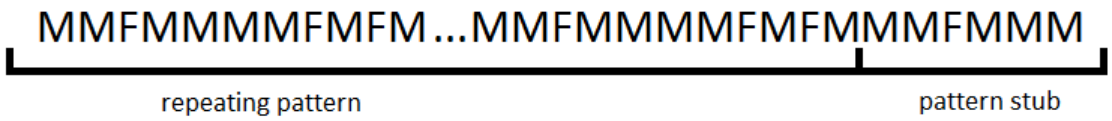

Figure 3.6: Modified plaintext generation from a set repeating pattern

Splitting the random fill into its separate characters also increases the possible locations of the fill such that the potential number of modified plaintexts can be represented by the formula

$$R^{\frac{Fp*EPC}{P}} \times \binom{Fp+P+1}{Fp} \tag{3.5}$$

while the actual number of available plaintexts during the tag's operation matches that of the random block

$$R^{\frac{Fp*EPC}{P}} \tag{3.6}$$

where Fp is the length of the random fill per pattern, EPC is the length of the electronic product code, R is the range of the random character, and P is the length of the pattern less the random fill.

To demonstrate this method, the following graphs will represent models that show a range in the length of random fill per pattern from 1 to 10 characters with a pattern length of 10 characters.

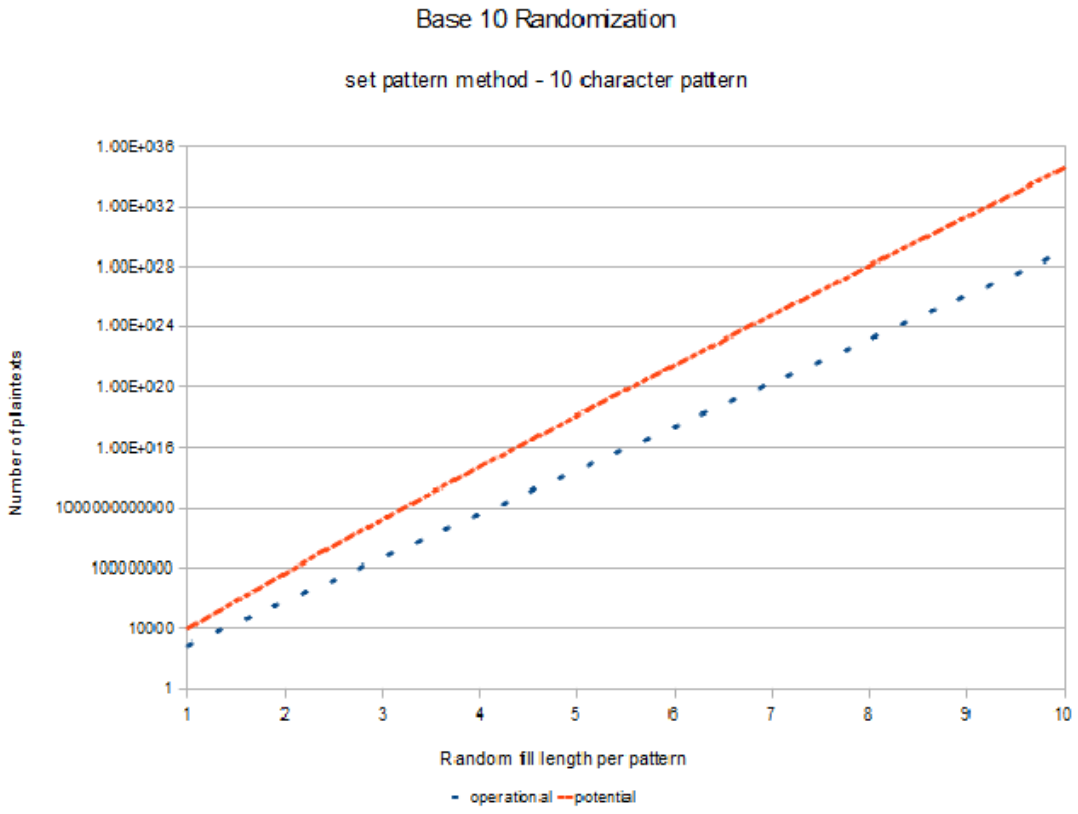

Figure 3.7: Set pattern plaintexts with base 10 implementation

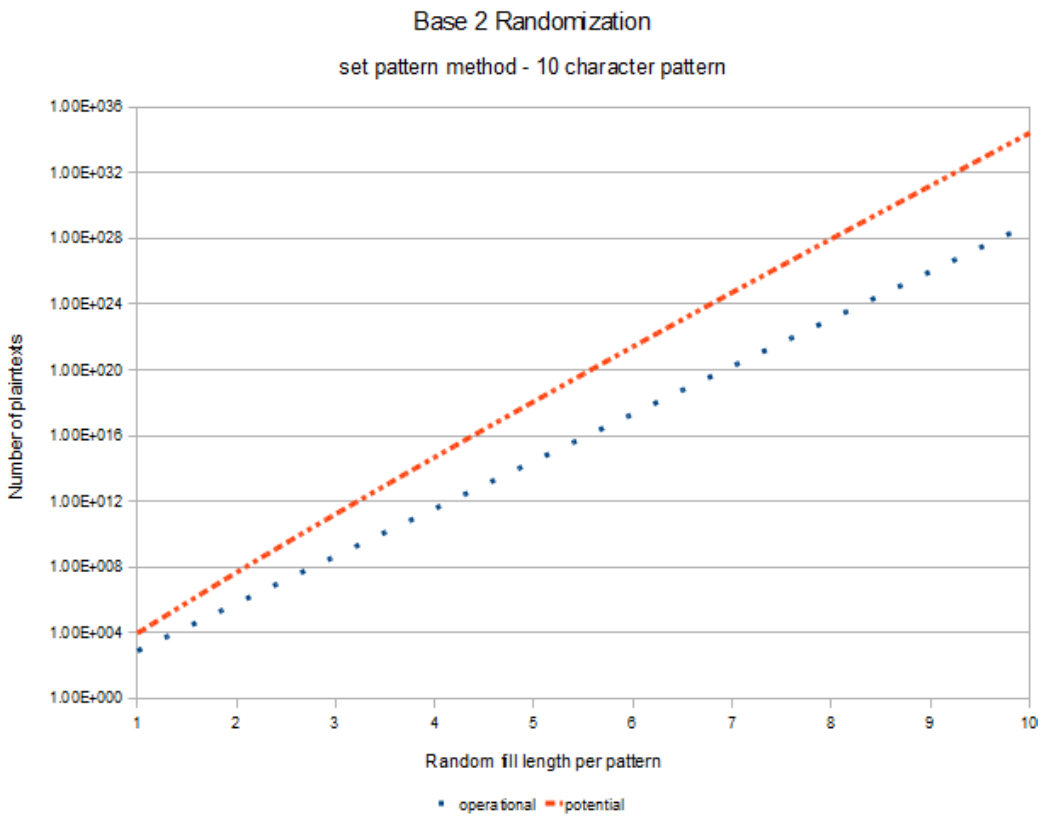

Figure 3.8: Set pattern plaintexts with base 2 implementation

The graphs also represent P as ten characters from the original plaintext. This means that for every ten original characters, 1 to 10 additional random characters will be inserted. This pattern will be repeated three times, with new values generated for the random slots, to reach the full length of the message.

The number of potential plaintexts has grown beyond that of the random block method due to the fact that the fill string has been broken up into separate characters. While this method helps expand the number of potential plaintexts, the number of operational plaintexts remain relatively low. Splitting the random block into separate characters has raised the number of base 2 modified plaintexts, however the repeating pattern aspect still makes the conversion from base 10 to base 2 after modification the better option.

The sole reason for repeating the pattern is to reduce the memory requirements of the tag as the amount of memory on the tag restricts the length of the pattern. The ideal length of the pattern would match the length of the EPC plus the random fill plus

one. Increasing the length of the pattern reduces the number of times the pattern must be repeated which, in turn, reduces the chances that an adversary will guess the pattern. If it remains cost efficient to store a full length pattern, the formulas determining the number of plaintexts can be reduced to the following:

$$R^{Fp} \times \binom{Fp + EPC + 1}{Fp} \tag{3.7}$$

while the actual number of available plaintexts during the tag's operation matches that of the random block

$$R^{Fp} \tag{3.8}$$

where Fp is the length of the random fill per pattern, EPC is the length of the electronic product code, R is the range of the random character.

This simplified formula changes the fill length effect, as can be seen in the following graph:

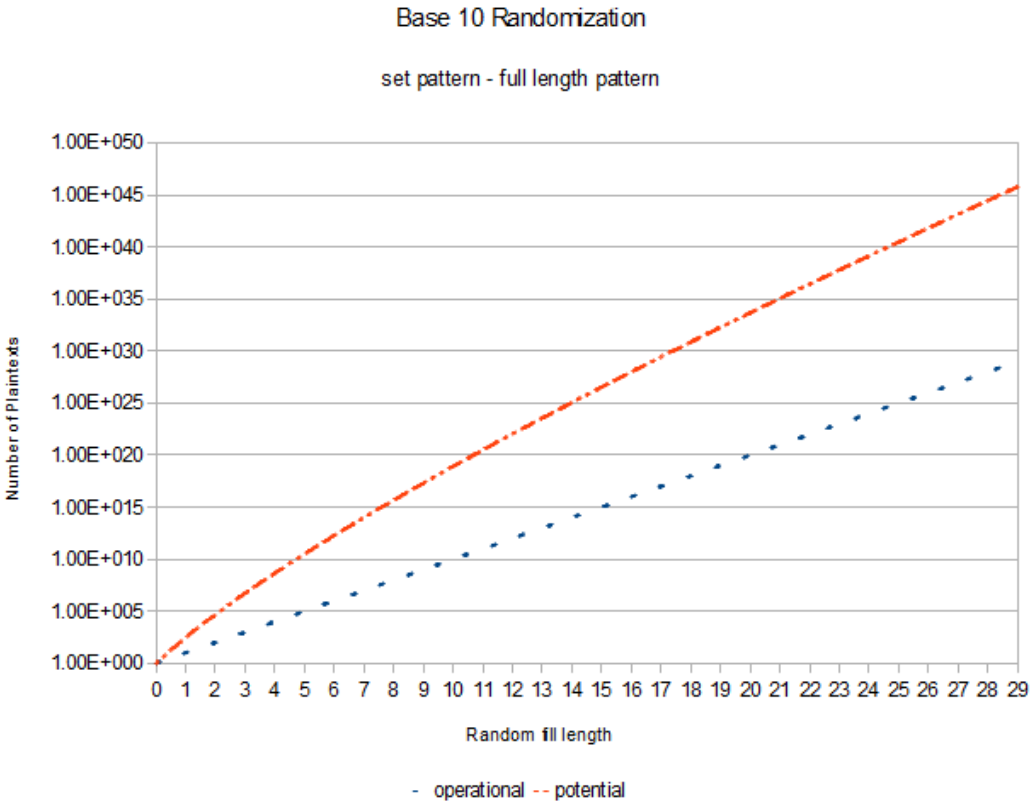

Figure 3.9: Set pattern plaintexts with base 10 implementation with a full length pattern

While the operational number of modified plaintexts have not changed, the removal of pattern repetition has increased the potential number of plaintexts. This is due to the pattern generator having the ability to insert the random fill between a larger number of characters. The next method discussed in this section will raise that number closer to the number of potential plaintexts.

Random pattern placement

While the set pattern method generates a larger number of modified plaintexts than the random block method, the difference between the operational and potential plaintexts is also larger. The use of a random pattern, rather than a preset pattern, will increase the operational number of plaintexts to be closer to the potential number of set pattern plaintexts. An additional benefit to using a random pattern is that it removes the need for the receiver to have previous knowledge of the RFID tag.

The generation of the random pattern follows the same steps as the generation of the set pattern. The difference between the two generations is the fact that a tag using this method will generate its own patterns. To maximize the variety in plaintexts, this pattern would be regenerated for each transmission between the tag and the receiver. The tag must also include a marker which will allow the receiver to be able to determine the pattern used for the modification. Since the pattern is based on the length of the identifier, which is part of the message being modified, the receiver can compare the expected identifier with the received message.

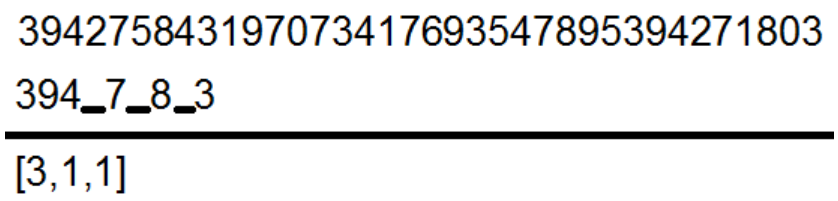

Figure 3.10: Recovering the pattern by comparing the identifier to the initial message characters

The random pattern and indexed randomization methods share the operation of random placements, however they also share the chance that a message can become corrupted by phantom characters. If the random fill character mimics the identifier character which the receiver is expecting, the pattern will become corrupted. This corruption can be avoided by creating two restrictions during the generation of the pattern. The tag first restricts the placement of the random fill characters such that they cannot be placed after the last identifier character. This allows the receiver to determine the number of random fill characters used in the pattern. The second restriction limits the values that each fill character can use within the first iteration of the pattern. Since the first iteration encompasses the identifier, stopping the fill character from matching the next expected identifier character will eliminate the receiver discovering phantom characters, however this does reduce the number of plaintexts generated. The formula of the unrestricted method is:

$$(R)^{Fp} \times R^{\frac{Fp*EPC}{P}} \times \binom{Fp+P+1}{Fp} \tag{3.9}$$

which can be compared to the restricted methods formula of:

$$(R-1)^{Fp} \times R^{\frac{Fp*EPC}{P}} \times \binom{Fp+P}{Fp} \tag{3.10}$$

where R is the random character range, Fp is the fill length per pattern, EPC is the length of the electronic product code, and P is the length of the pattern less the fill length. These differences are shown in the following graphs:

