OpenTag - Privacy Control Methods in RFID

by

Daniel Z. Shen

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

Author .....
eering and Computer Science
September 8, 2006

Certified .....
Henry Holtzman
uage Workshop
nesis Supervisor

Certifie .....
John Maeda
Co-Director, Physical Language Workshop
Supervisor

Accepted by .....
C. Smith
Chairman, Department Committee on Graduate Theses

OpenTag - Privacy Control Methods in RFID

by

Daniel Z. Shen

Submitted to the
Department of Electrical Engineering and Computer Science
on September 8, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The work documented in this thesis is part of the OpenTag project, which has the goal of designing and developing a flexible and more powerful RFID system to meet the needs of the approaching ubiquitous tagging of everyday items. The focus of this thesis is on methods to improve privacy control in item level RFID tags. Several methods for the prevention of unwanted identification and tracking of tagged items are explored in the context of item level tags, which are greatly limited in size, cost, and computational power. A subset of the methods are implemented in RFID reader-side software and in an RFID Tag Emulator that mimics the behavior of an RFID tag while being suitable for time and cost efficient prototyping.

Thesis Supervisor: Henry Holtzman
Title: Co-Director, Physical Language Workshop

Thesis Supervisor: John Maeda
Title: Co-Director, Physical Language Workshop

Acknowledgments

I would like to thank Henry Holtzman for his guidance, mentorship, and vision throughout this project. I would like to thank John Maeda for his direction and inspiration and all the members of the Physical Language Workshop for their advice, instruction, and unparalleled willingness to help during my time at the Media Lab. My sincere gratitude goes as well to Ryan Bavetta, Anne Hunter, Rich Fletcher, Mike Pihulic and other friends, family, and colleagues, whose help has been invaluable. Finally I would like to thank the Simplicity Consortium and the Media Lab for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to RFID

Radio Frequency Identification (RFID) is a technology that facilitates automatic identification through the use of RFID tags and readers. RFID tags are comprised of an antenna and a means of storing a uniquely identifying number. Often a silicon chip is used to store the uniquely identifying number and the antenna is used to communicate that number wirelessly to an RFID reader. This relatively simple idea has great potential in both consumer and commercial settings to reduce costs and improve efficiency and capability. It is not surprising that RFID has gained popularity rapidly within the last few years.

(Note: for a more detailed introduction on RFID technology, see *Understanding RFID Technology* by Holtzman and Garfinkel [10].)

1.2 Uses of RFID

The use of RFID in commercial settings promises increased visibility in the supply chain, theft prevention, and more powerful counterfeit detection. Wooed by such potential benefits, Wal-mart is already requiring its top 100 suppliers to use RFID tags on their shipments to three main distribution centers in Texas. The Department of Defense, Target, and a Best Buy are following suit [11].

At the consumer level, RFID tags are popping up in applications such as Exxon Mobil's Speedpass, in credit cards, and in proximity cards. Other potential applications, such as washing machines that scan clothing with embedded tags and adjust water temperature accordingly, or tagged drug containers that can be used to detect fatal drug interactions, are anticipated.

1.3 Privacy Concerns of RFID

Although RFID has many potential benefits and applications it is also plagued by privacy concerns in connection with its use. These privacy concerns have fueled a negative public reaction to the widespread use of RFID technology. Privacy is certainly a concern for EPCglobal's Electronic Product Code (EPC), which is currently the most widely accepted RFID protocol for ubiquitous tagging in commercial settings. Potential breaches in privacy stem from two characteristics of EPC:

1) EPC tags communicate a uniquely identifying number

2) EPC tags are promiscuous (i.e. they will communicate with any reader that communicates with them)

The two main classes of RFID privacy attacks are the unwanted scanning of people's possessions and the unwanted tracking of tags and people. Theoretically, any reader within range can scan all the tagged items a person is carrying. Some privacy concerns involve retailers and advertisers profiling customers automatically and popping up ads based on the clothing they wear or signaling store clerks to swarm upon a customer carrying, say, a quadruple platinum rewards credit card. Other concerns include an unwanted listener or even a thief with a reader who is able to detect the presence of electronics, jewelry, banknotes, or prescription drugs on a person or in a house. Another worry is that a network of readers could detect the location and movement of a tagged item that a person carries and that this information could be used to track that person, determine patterns of behavior, or monitor, for example, attendance at certain rallies or protests.