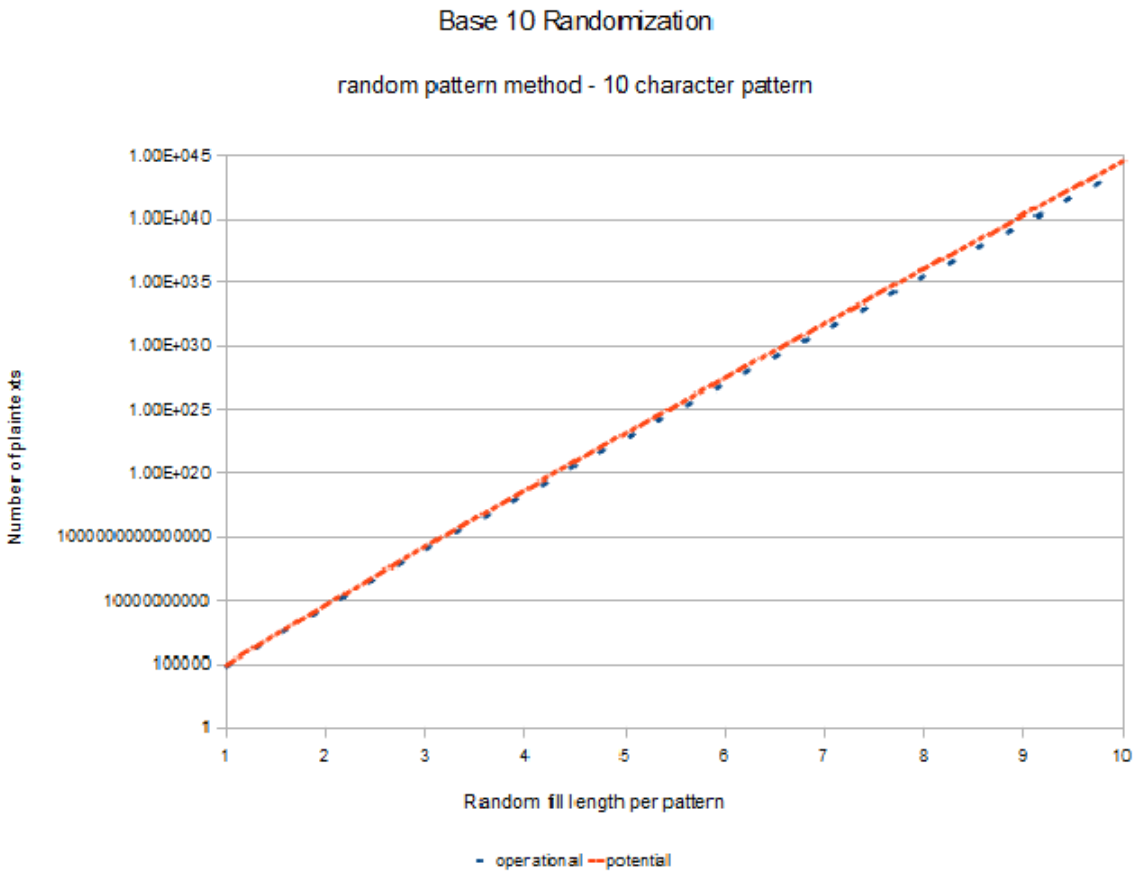

Figure 3.11: Random pattern plaintexts with base 10 implementation

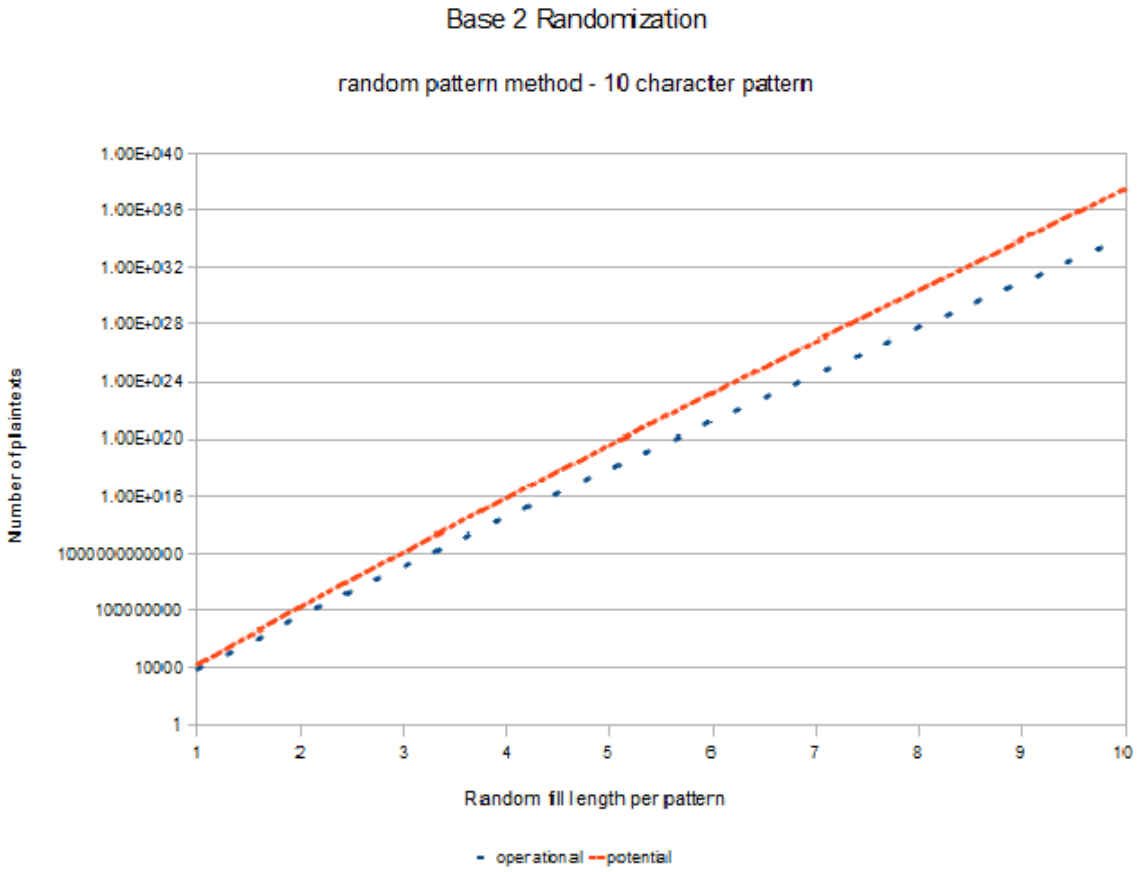

Figure 3.12: Random pattern plaintexts with base 2 implementation

The charts show that the difference between the restricted and unrestricted methods is less than the difference between the previous method's potential and operational values. The key point of the evaluation in section 4 will be to check whether or not the extra processing time required does not cost more than the gain received. Along with the increase of plaintexts generated, this method is more flexible, allowing the receiver to reduce its memory requirements by not storing the tags and their respective patterns.

The random pattern, unlike the set pattern, must be repeated at least once to satisfy the receiver's need to recover the pattern from a known value. While repetition is still required, it does not change the fact that the pattern length can still equal that of the EPC plus fill plus one as long as it does not greatly affect the cost of the tag. A full length pattern would simplify the formulas as follows:

$$(R)^{Fp} \times R^{Fp} \times \binom{Fp + EPC + 1}{Fp} \tag{3.11}$$

which can be compared to the restricted methods formula of:

$$(R-1)^{Fp} \times R^{Fp} \times \binom{Fp + EPC}{Fp} \tag{3.12}$$

where R is the random character range, Fp is the fill length per pattern, EPC is the length of the electronic product code. These differences are shown in the following graph:

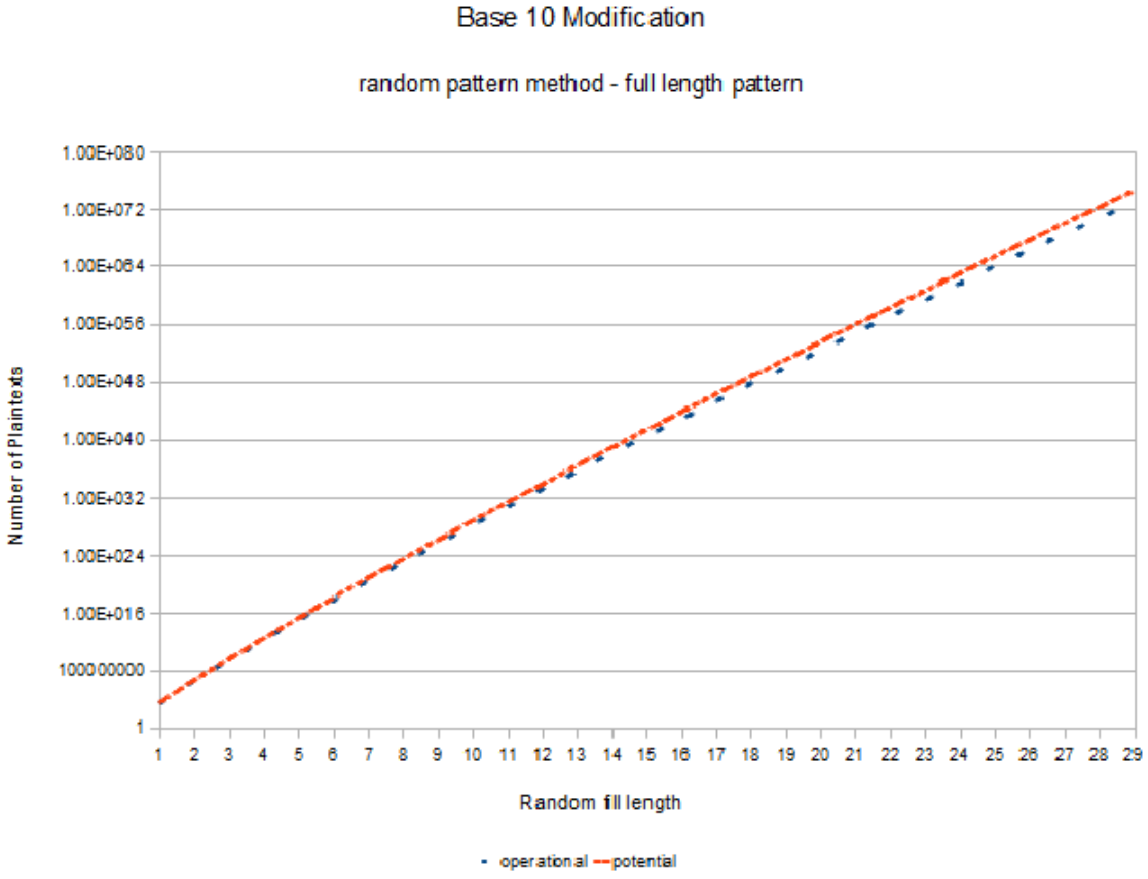

Figure 3.13: Random pattern plaintexts with base 10 implementation will a full length pattern

Indexed randomized placement

The final method discussed, called the indexed randomized method, splits and distributes the generated fill amongst the plaintext with a constant prefix added to each fill character.

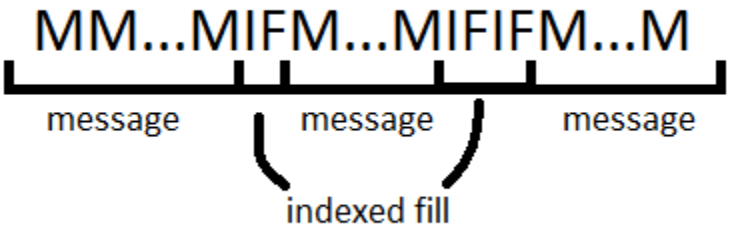

Figure 3.14: Indexed randomized placement

This prefix can either be a fixed value stored within the tag and receiver, or be a variable set by the tag's operational data. An example of a variable prefix would be the aloha slot value used within the tag's anti-collision protocols. Much like the encryption key, the prefix has the same problem of either keeping the value static and secure, variable depending on public data, or variable depending on synchronization. The third option would be to base the value of the prefix on a mathematical system which would change the prefix for each transmission, however this requires the tag and the reader to be synchronized. The location of the prefix and fill character is generated by the tag's random number generator, which produces the number of characters the tag should move along the plaintext before placing the fill.

Much like the second style of the random block method, the potential and operational number of modified plaintexts are equal. The number of modified plaintexts is determined by the following formula

$$10^F \times \binom{F + EPC + 1}{F} \tag{3.13}$$

where F is the fill length and EPC is the message length. The formula represents the total number of possible random values multiplied by the random choice of F characters out of the new modified length of the EPC plus the fill plus one. The results of this formula can be seen in the following graphs:

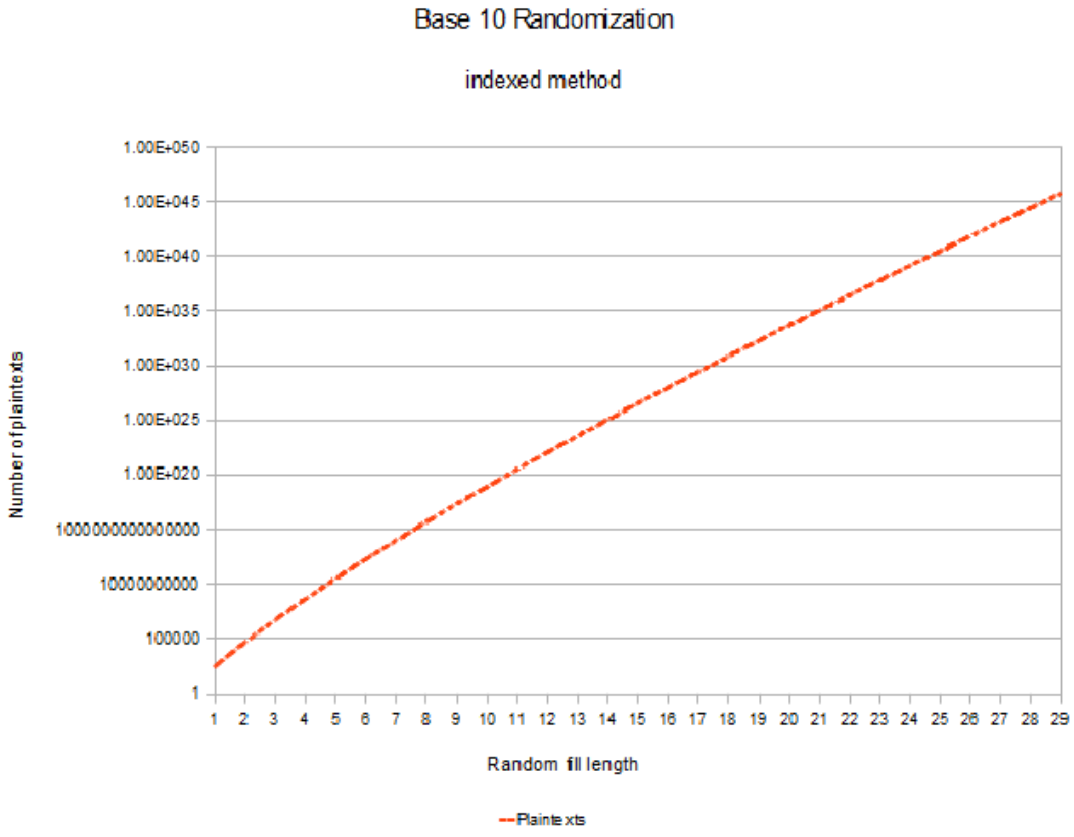

Figure 3.15: Indexed randomized plaintexts with base 10 implementation

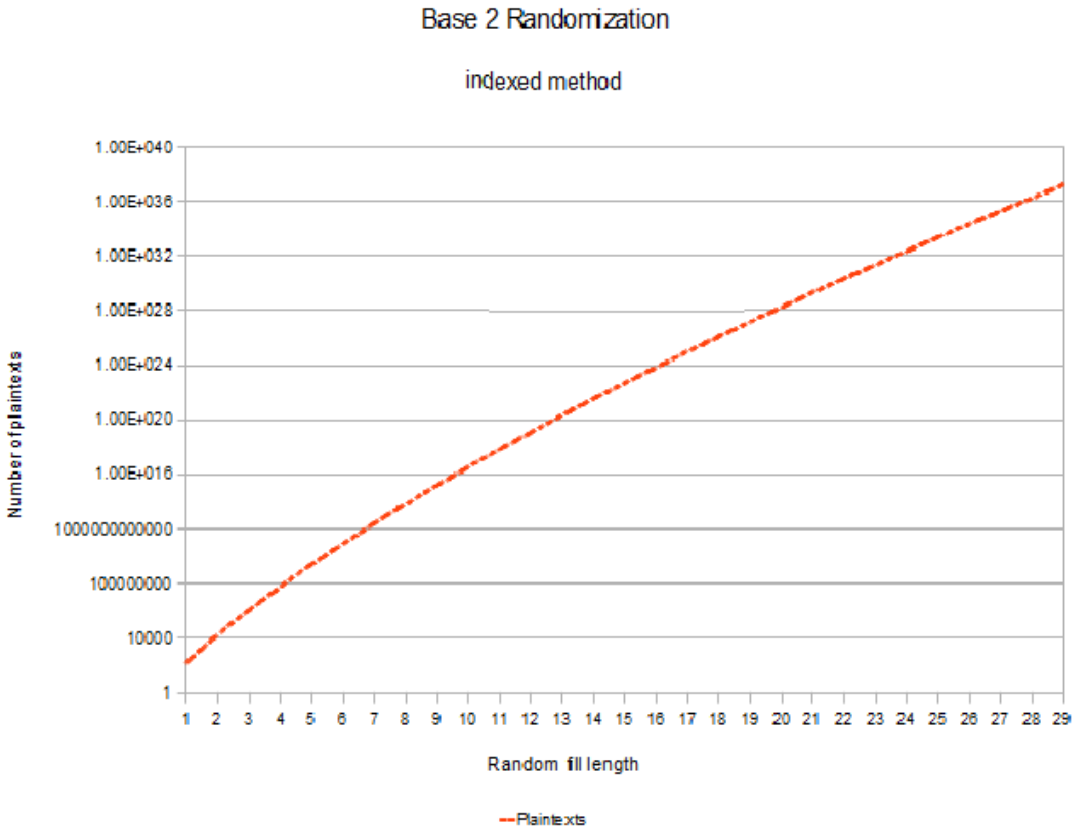

Figure 3.16: Indexed randomized plaintexts with base 2 implementation

While this method produces the highest number of possible modified plaintexts without a pattern, it is also the method which has the highest chance of failure. While the prefix allows the decoder to locate and remove the fill, it is also possible that the prefix chosen occurs naturally within the original plaintext. This duplication results in a 'phantom prefix' which directs the decoder to incorrectly remove a section of the original plaintext, corrupting the message. If the prefix is generated during the tag's manufacturing process, the prefix must not be duplicated in the original plaintext. The following examples show how the receiver would work when there is and is not a phantom prefix within the message.

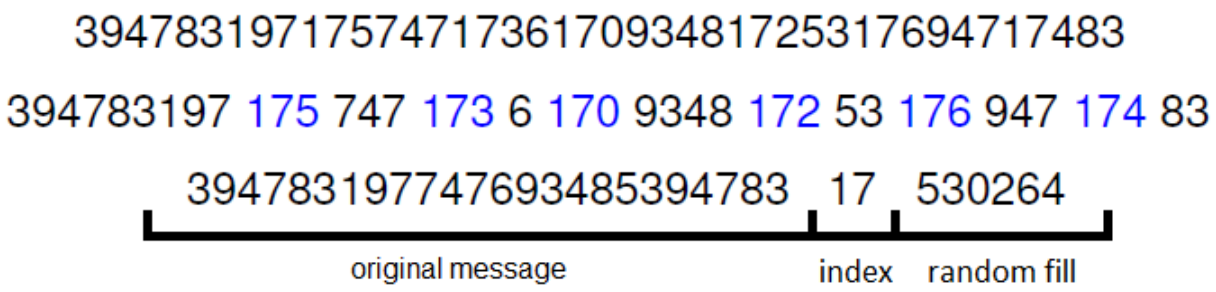

Figure 3.17: Demodified correctly with the index value of 17

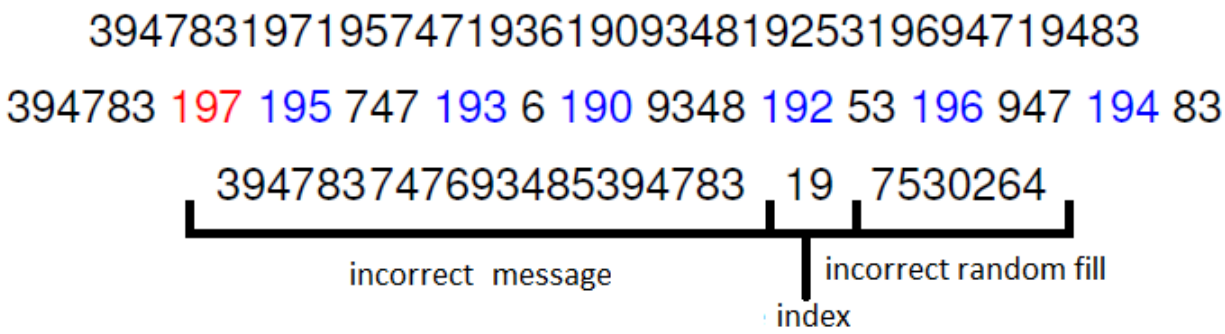

Figure 3.18: Demodified incorrectly with the index value of 19

When the receiver processed the message with an index value of 19 it accidentally removed three characters from the main body message. If the receiver knows the length of either the main body message or the random fill, it can detect that a phantom prefix has been identified, otherwise it will incorrectly log the transmission. While the length of the previously mentioned variables can be preset, the receiver will only be able to detect that a phantom prefix was found after the demodification. This means that the receiver will not be able to identify which indexed random fill character should be included in the main body message. There are three possible outcomes to the phantom prefix problem:

- the original plaintext cannot be recovered
- the original plaintext can be recovered after repeating the communication process [only applicable to variable prefixes or phantom prefixes caused by the random fill]
- the original plaintext can be recovered by creating a list of possible plaintexts

While the third method will generate a list of possible plaintexts it will also create the problem of choosing which plaintext to use. This method can work if the application is to act as a security checkpoint, allowing certain tags through, however there is still a risk that an allowed tag can be artificially generated from the list of possible plaintexts. If the application is to log any tag which passes the receiver, this generates a greater number of false tags than simply including the original incorrect reading. It is for these reasons that this method of placement is not recommended since the complications outweigh the greater number of modified plaintexts generated.

3.2.3 Comparing implementation methods

In addition to comparing the placement methods with each other, the method of implementation must also be determined. This refers to whether the original plaintext will be modified in its binary representation, or its base 10 representation. The benefit with working in the base 2 representation is that the removal of the conversion between the two representations will reduce the required computation time. The base 2 representation also provides a higher number of characters to insert random characters around (96 versus 29). The benefit of working in the base 10 representation is that the character range is much higher than the base 2 range (0-9 versus 0-1), providing a larger number of modified plaintexts with a shorter message.

The security of each implementation must also be compared to ensure that the modified plaintexts do not allow eavesdroppers to associate one modified plaintext with another. This would require more random characters to be added to the base 2 representation since a pattern would be easier to recognize.

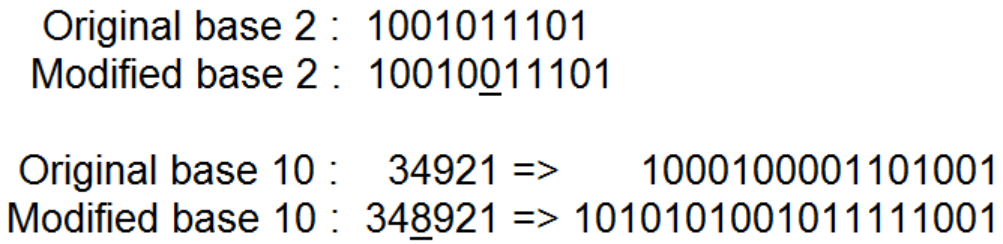

Figure 3.19: Comparison of base 2 and base 10 implementation

The base 10 representation also has a security issue. While patterns are harder to identify with the conversion from base 10 to base 2, that conversion does not guarantee that all of the available modified plaintexts will have the same length. The length of plaintext is dependent on the number of fill characters, their values, and position within the modified plaintext.

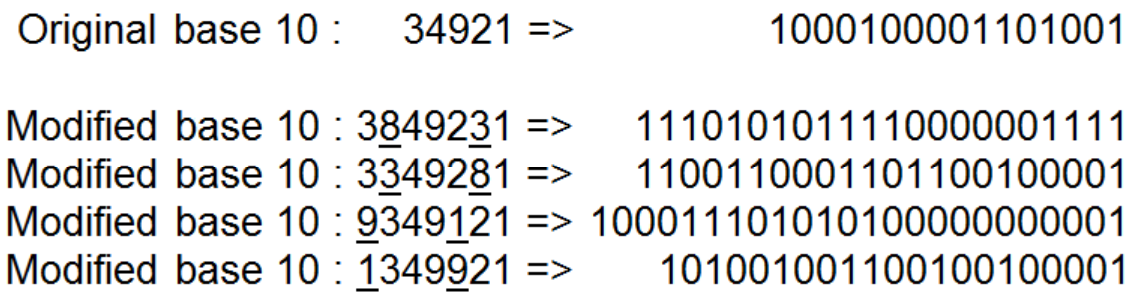

Figure 3.20: Comparison of base 10 modified plaintexts

The variance in message length could give clues to the attacker such that they can reduce the number of combinations of the three characteristics. To combat this, extra zeros could be padded to the front of the string prior to encryption, however that could also reveal (over time) the first few characters of the encryption key. It is riskier to reveal key characters than to reduce the number of combinations. Reducing combinations will make it easier to associate one message with another, while revealing encryption key characters may help to decrypt a message.

3.2.4 Placement methods and implementations chosen to be simulated in Chapter 4

In terms of placement methods, the random block, set pattern, and random pattern methods will be simulated in Chapter 4. The random block method and set pattern method generated roughly the same number of possible modified plaintexts, therefore the simulation will determine whether splitting and placing the random block into separate characters will greatly affect the computation time. The random pattern method can generate more modified plaintexts, but also requires a longer message. This length may increase the encryption time, which will be included in the total computation time. While the indexed placement method does create the largest number of modified plaintexts without requiring a pattern, its phantom prefix flaw makes this method unreliable. This unreliability is the reason for excluding it from the simulation phase.

In terms of implementation, the base 10 implementation will be used in Chapter 4's simulation of the random block, set pattern, and random pattern placement methods. The random fill characters required to hide any pattern would be several times the length of the original message. The goal of this research is to provide an alternative to generating a new encryption key, therefore the number of fill characters should not be greater than the number of characters within the key. Generating 29 integer values as a whole value results in 96 bits, the length of the average EPC key. Generating these values separately, using 4 bit representation, will increase that value to 116 bits. This small increase, while higher than that of the encryption key, is still many times smaller than that required for base 2 implementation. Separate fill generation also makes it easier to insert these characters into the original plaintext. It is for these reasons, that base 10 representation will be used in the implementation of the random fill placement methods.

Chapter 4 will take these choices and demonstrate how the programmed simulation will determine which method should be used in the hardware/firmware implementation of Chapter 5.

3.3 Encryption and decryption of the message

Once the plaintext has been modified, the tag must encrypt the data and transmit it to the receiver. The goal of this research is to reduce the encryption algorithm requirements in terms of security, therefore the more advanced cipher and hash function methods of encryption will not be used. The XOR encryption function will be used in the simulation of the placement methods. This simulation will also show if the randomization is sufficient enough that an attacker cannot determine the encryption key by comparing a series of transmitted ciphertexts.

There are two general forms of XOR encryption used by RFID tags, static key encryption and variable key encryption. If the tag data does not need to be changed or if the tag is designed such that it cannot be changed, the encryption can be done off the tag, then the resulting ciphertext is stored within the tags memory. While this form of encryption secures the data, the ciphertext does not change between transmissions, making it possible to track tags. The static encryption can also be performed on the tag itself, however it is not recommended for static keys. Static data provides a static ciphertext, therefore it would be a waste of computation processes to perform this encryption for each transmission.

When a tag generates its own ciphertext, it either generates a new encryption key for each transmission or chooses one from a list of prepared keys. While this method secures the data stored on the tag, and provides some privacy in terms of disassociation, the length of the message restricts the number of keys available to the tag. Using the example of the common EPC, a 96 bit key will provide a maximum of 2^{96} keys for the tag to use. If the tag generates a new key, the operational number matches the potential number of keys, otherwise the operational number is restricted to the number of keys stored in the memory.

The method proposed in this research combines the effect of a key's randomization with the modification of the plaintext being sent. Modifying the plaintext increases the potential number of ciphertexts in two ways. First, modifying the plaintext extends the length, thus extending the length of the key and increasing the potential number of keys. Second, modifying the plaintext with randomly generated fill increases the number of potential plaintexts. The result of increasing the length and potential number of ciphertexts increases the difficulty for an attacker to identify the original plaintext.

The number of ciphertexts generated by each advancing method are shown below compared to the number generated from the simple 96 bit key generation. Each graph shows the operational and potential ciphertexts generated as well as the potential number of 96 bit ciphertexts generated. The operational 96 bit ciphertext is not shown as that method only generates one ciphertext.

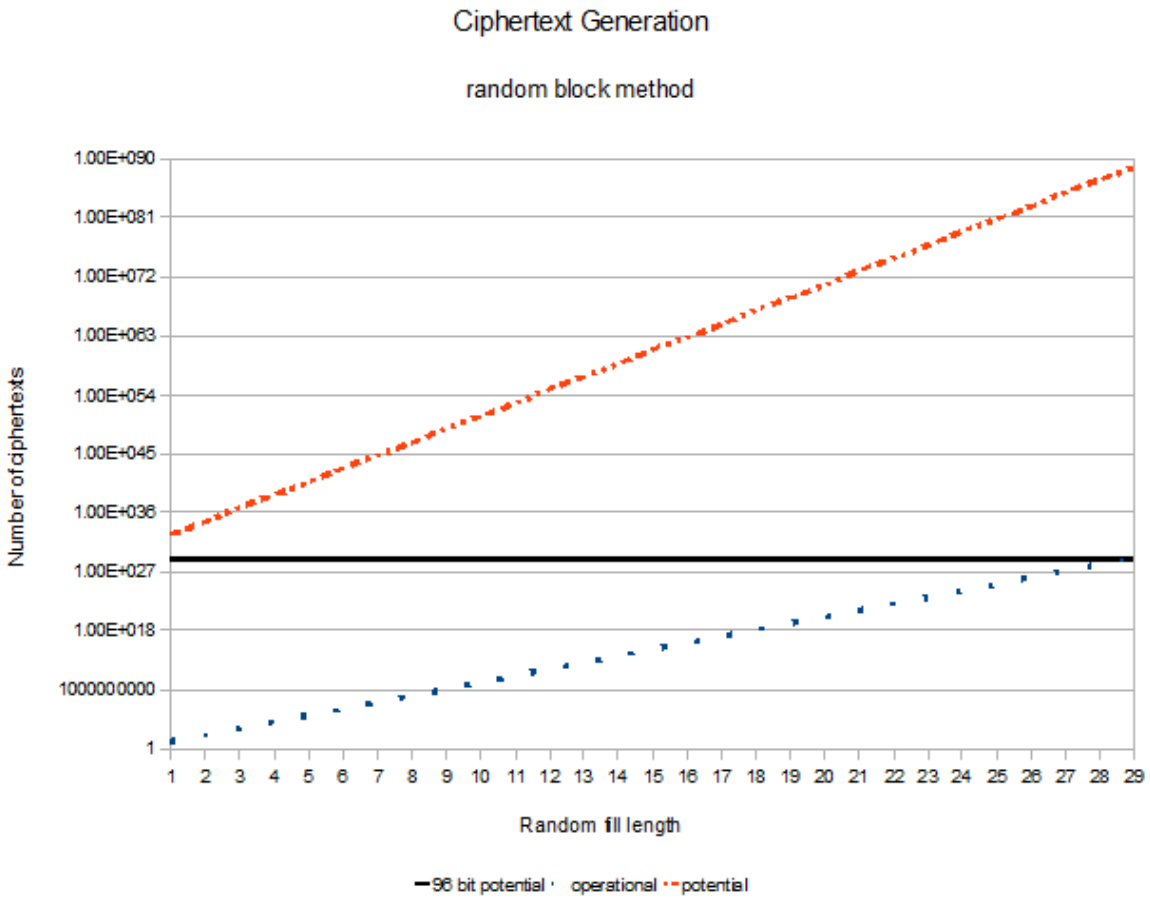

Figure 3.21: Ciphertexts generated by the random block method

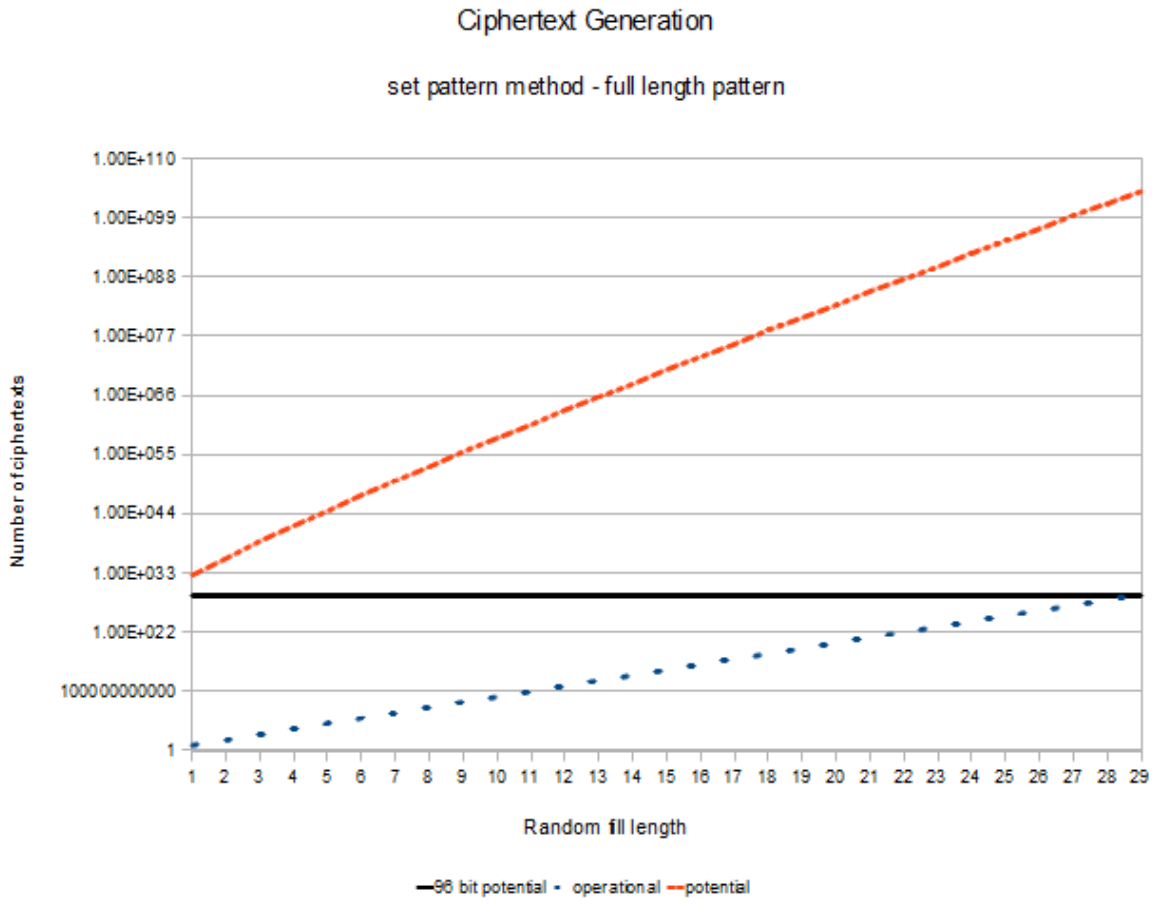

Figure 3.22: Ciphertexts generated by the full length set pattern method

The plaintexts generated by the full length random pattern method are double the length than those generated by the previous two methods. This results in a longer key length, thus a larger number of ciphertexts are generated.

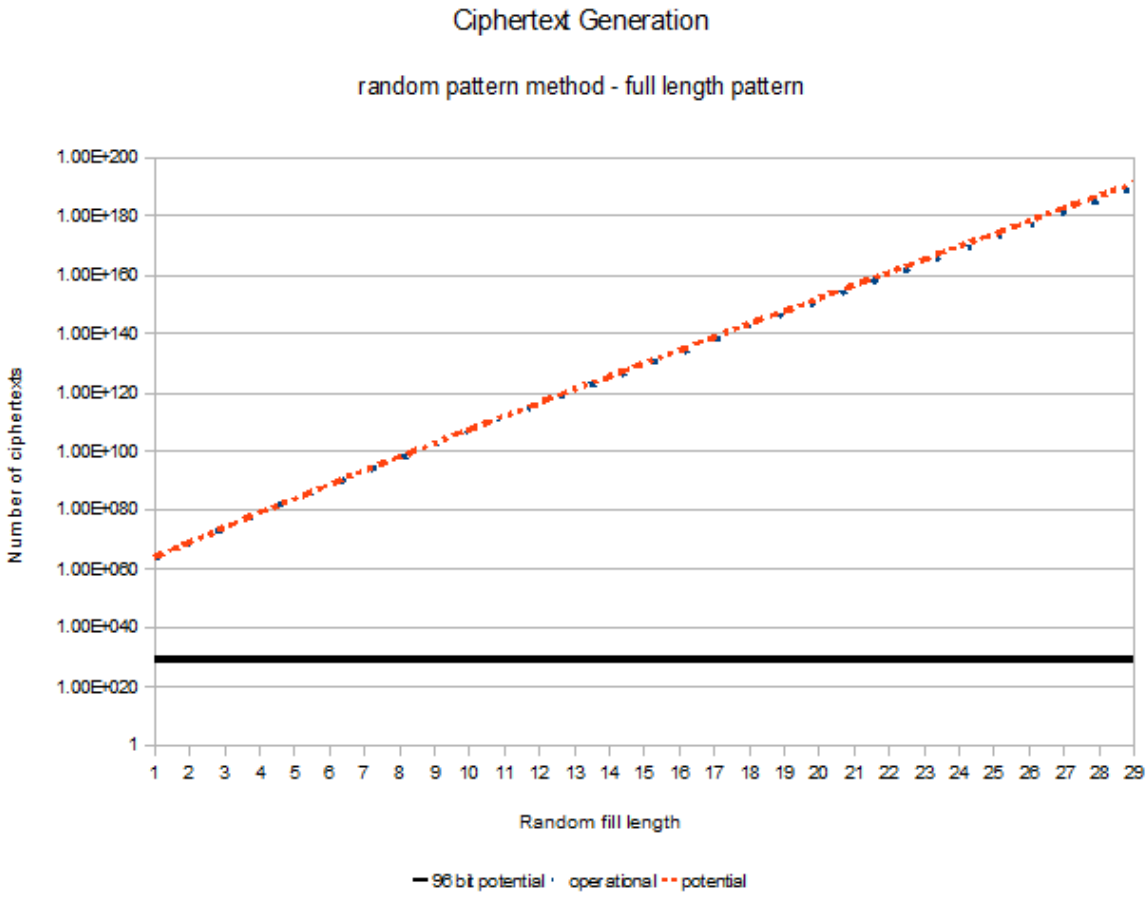

Figure 3.23: Ciphertexts generated by the full length random pattern method

3.4 Suggestions for tracking modified plaintexts

While modifying the plaintexts will help to disassociate the information from the tag, an attacker can still record the message and attempt to reuse it. This reuse is dangerous when the application secures locations or information. Tracking, in addition to pre-communication authentication, can help to reduce the effectiveness of these repeating attacks. The three methods of tracking proposed in this section use historic comparison and problem solving to determine whether or not a tag's modified message has been used recently. These methods all work on the assumption that the tag will not generate the same modified plaintext twice during an average period of time. The following subsections will describe the benefits and shortcomings of each of the three proposed trackers.

3.4.1 Historical filters

Historical filtering is the simplest method and is the logging of the last X received transmissions. While it is possible to save all transmissions, the time required to compare plaintexts will increase as the database size increases. Another reason to limit the size of the filter is the fact that the tag will eventually repeat the same plaintext. While the number of stored transmissions cannot be too large, it should not be too small, as it will become more vulnerable to flooding attacks. A flooding attack occurs when the attacker transmits a large number of saved communications with the goal of removing the true receiver history. With the list cleared, the attacker can use a saved communication, separate from the flooding plaintexts, to gain access.

Encoded Message	Time code
d	td
c	tc
b	tb
a	ta

Encoded Message	Time code
j	tj
i	ti
h	th
g	tg
f	tf
e	te
d	td
c	tc
b	td
a	ta

Figure 3.24: Flooding attack used to remove plaintext ‘a’

Two methods to reduce the effectiveness of these attacks would be to also filter the received transmissions by the unique tag identifier (which would be required to be stored in the additional data portion of the message) and to vary the size of the database with the frequency of communications. The second method will increase the comparison time during a flooding attack, however this outcome is better than a security breach.

3.4.2 Problem solving filters

Another method to deter flooding attacks would be to sort the information using the plaintext to solve a modulo based mathematical problem. This filter associates the result with a specific field in the database, meaning that the attacker would have to estimate the value used to separate the plaintexts into groupings to perform an efficient attack. The benefit to this method is that the divisor can change on a frequent basis to protect the list of received transmissions. This benefit can also become the filter's disadvantage as the list can shrink when recalculating the modulo value. Multiple plaintexts which resulted in unique modulo values using divisor X can be reassigned to a single modulo using divisor Y. This issue leaves gaps in the table which will allow any communication to occur since it has nothing to compare it to.

Encoded Message	Mod X
d	0
s	1
e	2
g	3
a	4
t	5
z	6
j	7
w	8

Encoded Message	Mod Y
e	0
	1
	2
a	3
w	4
t	5
	6
	7
	8

Figure 3.25: Gaps created by divisor change

While reducing the rate of divisor change will help reduce the gaps, it will also give the attacker a longer period of time to attempt to crack the pattern. A better solution is to use the third filter which is a hybrid of the two proposed methods.

3.4.3 Hybrid filters

The hybrid filter, as the name suggests, is the combination of the historical and problem solving filters. While this does increase the database size needed to store the extra data, both methods can help to reduce the effect of the other's vulnerabilities. The problem solving filter will help to reduce the flooding attacks which the historical filter are susceptible to, while the historical database can help fill in the gaps when the divisor is changed in the problem solving filter.

Encoded Message	Mod X
d	0
s	1
e	2
g	3
a	4
t	5
z	6
j	7
w	8

Encoded Message	Mod Y
e	0
o	1
	2
a	3
w	4
t	5
m	6
p	7
u	8

Memory - a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z .....

Figure 3.26: Hybrid database using both historical and problem solving

It must be made clear that the hybrid filter will not completely remove the chance that a flooding, or similar, attack will succeed. The goal of these filters is to reduce the effectiveness by increasing the difficulty as well as the effort required for the attacker to obtain a large list of plaintexts.

Chapter 4

Development and evaluation of a software prototype

A software simulation tool was developed to help determine which of the methods proposed in Chapter 3 performed the best. This evaluation was based on the largest number of random combinations available with the smallest impact on computation time.

4.1 Construction of the simulation

Simulating the plaintext modification will provide the duration of each process relative to one another as well as provide information on how these values will be displayed after encryption. While the methods described in Chapter 3 may generate a large number of plaintexts within a base 10 representation, the final encrypted message will remain in the base 2 representation. The simulated outcomes will show how the conversion process between the two representations will affect how well the proposed methods aid the tag in disassociating itself from the plain EPC value.

The simulation was programmed in the Java programming language and includes processes to convert the binary EPC into a base 10 representation, to modify this original value using a random number generator and the proposed placement methods, to convert the modified EPC back into a base 2 value, and finally to encrypt the data using a static encryption key and an XOR encryption algorithm. Each subsection will introduce possible options for each process, and will explain why one method's benefits outweigh the others.

4.1.1 Conversion and storage of the EPC

Several characteristics and processes must be set when creating the code to simulate the modification of RFID plaintexts. The most important decision is the representation of this plaintext as it affects the decisions on how to modify the data. The most common ways to represent integer values are either as a native integer, a string conversion of the integer value (i.e. Integer.toString() conversion within the Java programming language), or as an array of digits. The length of the integer restricts this decision as a positive signed integer ranges from 0 to 2^{31}-1, an integer with up to ten digits long. Replacing the integer with a long value will increase the length of a number. However, it is still too small to match the length of the standard EPC code (96 bits or an integer with 29 digits). A string will not restrict the length of the EPC and will provide a easy method of random fill insertion. While a string may be the easiest form of storage and modification, it does not provide a simple method of preparing the message for encryption and transmission. The last option, an array, provides the most flexible representation for EPC storage, modification, base conversion, and encryption.

$$1010111001_2 \Rightarrow 697_{10} \Rightarrow \boxed{6}\boxed{9}\boxed{7}$$

Figure 4.1: Representing a base 2 value as an array filled with base 10 values

Since each digit of the base 10 representation has a maximum value of 9, a byte array will suffice in storing the EPC. The first step of storing the EPC, converting the binary EPC to an array representation, should not be run on the tag itself as the tag does not need to have the ability to change the EPC. This does not imply that the EPC must be a static value since some tags allow for reprogramming after authentication. Converting a number from base 2 to base 10 simply involves adding the individual bit values together to form an integer.

$$10110_2 \Rightarrow (1)(2^4) + (0)(2^3) + (1)(2^2) + (1)(2^1) + (0)(2^0) = 22_{10}$$

Figure 4.2: Small value conversion from base 2 to base 10

This process is suitable for small numbers, up to the max value of the long data type, however the EPC is much longer than this range. A BigInteger data type could be used for off-tag processing which would then require separation into individual digits for

the array. The main question, when converting a number with these size limitations, is how the values will be represented when the desired values are larger than the maximum value supported by the processor. It would be possible, memory permitting, to store the array representation of each base 2 value such that each applicable array is added to the total sum which would represent the base 10 EPC.

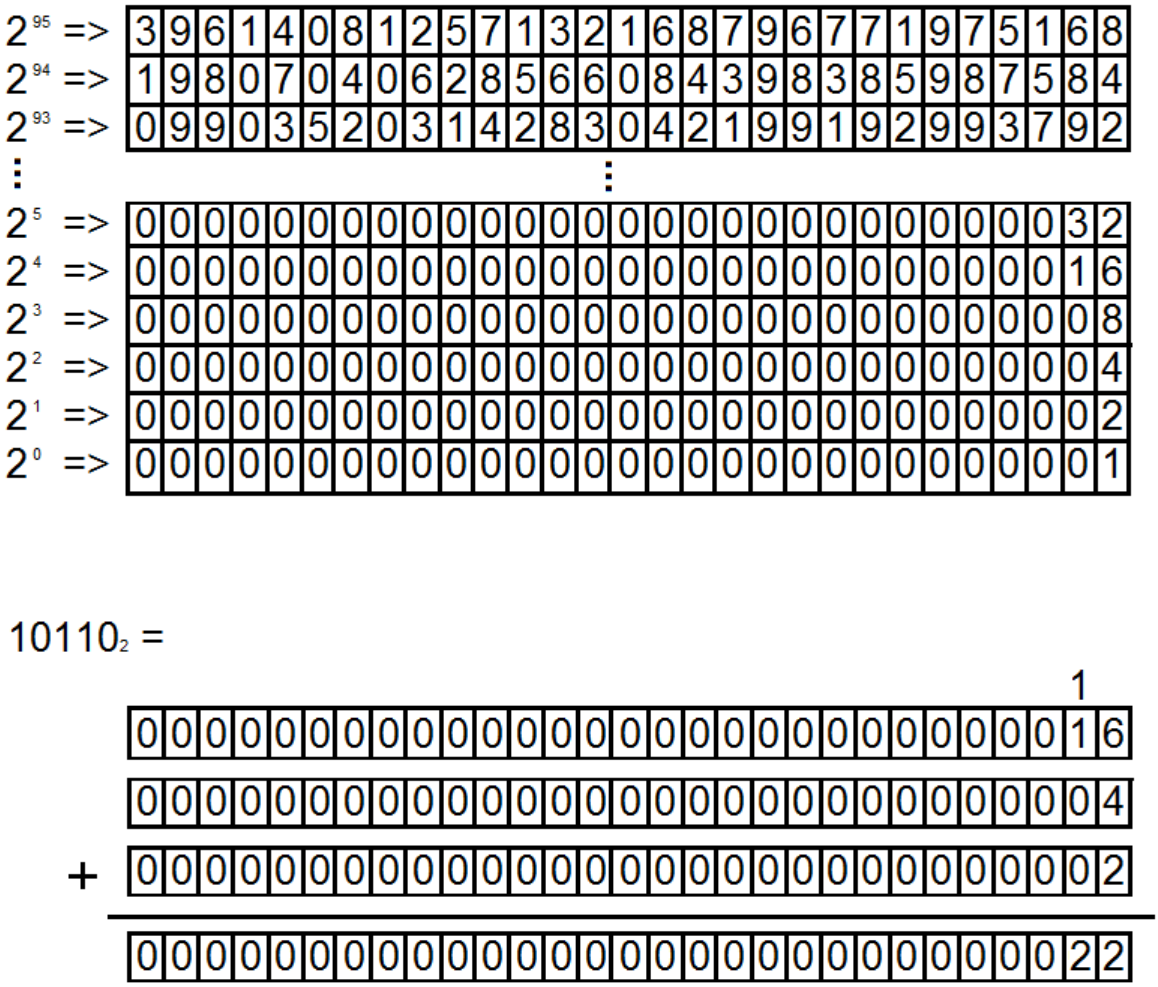

Figure 4.3: Adding stored arrays to form the base 10 EPC

The problem with this method is the memory required. When the goal of creating a RFID tag is to reduce costs, adding more memory to convert representations is counter-intuitive. There must be a compromise between memory requirements and process complexity such that the cheapest option is used. Since the EPC length varies in both the original and modified forms, the complexity should be raised to reduce the memory requirements. To save memory, the base 10 EPC value can be determined by the product of one repeated static array and a variable array.

$$
\begin{aligned}
2^{95} &= (2^{2})(2^{31})(2^{31})(2^{31}) \\
2^{94} &= (2^{1})(2^{31})(2^{31})(2^{31}) \\
2^{93} &= (2^{0})(2^{31})(2^{31})(2^{31}) \\
2^{92} &= (2^{0})(2^{30})(2^{31})(2^{31}) \\
&\vdots \\
2^{5} &= (2^{0})(2^{0})(2^{0})(2^{5}) \\
2^{4} &= (2^{0})(2^{0})(2^{0})(2^{4}) \\
2^{3} &= (2^{0})(2^{0})(2^{0})(2^{3}) \\
2^{2} &= (2^{0})(2^{0})(2^{0})(2^{2}) \\
2^{1} &= (2^{0})(2^{0})(2^{0})(2^{1}) \\
2^{0} &= (2^{0})(2^{0})(2^{0})(2^{0})
\end{aligned}
$$

Figure 4.4: Memory reduction by repeating calculations with the same value

Depending on the number of gates used and the amount of memory available on the tag, the variable array could either be generated on demand or by storing the reduced number of array values. Reducing the size of the set variable will increase the number of multiplication operations required to determine the base 10 value of the specific bit, however it will also reduce the number of variable arrays stored within the memory. Once a bit value is determined it is added to EPC array such that the total sum of all base 2 bits (in their base 10 representation) equals the base 10 representation of the EPC.

4.1.2 Modification of the EPC

The process of EPC modification depends on which of the proposed placement methods are used. There will be four versions of the simulation developed, one for each of the modification methods plus one control to use as a comparison tool to rank the processing times. Each version of the simulation, with the exception of the control, will utilize the tag's random number generator to provide the random fill for the EPC modification.

Random block placement

The random block placement method is the simplest of the proposals from chapter 3, and is subsequently the simplest to implement. If the position of the random block is static, then the majority of the placement work can be done during the tag's manufacturing process. Once the tag's base 10 EPC is ready to be stored within the tag, a gap can be left in the original array to accommodate the random block.

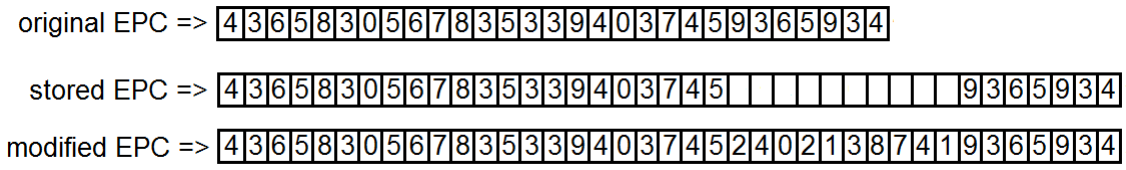

Figure 4.5: Creation and storage of the base 10 EPC prepared for the random block modification

If the random block's position is variable for each transmission, then the EPC array would be programmed into the tag intact and the values would be either shifted or copied into a separate array to await the conversion process back into a base 2 representation.

Set pattern placement

The preparation and storage for the set pattern placement is similar to the random block method, where gaps are left in the tag's stored version of the EPC. What differs is the process in determining where the gaps should be placed. One gap was required for the random block placement method, where as the set pattern method requires gaps to be spaced out in its static pattern. This pattern is generated during the manufacturing stage and can be stored within the tag's memory in the form of an array. The numbers within the pattern represent the number of original EPC characters to copy prior to inserting

a randomly generated value. The last value of the pattern represents the number of original EPC characters to copy prior to restarting the pattern if its length is less than the original EPC. For the sake of security, it is often preferred to maintain a non repeating pattern, however this may not be possible due to the memory requirements of the tag.

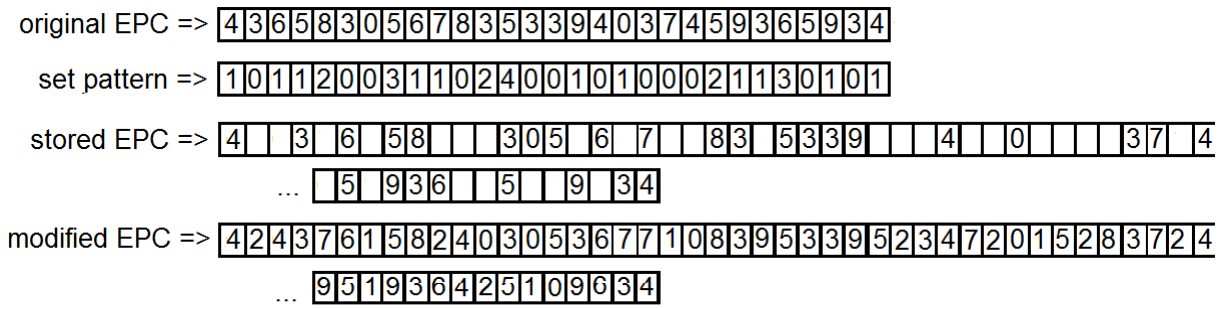

Figure 4.6: Creation and storage of the base 10 EPC prepared for the set pattern modification

Since a new pattern is generated for each tag, it is presumed that the receiver must know which tag it is communicating with, therefore it will know the pattern it must use to recover the EPC. If this characteristic is not desired by the system designer, then the pattern would need to be implemented with a shared identifier for the receiver to determine the pattern used. This method would not be recommended if this was the case as the static nature of the pattern may be revealed over subsequent transmissions. If the tag's identity and pattern cannot be preprogrammed into the secure network of receivers, then the next method of placement (random pattern placement) should be used instead.

Random pattern placement

The final method of modification is the random pattern. This method can be used to create the largest number of modified plaintexts, however it requires the most steps and compromises to be implemented. Like the set pattern, the random pattern placement method uses an array to direct the copying of original EPC characters and the generation of random fill. Unlike the set pattern, the random pattern placement method allows the RFID tag to generate a new pattern for each transmission. This affects which parties know the pattern, since the receiver or network of receivers cannot be preprogrammed with the pattern values.

To notify the receiver of the proper pattern such that it can recover the original EPC, the tag must include an additional identifier along with the modified message. This identifier can either be static or variable, where the static value would be decided during the manufacturing process, and the variable value could be generated by the receiver during the authentication process. The random pattern is then applied to that identifier and is attached to the newly modified EPC.

The length of the random pattern also differs from the length of the set pattern due to the necessity of compromising performance with security. The length can affect the tag‘s performance in two manners. First, the tag must generate this pattern for each transmission rather than have one stored within its memory. Second, if the pattern‘s length was identical to that of the EPC, then the special identifier must also be that length. This would result in a message four times the length of the original EPC and two times as long as the set pattern placement method. The computation time will increase with the length of the modified message as there are still two processes to alter the modified message prior to transmission, the conversion back to base 2 representation, and the encryption of the resulting value.

4.1.3 Conversion of the modified EPC

Once the random fill has been generated, that new plaintext must be converted back into the base 2 representation prior to the final encryption stage. Unlike the previous conversion, the base 2 bit placement is not known, which would require the calculation of each bit’s base 10 representation to determine if that bit should be stored as a 1 or a 0. If the reverse of the previous conversion method was used, the number of calculations required would make the modification process calculate results slower than desired. This is especially true when the calculations reveal that a bit should be stored as a 0, which would have made those calculations unnecessary in the original conversion process.

This inefficiency shows the need to develop a new conversion method to decrease the amount of time required to convert the modified plaintext from the base 10 to base 2 representation. Prior stages of the modification process have already split the base 10 representation into a series of single digits, this separation can be used to quicken the conversion process. A more efficient conversion process involves the addition of each character’s binary representation and multiplying each consecutive sum by the binary representation of 10.

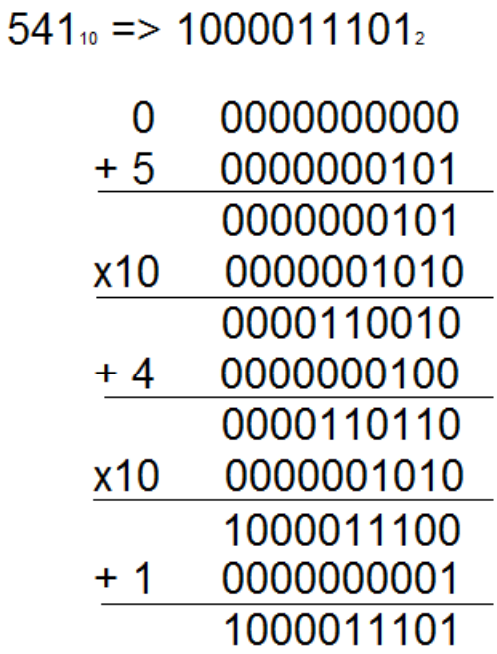

Figure 4.7: Conversion between base 10 and base 2 representation

4.1.4 Encryption of the modified EPC

The encryption process of the proposed protocol uses a XOR function with a static encryption key. Since base 10 implementation has the affect of varying the length of the modified message, the length of the stored key will be the maximum possible length of the array. As mentioned in chapter 3, this varying length affects the encryption process by reducing the number of key characters used or by revealing the first few key characters.

		XOR	cutoff XOR
encryption key	101101	000000	000000
plaintext A	001001	100100	0100
plaintext B	000110	101011	011
plaintext C	010101	111000	11000
plaintext D	110011	011110	011110
plaintext E	101010	000111	000111
plaintext F	001111	100010	0010
plaintext G	010011	111110	11110

Table 4.1: Comparison of XOR implementations

One of the many characteristics of encryption is that an attacker can gain useful information from the ciphertext which can be used to determine the tag's identity without direct decryption. A disadvantage to any of the base 10 representation is that the varying length can carry through to the base 2 representation after the conversion. Unless the values within the highest cells are close to their maximum values, the first few bits will equal zero. If the transmitted base 2 representation has its length set at the maximum length, those zeros in the highest cells will reveal the corresponding values of the encryption key. Limiting the length of the ciphertext to that of the highest one-valued cell of the plaintext will reduce the number of full length ciphertexts for the adversary to use. It is for this reason that the simulation will limit the encryption process to only use the number of key characters required to encrypt the modified plaintext.

4.2 Comparison of the different methods of modification

These simulations provide data, such as the computation time and randomness of subsequent transmissions, to determine which modification method is the best. Ideally the method which provides the highest number of possible modified plaintexts in the shortest amount of time would be considered the best method. While the simulation does not provide the exact computation time, due to the higher computation power, it should be able to provide a time ratio compared to other simulations.

One caveat to this ideal selection is the degree of randomness between each modification. If one modified plaintext can be associated to another plaintext, by means of pattern matching or static characters, that method of modification should be discarded.

4.2.1 Simulating the standard tag

Simulating the standard tag (i.e. without any plaintext modification) is simple, however it does provide a benchmark to compare the computation time of each placement method. Without a varying plaintext or varying encryption key the XOR results, as can be expected, are identical for each preparation of the ciphertext. The average simulation time, calculated over 100000 iterations, is 2021 nanoseconds.

To provide a better comparison to the placement methods providing varying ciphertexts, the standard tag simulation can be run such that it generates a new encryption key for each iteration. This ignores any time associated with sharing the new key during the authentication stage. The message lengths for the random block and set pattern placement methods will reach a maximum of 193 bits for this series of simulation, and the random pattern message can reach 386 bits in length. The following graph and its values will be used to compare these simulations with that of generating a new key for each encryption.

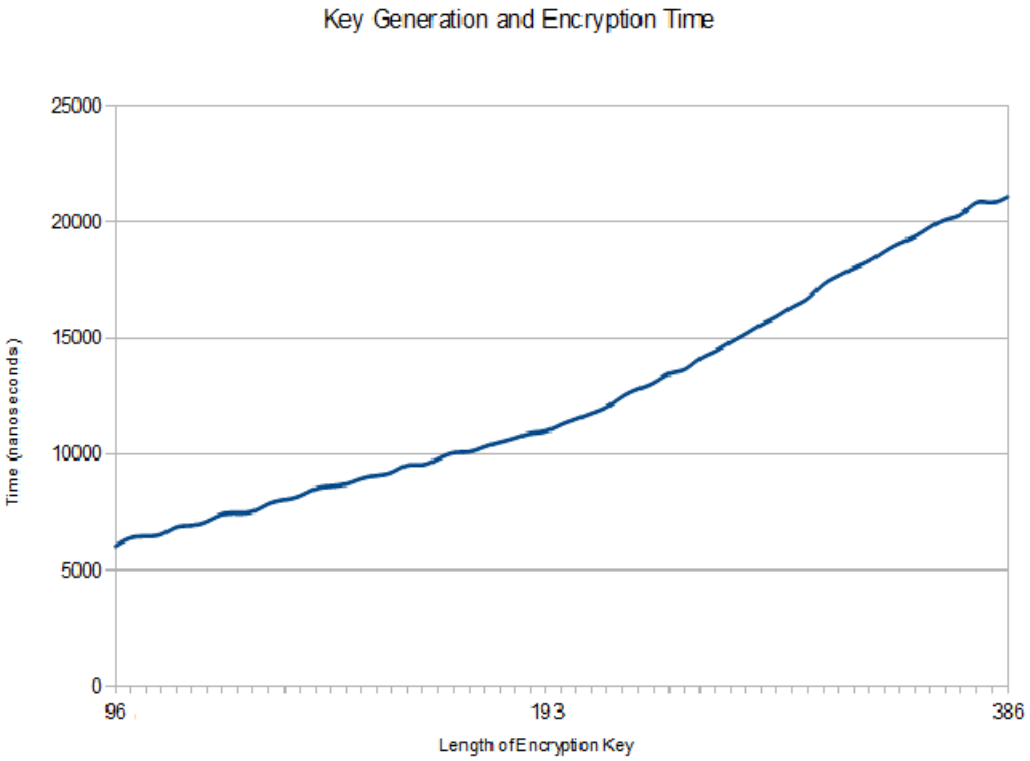

Figure 4.8: Key generation and encryption process duration

While these values provide a good benchmark to compare the random placement methods, the computation time of each process is expected to be smaller within the computer simulation when compared to those run on a RFID tag. Once the final method is selected Chapter 5 will discuss the development of a hardware prototype which should provide more accurate computation times.

4.2.2 Simulating random block placement

Running the random block simulation with a fill length of 29 characters resulted in an average simulation time of 81573 nanoseconds, however the focus of this simulation is how the location of the random block affects the randomness of the simulation. If the attacker were able to observe the base 10 representation after its modification, and prior to the conversion and encryption processes, the positioning of the random fill would be clear. This is made even easier with the random block placement due to the fact that the fill characters are grouped together. While the fill is random for each iteration, the remainder and majority of the message remains static. The simulation shows that this characteristic, to an extent, follows through to the base 2 representation after the plaintext's conversion and encryption. What is interesting, is how the positioning of the random block affects how many ciphertext bits remain static.

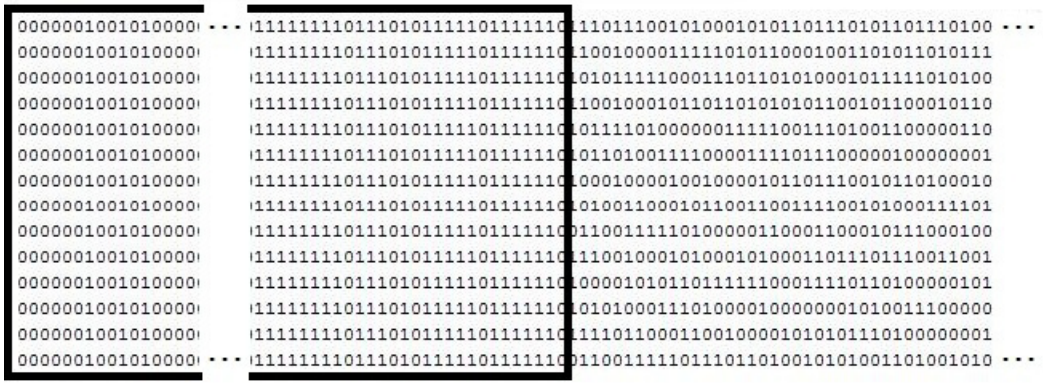

Figure 4.9: XOR results with random block plaintext modification, low placement

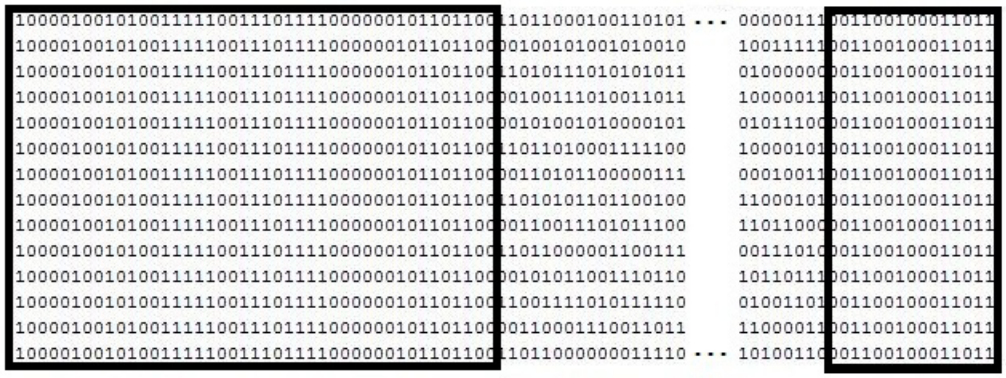

Figure 4.10: XOR results with random block plaintext modification, mid placement

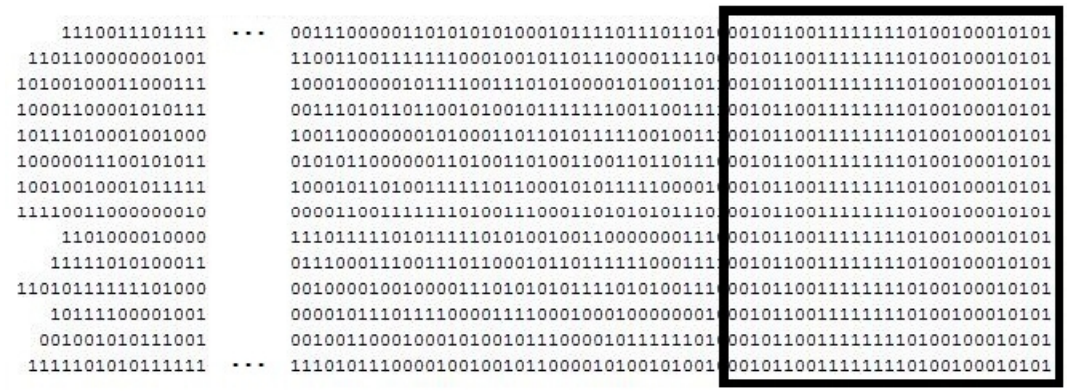

Figure 4.11: XOR results with random block plaintext modification, high placement

The position of the random block affected where and how many of the ciphertext bits remained static. As the random block's position increased in the array's cell structure, the number of static ciphertext bits decreased due to the magnitude of the fill. Placing the random block at the right end of the message (lowest value) meant that half of the ciphertext remained static. When the random block was placed in the middle of the plaintext, the number of static cipher text characters reduced to a third. Placing the random block as the highest plaintext value resulted in the lowest eighth of the message remaining static. This method is not recommended since there remains a static segment of the ciphertext which can be easily tracked.

4.2.3 Simulating set pattern placement

In addition to calculating the performance of this placement method, the simulation also showed whether or not this method shared the same static character issues of the previous method. Since the random fill pattern is set, the number of modified plaintexts available remain the same as the random block method, however the separation spreads this randomness over the entire message. Avoiding large blocks of static characters, especially near the higher end of the array, reduces and in some cases eliminates static values from appearing in each ciphertext.

Direct observation of the ciphertexts generated by the set pattern placement did not raise any concerns when using some of the larger fill lengths. The high end of the ciphertext began repeating itself for a small number of characters as the length of the random fill shrunk. This was expected to an extent as the segments of original

plaintext characters grew in length as the fill length shrunk. The ciphertext's cell values were summed as the XOR algorithm encrypted each modified plaintext to provide the percentage that a cell's value would be set as one. Ideally this percentage would be fifty percent for each cell, meaning that there was an equal chance that the cell's value could be set as a one or a zero. This equality provides randomness and makes determining a pattern difficult for an adversary recording multiple tag transmissions. An exception to this rule is when the cells are within the variable length portion of the ciphertext. The length of the ciphertext is affected by the length of the modified plaintext's integer representation. Any cell within this range with a percentage less than fifty percent will not provide a pattern to an attacker.

The simulations allowed for a +/- 2 percent tolerance before identifying a cell as a member of a repeating segment length. They also defined a repeating segment within the variable range to be three characters, in a consecutive set of five, with a percentage of over fifty-five percent. While there were exceptions, simulations using a fill length greater than 20 resulted in fewer repeating segments. The majority of these segments fell within +/-10 percent of the target fifty percent. As the fill length fell below that point, one hundred percent and zero percent cells began to appear more frequently and the repeating segment length grew. A commonality between all repeating segments was that they generally would be located at the higher value end of the ciphertext before stretching lower into the ciphertext as the fill length shrunk. It is important to note that an attacker would have to be able to identify, separate, and individually record a tag's transmission many times to begin to find a repeating segment. If the fill length remains above 20 with minor restrictions on the placement pattern, these segments should not be detectable.

In addition to testing the method's randomness, the computation time was also recorded during the simulation. The following graph shows the average time required to generate a ciphertext compared to that of the key generator.

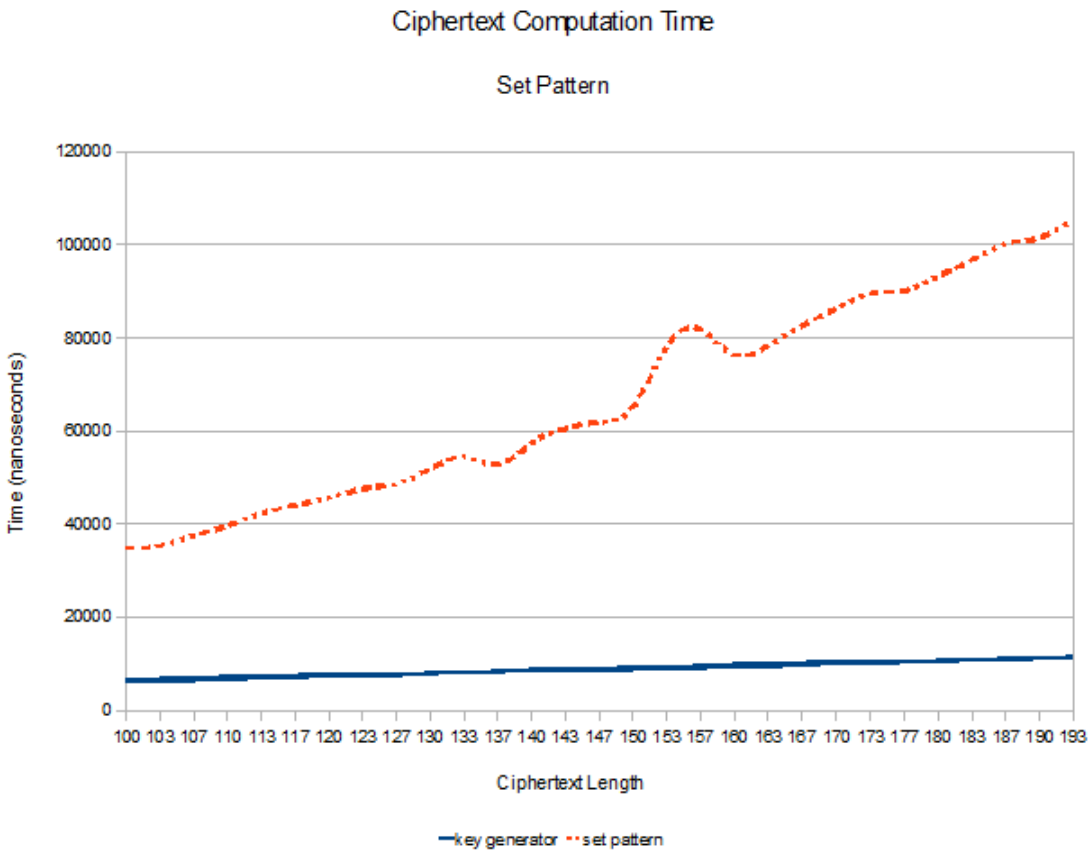

Figure 4.12: Computation time for the set pattern placement method

This time, while greater than the time set by the key generator, must be compared to that of the random pattern placement method before a decision is made on which method will be used in chapter 5.

4.2.4 Simulating random pattern placement

Simulating the random pattern placement method of modifying the plaintext provided comparable results to the set pattern method. Much like the set pattern, the random pattern created random ciphertexts with few repeating segments when using large fill lengths. The difference between the two methods was that the segment's length and repetition became visible when the fill length was as large as 28 characters, this can be attributed to the length of the ciphertext. Since the identifier must be equal to the length of the fill, the random pattern ciphertext has a maximum length equal to twice

the length of the set pattern ciphertext. As the ciphertexts become larger, the higher cell's impact becomes greater and the randomness of the ciphertext becomes weaker.

The simulation was also recorded to determine if the greater ciphertext length also affected the computation time. The following graph shows the random pattern computation time compared to that of the key generator.

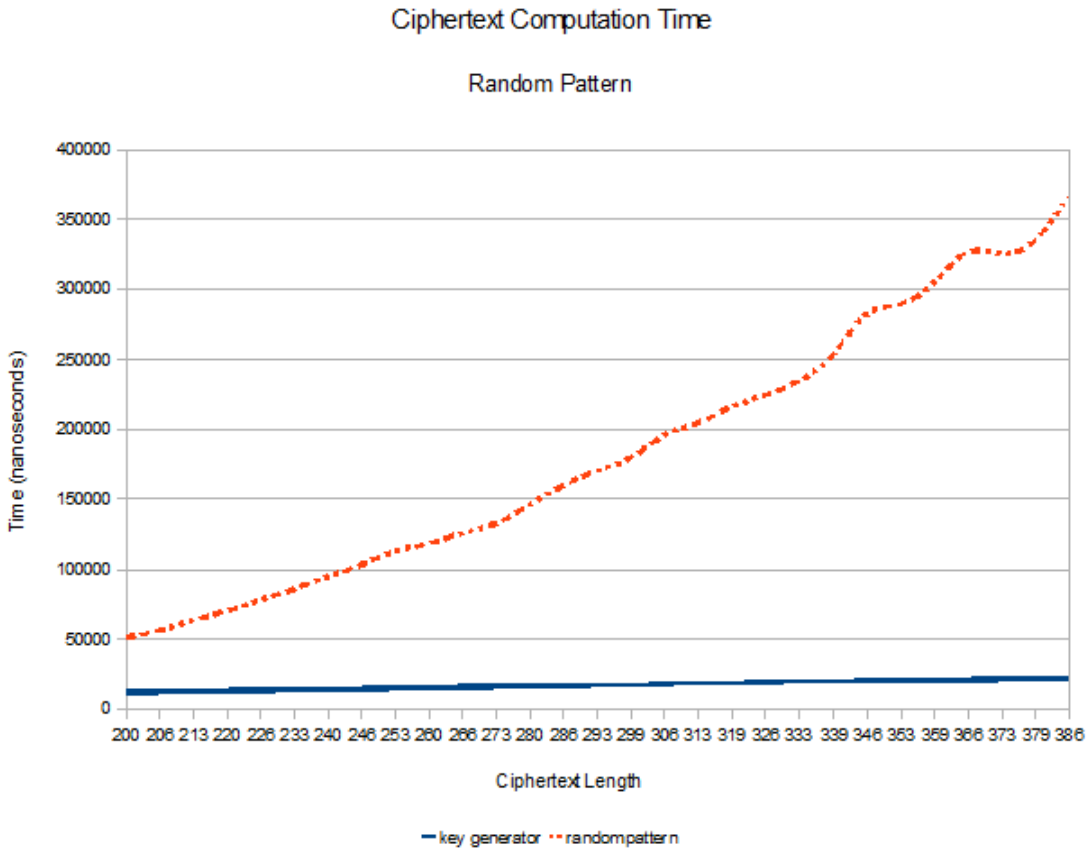

Figure 4.13: Computation time for the random pattern placement method

This graph shows that the difference in length of the ciphertext as well as the need to generate its own pattern makes the random pattern method much slower than the set ciphertext.

4.3 Placement method chosen for chapter 5

Three placement methods were simulated in this chapter to determine which of the three could benefit RFID tags the most with the lowest cost. The random block method allowed the larger EPC blocks to create repeating segments in each ciphertext generated. This marker makes it easy for eavesdroppers to detect and associate transmissions with one tag. For its lack of privacy, the random block method was excluded from the hardware prototype. While the random pattern method produced the greatest number of operational ciphertexts, it also suffered from repeating segments, large computation time, and large memory requirements. It is for these reasons that the random pattern method was excluded from the hardware prototype.

The set pattern method provides a modification method which contains the highlights of the two rejected placement methods. The set pattern method requires little memory to store its pattern: a pattern which can vary in length depending on the memory constraints of the tag. This pattern also has the option of being changed by the receiver providing that the the memory is not read-only. The computation time is also relatively low when compared to the random pattern method. These benefits make the set pattern placement method the only method which will be implemented in the hardware prototype.

Chapter 5

Development of a hardware prototype

This chapter discusses the development of a modification firmware which runs on a microcontroller and the results this firmware produces. Some RFID tags, such as the OpenBeacon Sputnik RFID tag, use a microcontroller to process the tag's different communication and encryption functions. The platform used to test the translated firmware is the Arduino Duemilanove microcontroller which is based on the ATmega328 microprocessor. For the purpose of this prototype, the pseudo-random number generator provided on the microcontroller was used to generate the random fill. The language used by this platform is an open sourced variation of C and C++.

The construction of the hardware prototype can be described as a series of development and testing phases. The first phase in creating the firmware was to translate the program from the Java simulation into the Arduino language. This translation will not be discussed as it is a simple substitution of Java functions to Arduino equivalents. Since the firmware is limited to the set pattern placement method, it is assumed that the values of the non-random characters will remain static, therefore the plaintext will be stored on the tag in its modified integer form. This means that the conversion from base 2 to base 10 will not be included in the firmware. The second phase in developing the firmware is to optimize the code such that it compromises between the speed of operation and the memory space required. The final phase in developing the firmware is to compare the results with the goals set in the first stage. If these goals are not met, then an alternative approach to modifying the plaintext must be found.

5.1 Optimization of functions

To optimize any type of program, the goals and limitations of the platform it is running on must be set. The timing goal of this firmware was to modify and encrypt the plaintext within twenty milliseconds. This time was chosen to meet both the original timing scheme and the interleaved scheme proposed by [13]. The memory goal was to use as little memory as possible to reduce the cost of manufacturing if this program is converted from a firmware to a logic gate representation. With respect to firmware generation, the timing goal will outweigh the memory goal in decisions where one optimization will only better one or the other. The goals also affect how the limitations are set.

Limiting firmware characteristics such as the random fill length will aid in meeting the goals set in the previous paragraph. Ideally the modifying process would use as long a fill length that the memory would allow, however this is unrealistic and is counterproductive to the memory goal and will not satisfy the speed goal. Since the thesis is based on the 96 bit EPC, the maximum integer representation length will be restricted to 29 characters. The remainder of the development process was done with a random fill length of 29 characters to ensure that the goals met the maximum modification requirements.

The original translation resulted in six integer arrays to store the data being used in the modification process. Three of these arrays (referred to as binary arrays) were the length of the maximum binary representation of the modified plaintext. These binary arrays were used to store the modified plaintext and later the ciphertext, a copy of the plaintext used during the modification stage, and the encryption key. The fourth and fifth arrays were used to store the integer representation of the modified plaintext and the set placement pattern respectively. The final array was a 2 dimensional array which stored the binary representations of each possible numerical character (0-9). Table 5.1 demonstrates the character lengths and corresponding memory requirements :

Array	Data type	Character length	Bits
Encryption key	int	193	6176
Modified binary plaintext	int	193	6176
Copy of modified binary plaintext	int	193	6176
Modified integer plaintext	int	58	1856
Pattern	int	29	928
Binary character representation	int	40	1280

Table 5.1: Original memory for function arrays

This does not include the single variables used as counters and mathematical carry values during the modification operation. The content of each array must be considered when optimizing the memory. Each cell of the binary arrays and the binary character array, by definition, can either have a value of 1 or 0. Replacing these arrays with boolean arrays and adjusting the code to suit will reduce the memory requirements. It is important to note that while a boolean value is only one bit, the Arduino stores it as a full byte. The remaining integer representation and pattern arrays cannot be stored as a boolean value, however since the maximum cell value is less than 2^8, the data arrays can be stored as unsigned byte arrays. The optimized arrays are represented by the following chart.

Array	Data type	Character length	Bits
Encryption key	boolean (8 bit)	193	1544
Modified binary plaintext	boolean (8 bit)	193	1544
Copy of modified binary plaintext	boolean (8 bit)	193	1544
Modified integer plaintext	byte	58	464
Pattern	byte	29	232
Binary character representation	boolean (8 bit)	40	320

Table 5.2: Optimized memory for function arrays

In addition to optimizing the memory, the speed of the original translation was slower than the goal set out in the first stage of the firmware development. It is worth noting that changing the data types of the arrays not only reduced the memory requirements, it also reduced the operating time of the modifying and encryption functions. The modification and encryption process would take up to 39 milliseconds to complete in the original translation; altering the data types reduced this time to under 23 milliseconds. This value was still greater than the goal of 20 milliseconds, so further alterations to the operation itself were required. The addition of if statements to reduce the overwriting of cells with the identical value further reduced the duration of the functions to 17 milliseconds. Since this goal is met with the maximum fill length, reducing the fill length should also reduce the operating time. This timing along with the randomness of different fill lengths is tested in the next section.

5.2 Speed and randomization results

The simulation in Chapter 4 provided a ratio of operation times when comparing one randomization method to another, this firmware test will provide more accurate operation times for the set pattern placement method. In addition to the operating time ratio, Chapter 4 also provided a minimum random fill length. The average increase in operating time for each random fill character added is six tenths of a millisecond. This trend can be seen in the following figure:

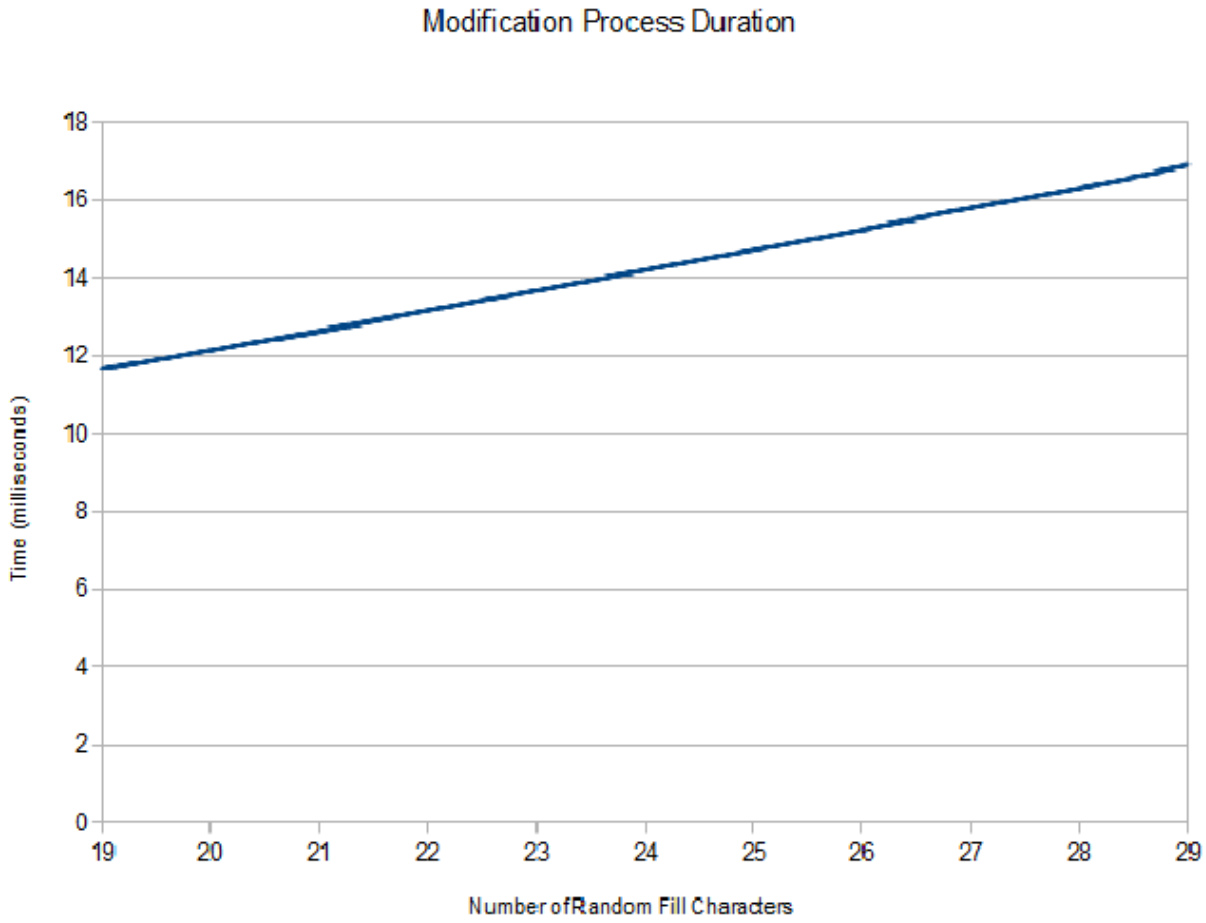

Figure 5.1: Comparing operation times to the random fill length

These times can also be compared to the time required to generate the ciphertext with a fill length of zero (3.5 milliseconds). This represents the time required to generate a static ciphertext from the same inputs.

The method in determining randomness was also discussed in Chapter 4. Multiple EPC and pattern values were generated to test this randomness with the hardware prototype. Each EPC was paired with a single pattern which was used to modify and encrypt the given plaintext ten thousand times. The characteristics of four tags will be further described and their pattern's effect on the ciphertext will be explained.

The EPC integer representations were generated using the Arduino's random number generator and were restricted to the maximum length of 29 digits. The final EPC values are shown in the following table.

Tag	EPC
A	13857171439284561290046983849
B	18810910594313668340395076599
C	12261209955999467056310093988
D	13182148802321680428568305371

Table 5.3: EPC values for tags A, B, C, and D

The 29 character patterns were also generated using the Arduino's random number generator. To avoid large blocks of static cells, as identified in Chapter 4, the generator limited the number of consecutive static characters by forcing the random number generator to regenerate the random character's placement once a cell location was duplicated. Regenerating a position has a higher chance of reducing consecutive static characters when compared to moving that fill character to the next available position. The randomly generated patterns are shown in table 5.4.

Tag	Pattern
A	3 5 6 9 12 14 15 16 17 20 22 23 24 26 29 30 34 35 40 41 43 44 46 47 49 52 53 54 55
B	0 1 2 8 9 12 13 15 17 19 20 21 22 25 33 34 35 36 40 41 42 44 45 47 49 50 52 53 54
C	2 3 4 5 6 7 8 9 14 17 18 22 23 24 27 31 33 34 35 37 39 41 42 44 48 49 50 52 55
D	0 1 3 4 6 7 10 11 12 16 19 25 28 33 34 35 36 37 40 41 42 43 44 46 51 52 53 55 56

Table 5.4: Pattern values for tags A, B, C, and D

Once the plaintexts and patterns were generated, they were modified and encrypted 10000 times. The following cell values were observed.

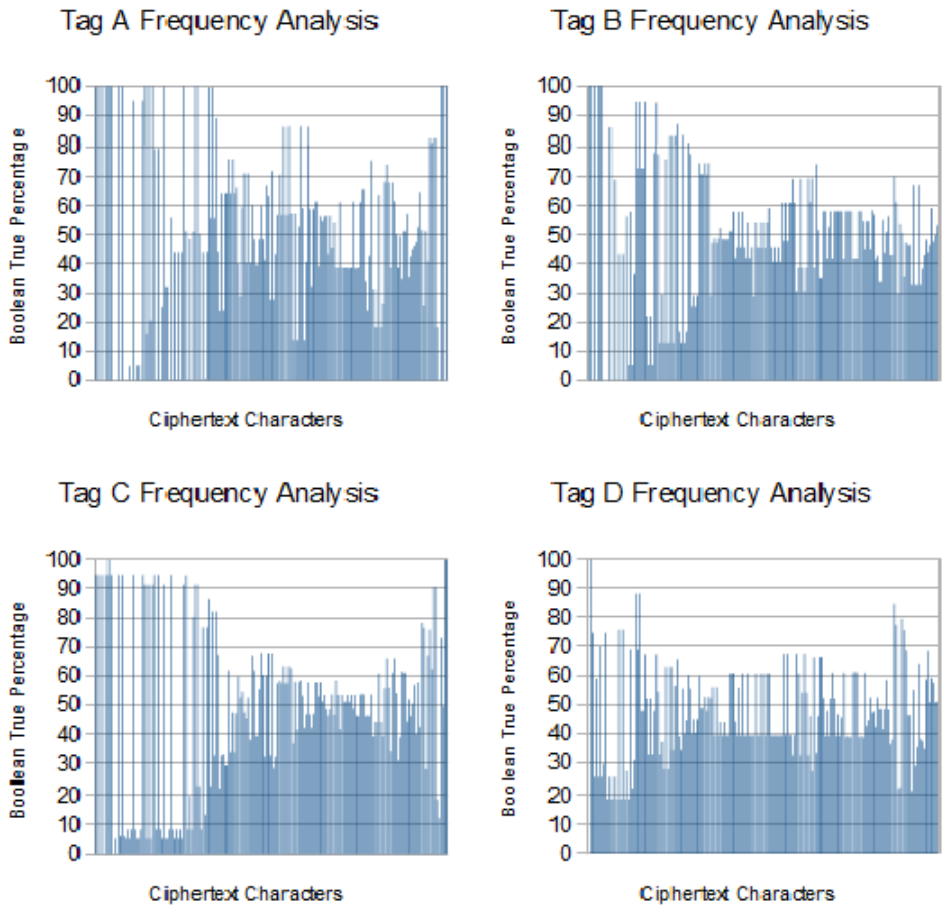

Figure 5.2: Boolean true frequency for tags A, B, C, and D

Each line represents a single cell and the likelihood that its value will be equal to one (boolean true). As explained in Chapter 4, the closer a tag's character frequency is to fifty percent, the harder it is to associate that tag with each transmission. As expected, some commonalities appear between the hardware and software prototypes, such as the grouping of repeating characters and the frequency extremes near the higher valued cells. The areas of each graph which has a frequency closer to fifty percent can be tied to those portions of the pattern which have a greater number of fill characters within one area of the modified plaintext. A similar statement can be said for groups of static characters; these groups can lead to the peaks and valleys observed in the above

graphs. The pattern's effect on the frequency of boolean true characters can be shown by comparing the frequency analysis of the same tag using two different patterns. Tags A and C are used in the following example where the first pattern is the corresponding pattern found in table 5.4 and the second pattern is :

Shared Pattern Value
0 4 5 6 9 11 13 15 20 21 23 27 30 31 32 34 36 38 41 42 43 44 47 48 50 52 53 56 57

Table 5.5: Shared second pattern value

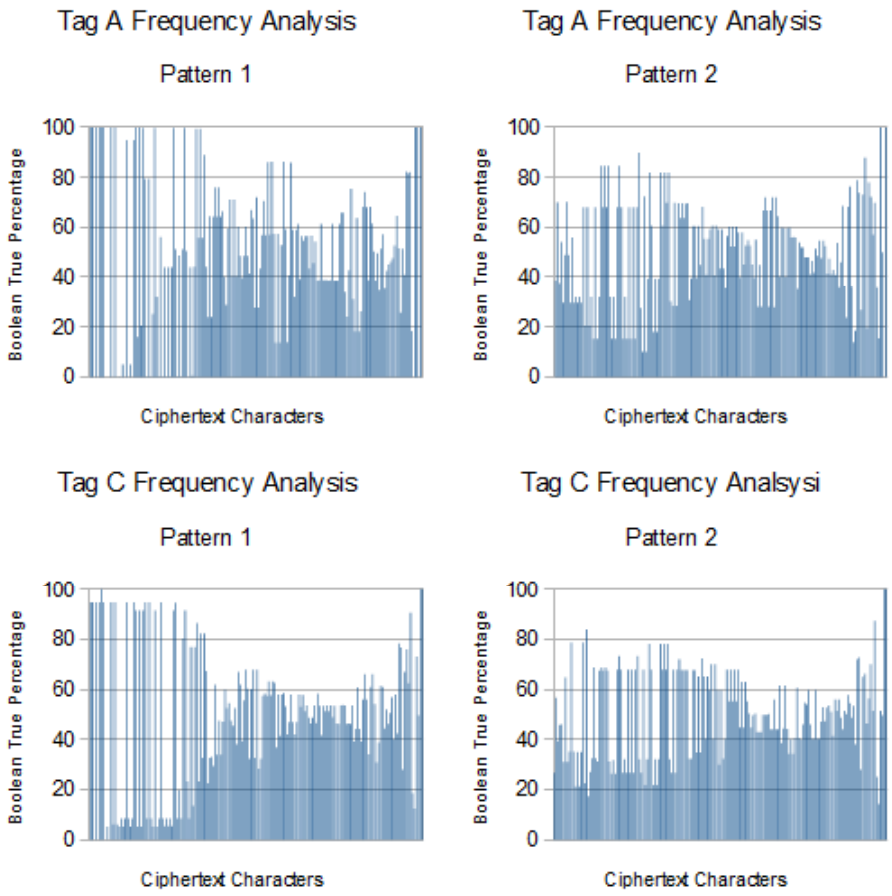

Figure 5.3: Frequency Analysis for tags A and C with dual patterns

This information can only undermine the modification process if the adversary has access to a sample size of this magnitude. For an adversary to gather a large enough set of usable tag transmissions, it must be able to identify each tag and separate each transmission from other tags in the system. This creates a circular logic problem for the attacker as it can identify a tag from a set of transmissions, however to gather that sample it must identify a tag. The number of tags in a system further distract an adversary from associating transmissions to specific tags. The average frequency of occurence over the four tags described is shown in the following figure.

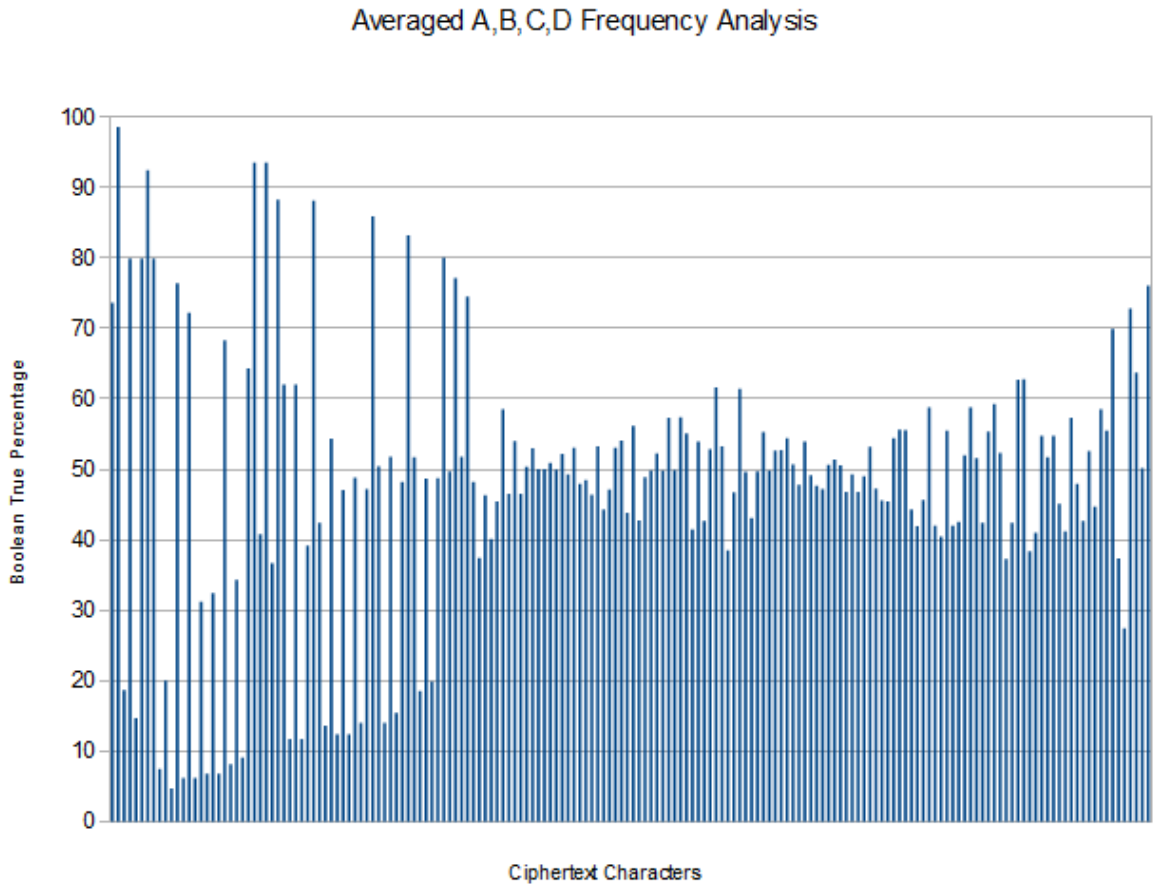

Figure 5.4: Average boolean true percentage for tags A, B, C, and D

The average of tags A, B, C, and D show that a collection of tags will help hide each other's peaks, valleys, and flat portions of the pattern. This averaging of boolean true percentages represents the database of recorded transmissions the attacker must compare to the individual transmission to determine a pattern. Since the average cell percentage

is also close (and sometimes closer than an individual cell percentage) to fifty percent, it increases the odds that an unrelated tag will be falsely identified as the same tag the attacker is targeting. The averaged values for each cell travel closer to fifty percent as the number of tags within the system increases. This is shown in the following diagram where 100 unique plaintexts are modified with its own set pattern.

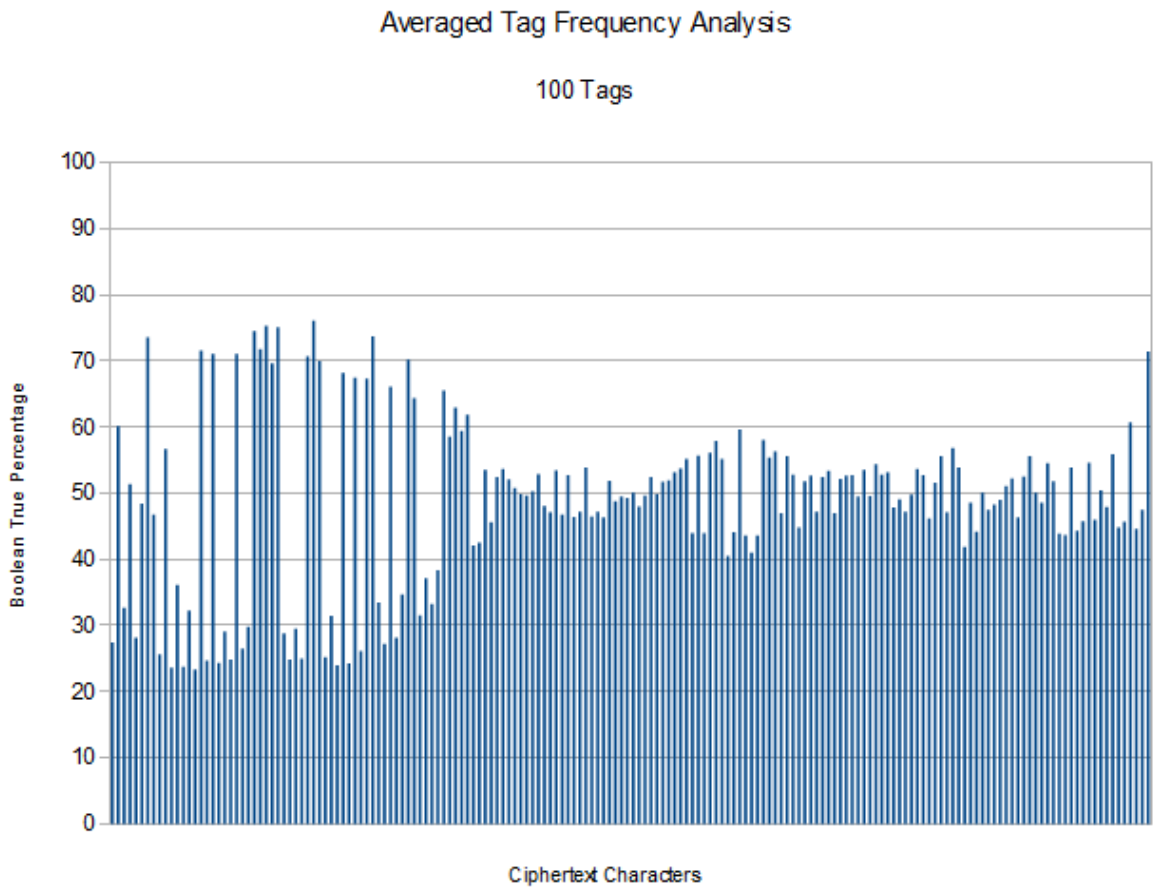

Figure 5.5: Average boolean true frequency for one hundred tags

The same 100 tags were also tested with the shared pattern listed in table 5.5 to determine the impact to the tag average of individual patterns versus the tag average using a shared pattern. The purpose of this comparison is to determine whether or not a shared pattern, designed to minimize frequency peaks and valleys, is required to secure the anonymity of these RFID tags.

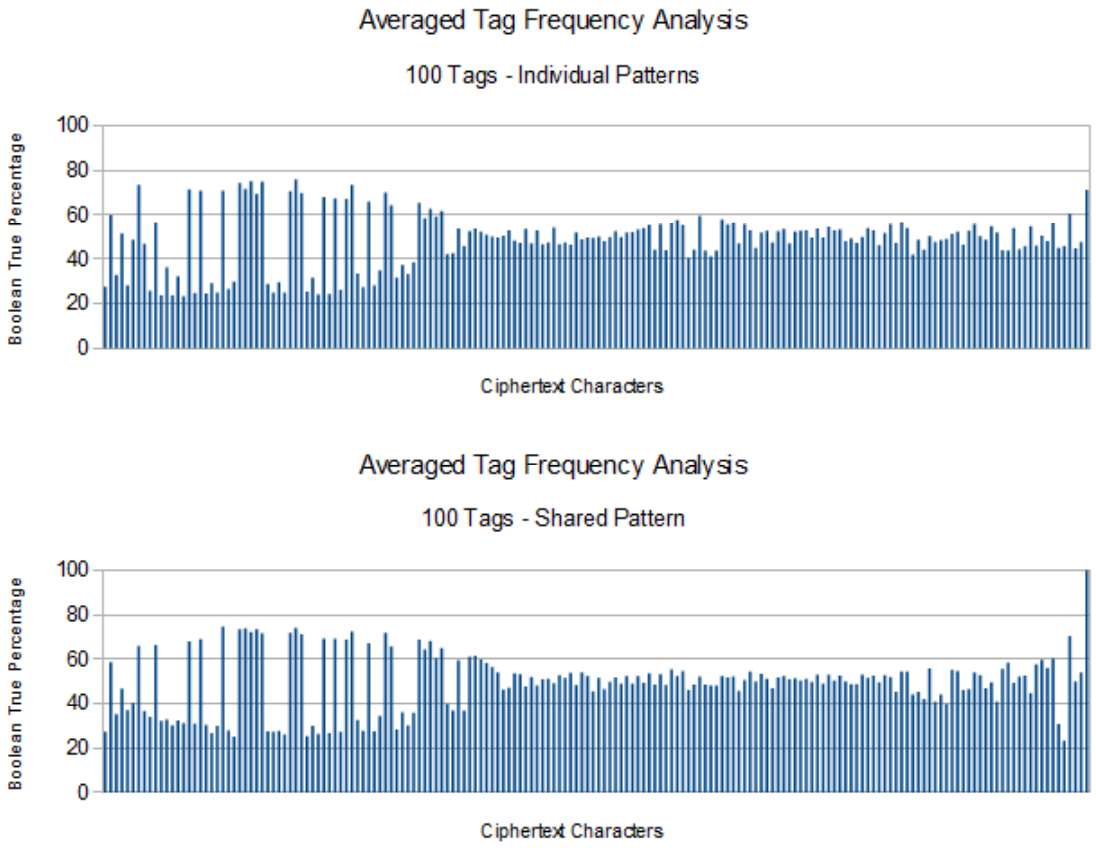

Figure 5.6: Pattern impact on the average boolean true frequency for one hundred tags

Figure 5.6 shows that a shared pattern has little positive impact on the average frequency analysis. The benefit of a shared pattern is outweighed by the fact that this method provides a larger sample set of a single pattern for any adversary to use. Increasing the number of patterns provides an increased number of sample sets as well as reducing the size of these samples thus increasing the difficulty in determining the pattern used by each tag.

The previous figures were generated using a EPC length of 29 and a pattern length of 29. This let the graphs show how the boolean true frequency is affected by the position of the pattern and the values of the random fill characters. The next concern is the length of pattern (length of the random fill) and how it affects the frequency of occurence. The following diagrams show how the pattern appears with an EPC length of 29 and a pattern length of 0, 9, 19, 29, and 39.

Figure 5.7: Frequency analysis for fill length smaller than the EPC length

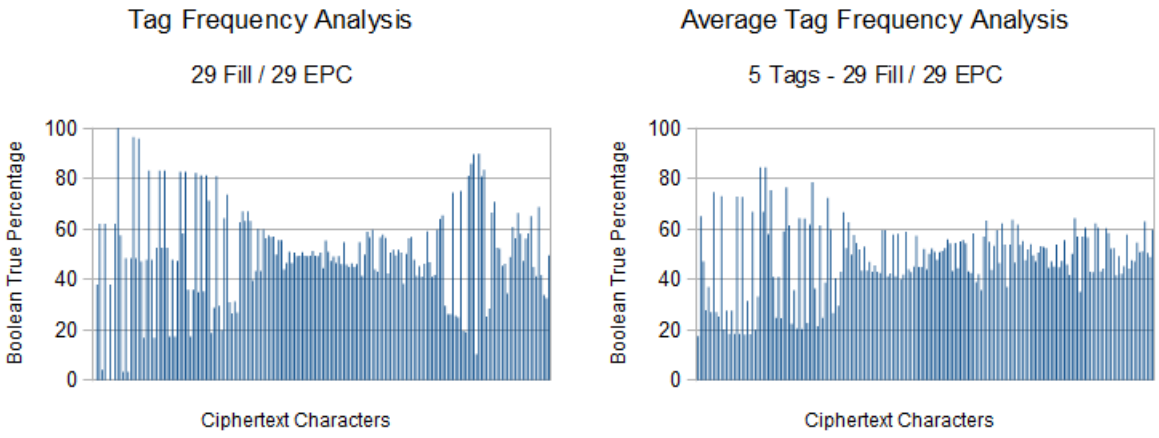

Figure 5.8: Frequency analysis for fill length equal to the EPC length

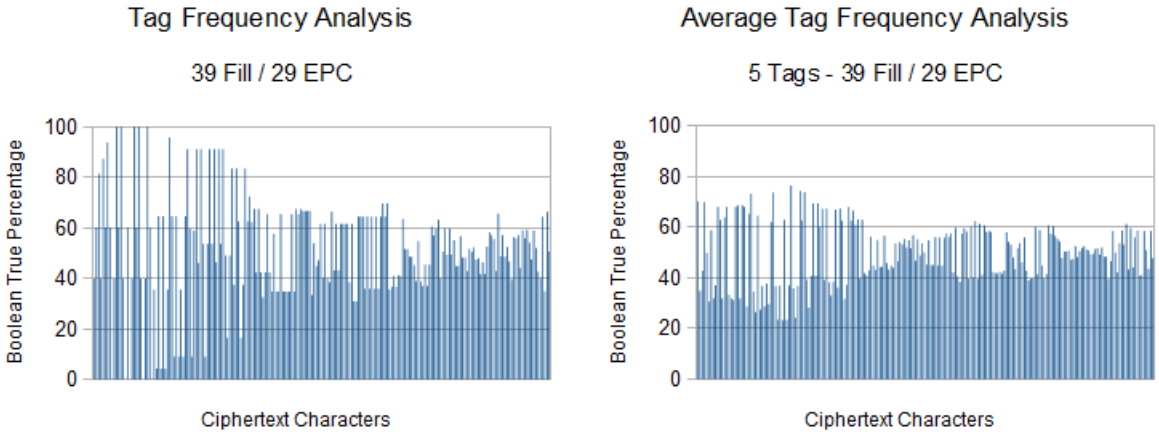

Figure 5.9: Frequency analysis for fill length greater than the EPC length

Much like the simulations, the figures generated by the hardware prototype show that by adding a random fill helps to reduce the static nature. Figure 5.7 shows that increasing the length of the random fill brings the individual boolean true percentage of each ciphertext character closer to 50 percent. While averaging the tags with smaller fill lengths provides similar frequency analysis to that of a tag with larger fill lengths, the 100 percent peaks and 0 percent valleys of the individual tag transmissions provides a tool for an attacker to use. If there are a large number of these peaks, they can be used to separate tag transmissions into smaller groups. Small groups of tags will makes it easier for an attacker to identify a tag from segments of a transmission.

Figures 5.8 and 5.9 show that there is little difference between a fill length of 29 and 39 characters. This means that a tag using the set pattern placement with a random fill length matching that of the EPC will provide sufficient anonymity. The hardware prototype of the set pattern placement method appears to match the results of the software prototype. The computation time and randomness of this method meets the goals set in the development of the protocol and of the prototypes. This method of message modification provides an alternative to larger encryption techniques while maintaining a random ciphertext without synchronization.

Chapter 6

Conclusion

6.1 Summary

The goal of this thesis was to develop an alternative secure method of protecting a RFID tag. The majority of the research, development and testing was on plaintext modification prior to the encryption phase. Summarizing previous work showed that many groups attempted to secure the data on RFID tags by increasing the complexity of the encryption algorithm or by increasing the synchronization requirements with the receiver. This research also provided the maximum time available to encrypt the tag data: twenty five milliseconds for a UHF tag.

The XOR encryption algorithm was chosen and used as a comparison tool while developing multiple plaintext modification methods. Four modification methods were initially proposed: the random block placement method, the set pattern placement method, the random pattern placement method, and the indexed placement method. Each proposed method was compared against each other as well as the basic XOR encryption algorithm to determine which method best met the timing requirements. The number of ciphertexts generated by each method was also used as a comparison tool when choosing the best placement method. This comparison determined that the set pattern placement method provided the largest number of ciphertexts (10^{29} operational and $7.42*10^{103}$ potential) within 17 milliseconds.

The hardware prototype generated a boolean true frequency map for each cell of the ciphertext. This map showed the frequency that a cell's value would be equal to one over a series of ten thousand modifications and encryptions. The maps verified the simulation

data and showed that the length of the random input should be within ten characters, if not equal to, the length of the original plaintext. When the fill length dropped below that point, the frequency map showed that a larger number of cell values could remain static. The limiting factor for the plaintext's length is the multiplication component of the base converter. The converter changes the modified plaintext's representation from base 10 back to base 2. While the multiplication was limited to the filled cells, each additional plaintext character would require all of the previously converted cells to be re-calculated. Despite the limitation, the set pattern placement method can secure the 96 bit plaintext which is common among all EPC variations.

This method of plaintext modification showed that it is possible to generate multiple ciphertexts using XOR encryption without requiring constant synchronization between the RFID tag and the receiver network.

6.2 Future work

Future work on this topic could include further study into the generation and distribution of the set pattern. The current method of pattern generation is not dependant on the value of the tag's EPC. Generating a pattern using randomly generated positions can cause larger groups of static cells, thus increasing the boolean true percentage in the frequency analysis of that tag. Restrictions can be placed on the pattern generator to prevent these groups from existing, however that also reduces the number of operational ciphertexts generated by the set pattern placement method. Further research could determine if the analysis of the plaintext value would have a beneficial impact on the pattern. An example of this impact could be that the cell value dictates the pattern placement such that it minimizes the frequency map's peaks and valleys. While this would still reduce the operational ciphertexts, it should provide greater anonymity than a general placement restriction. Since the pattern is a set value in the tag's memory and not generated by the tag itself, the processing required to optimize the pattern used would not affect the tag's modification/encryption performance.

Further research may also reveal a method to decrease the computation time of the modified ciphertext. This decrease in computation time would allow longer plaintexts to be processed within the time restrictions. In addition to decreasing the computation time, rescheduling computation to occur while the tag is not communicating with a receiver could also be considered. This process would either require an independent power source

or for the tag to remain in the reading range of a receiver in a non-transmission state.

Security analysis could determine whether or not the distribution of patterns must be repeated to ensure the security of the RFID tag's data. This would involve testing the brute force required to observe patterns from a list of recorded transmissions. It is suspected that the combination of the encryption algorithm in conjunction with the random modification would provide too many false positives to determine which patterns are legitimate.

As the use of RFID tags become more common, the security and convenience of the tag must match the user's expectations. The results found in the development of this thesis present a new method in which to provide this necessary security.

References

[1] 13.56 mhz ism band class 1 radio frequency identification tag interface specification: Candidate reccomendation, version 1.0.0. Technical report, Cambridge, Maine, 2003.

[2] A. Alimohammad, S.F. Fard, B.F. Cockburn, and C. Schlegel. On the efficiency and accuracy of hybrid pseudo-random number generators for fpga-based simulations. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, april 2008.

[3] A. Arbit, Y. Oren, and A. Wool. Toward practical public key anti-counterfeiting for low-cost epc tags. In *RFID (RFID), 2011 IEEE International Conference on*, pages 184 –191, april 2011.

[4] G.K. Balachandran and R.E. Barnett. A 440-na true random number generator for passive rfid tags. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 55(11):3723 –3732, dec. 2008.

[5] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on aes variants with up to 10 rounds. Cryptology ePrint Archive, Report 2009/374, 2009. `http://eprint.iacr.org/`.

[6] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full aes-192 and aes-256. Cryptology ePrint Archive, Report 2009/317, 2009. `http://eprint.iacr.org/`.

[7] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and related-key attack on the full aes-256. Crypto 2009, 2009.

[8] D. Cartasegna, A. Cito, F. Conso, A. Donida, M. Grassi, L. Malvasi, G. Rescio, and P. Malcovati. Smart rfid label for monitoring the preservation conditions of food. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1161 –1164, may 2009.

[9] Tom St. Denis. *Cryptography for Developers*. Syngress Publishing Inc., Rockland, Maine, 2007.

[10] M.J. Dubai, T.R. Mahesh, and P.A. Ghosh. Design of new security algorithm: Using hybrid cryptography architecture. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 5, pages 99 –101, april 2011.

[11] M. Feldhofer and C. Rechberger. A case against currently used hash functions in rfid protocols. Workshop on RFID Security, 2006.

[12] M. Feldhofer and J. Wolkerstorfer. Strong crypto for rfid tags - a comparison of low-power hardware implementations. IEEE International Symposium on Circuits and Systems, 2007.

[13] Martin Feldhofer and Johannes Wolkerstorfer. Hardware implementation of symmetric algorithms for rfid security. In Paris Kitsos and Yan Zhang, editors, *RFID Security*, pages 373–415. Springer US, 2009. 10.1007/978-0-387-76481-$8_1$5.

[14] Yongqing Fu, Chun Zhang, and Jingchao Wang. A research on denial of service attack in passive rfid system. In *Anti-Counterfeiting Security and Identification in Communication (ASID), 2010 International Conference on*, pages 24 –28, july 2010.

[15] A.R. Ganesh, P.N. Manikandan, S.P. Sethu, R. Sundararajan, and K. Pargunarajan. An improved aes-ecc hybrid encryption scheme for secure communication in cooperative diversity based wireless sensor networks. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 1209 –1214, june 2011.

[16] N. Gay and W.-J. Fischer. Ultra-low-power rfid-based sensor mote. In *Sensors, 2010 IEEE*, pages 1293 –1298, nov. 2010.

[17] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for aes-like permutations. Cryptology ePrint Archive, Report 2009/531, 2009. `http://eprint.iacr.org/`.

[18] U. Gu andler and S. Ergu andn. A high speed ic random number generator based on phase noise in ring oscillators. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 425 –428, 30 2010-june 2 2010.

[19] U. Gu andler, S. Ergu andn, and G. Du andndar. A digital ic random number generator with logic gates only. In *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, pages 239 –242, dec. 2010.

[20] Gerhard Hancke. Eavesdropping attacks on high frequency rfid tokens. 4th Workshop on RFID Security (RFIDSec), 2008.

[21] Dijiang Huang and Harsh Kapoor. Towards lightweight secure communication protocols for passive rfid. IEEE Secon, 2009.

[22] EPCGlobal Inc. Tag class definitions 1.0, 2007.

[23] EPCGlobal Inc. *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz Version 1.2.0*, 2008.

[24] EPCGlobal Inc. *EPC Tag Data Standard Version 1.5*, 2010.

[25] Zhen-Ai Jin, Zi-Qiang Cheng, and Kee-Young Yoo. Spacing based authentication protocol for low-cost rfid. Second International Conference on Future Generation Communication and Networking, 2008.

[26] Mooseop Kim, Jaecheol Ryou, Yongje Choi, and Sungik Jun. Low-cost cryptographic circuits for authentication in radio frequency identification systems. IEEE, 2006.

[27] N.J. Lavigne, J.A. Marshall, and U. Artan. Towards underground mine drift mapping with rfid. In *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, pages 1 –6, may 2010.

[28] Byoung Nam Lee, Yong-Woon Kim, and Hyoung Jun Kim. Evolution of rfid applications and its implications: Standardization perspective. In *Management of Engineering and Technology, Portland International Center for*, pages 903 –910, aug. 2007.

[29] Zuohu Liu, Minghe Huang, and Shaojun Zhu. The design and implementation of a pseudo random number generation algorithm. In *Computational Intelligence and Natural Computing, 2009. CINC '09. International Conference on*, volume 2, pages 126 –129, june 2009.

[30] Andreas Loeffler, Uwe Wissendheit, Heinz Gerhaeuser, and Dina Kuznetsova. Inductively-charged and temporarily self-sufficient operating sensor using standard rfid-hf-technology. *RFID Systems and Technologies (RFID SysTech), 2009 5th European Workshop on*, pages 1 –7, june 2009.

[31] Andreas Loeffler, Uwe Wissendheit, Heinz Gerhaeuser, and Dina Kuznetsova. A multi-purpose rfid reader supporting indoor navigation systems. In *RFID-Technology and Applications (RFID-TA), 2010 IEEE International Conference on*, pages 43 –48, june 2010.

[32] B. McPhail, K. Boa, J. Ferenbok, K.L. Smith, and A. Clement. Identity, privacy and security challenges with ontario's enhanced driver's licence. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, pages 742 –747, sept. 2009.

[33] M. Meingast, J. King, and D.K. Mulligan. Embedded rfid and everyday things: A case study of the security and privacy risks of the u.s. e-passport. In *RFID, 2007. IEEE International Conference on*, pages 7 –14, march 2007.

[34] M.S. Miah and W. Gueaieb. An rfid-based robot navigation system with a customized rfid tag architecture. In *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, pages 25 –30, dec. 2007.

[35] Gustavus J. Simmons, editor. *Contemporary Cryptology : The Science of Information Integrity.* Wiley-IEEE press, New York, New York, 1992.

[36] D. Tagra, M. Rahman, and S. Sampalli. Technique for preventing dos attacks on rfid systems. In *Software, Telecommunications and Computer Networks (SoftCOM), 2010 International Conference on*, pages 6 –10, sept. 2010.

[37] V. Tavas, A.S. Demirkol, S. Ozoguz, S. Kilinc, A. Toker, and A. Zeki. An ic random number generator based on chaos. In *Applied Electronics (AE), 2010 International Conference on*, pages 1 –4, sept. 2010.

[38] Savi Technology. Active and passive rfid : Two distinct, but complementary, technologies for real-time supply chain visibility, 2002.

[39] M. Umaparvathi and D.K. Varughese. Evaluation of symmetric encryption algorithms for manets. In *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on*, pages 1 –3, dec. 2010.

[40] S. Willis and S. Helal. Rfid information grid for blind navigation and wayfinding. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on*, pages 34 – 37, oct. 2005.

[41] Chung Hsien Zah and S.F. Fezer. Embedded sensor scout flooring system by interface flor. In *Technologies for Practical Robot Applications, 2008. TePRA 2008. IEEE International Conference on*, pages 106 –110, nov. 2008.

[42] Yan Zhang, Laurence Yang, and Jiming Chen. *RFID and Sensor Networks.* CRC Press, Taylor and Francis Group, Boca Raton, Florida, 2010.

Permissions

Figures 2.3, 2.4, 2.5, 2.6, 2.7, and 2.8 were used with permission from their respective authors and the IEEE Xplore digital library