Most of these concerns are well within the range of possibility for RFID technology.

It is important to note that tracking the movements of people or determining what types of clothing or possessions they have is possible without RFID and that those types of breaches of privacy are not new. What RFID provides is a simple, cheap, and automatic way of gathering this information, which makes these breaches of privacy much more likely.

1.4 Using EPC as a Basis

When implementing various methods of privacy control, we chose to use the EPC protocol from EPCglobal as a starting place. This choice was made because EPC is the most widely accepted RFID standard.

EPC was developed by the AutoID Center for the RFID equivalent of the Universal Product Code (UPC). The main purpose of EPC is for specific product identification as well as case and pallet identification in commercial settings. An EPC tag identifies the manufacturer, product, version, and serial number. The serial number is further associated with data related to the description, size, manufacturing, and storage information of the product through a standardized architecture that allows for access to external databases. This architecture is called the EPC Network [19].

1.5 Problem Statement

RFID technology has a great potential to benefit both companies and individual consumers but its openness and its ability to communicate a unique identification expose users to potential breaches of privacy.

Currently, there are several proposed approaches to improving privacy in RFID. In this project we will choose a few of these approaches to implement and test. In choosing between the approaches, we will weigh the benefits of the simplicity and cost effectiveness of a tag against the level of privacy control offered. Then we will build a hardware tag simulator that allows us the flexibility to test the different approaches and we will write the firmware that implements the proposed privacy

controls. Finally we will develop reader-side softare that works with the RFID tags to test and demonstrate the improved privacy controls.

Chapter 2

Overview of Privacy Control Methods

Several approaches for adding privacy control to RFID have been proposed. In choosing which forms of privacy control would be chosen for implementation, many methods were explored. Several methods have been previously proposed: Blocker Tags, Soft Blocking, Encryption of ID, Distance Detection, Pseudonyms, and Hash Chains. Some of the methods, like Private Databases, are new and explored in this thesis.

2.1 Existing and Proposed Methods for Privacy Control

2.1.1 Blocker Tags

The Blocker Tag was proposed by Juels, Rivest, and Szydlo at the *8th ACM Conference on Computer and Communications Security* as a means of improving privacy [1]. The central idea of the Blocker Tag is that it jams a reader that is trying to read a private tag (i.e. any tag that you don't want read) by pretending to be all possible tags. As an example, let's say that a customer is walking home with a shopping bag containing a soda, a watch, and a Carrot Top DVD and that each item has an RFID tag with a unique n-bit serial number. Any reader within range can theoretically read

the tags and determine the contents of the customer's bag.

If the customer chooses to keep the contents of the bag private, a Blocker Tag can be included amongst the other items, perhaps even embedded in the bag itself. According to EPCglobal's protocol for RFID communications [4], a reader determines what tags are present by querying the tags for their serial numbers one bit at a time. It first sends out a query for the first bit and all tags with a "0" for the first bit answer with a "0" and all tags with a "1" answer "1". This is repeated for all n bits of the of the n-bit serial number and the reader in the end knows which serial numbers are present and can choose to further communicate with the tags with which they are associated. A Blocker Tag jams the reader by answering with a "0" and a "1" for every bit in the serial number, thereby announcing to the reader that every possible tag ID is present. In the case of the EPCglobal's 96-bit serial numbers, that's 2^{96} or 79 octillion unique IDs.

This Blocker Tag can be varied slightly to offer greater flexibility. Let's say that we wanted readers to detect the watch and soda, but we want to keep the Carrot Top DVD private. In this case, a specific bit in the serial number may be used to signal that the tag is private. Any reader that is reading the set of serial numbers without the private bit set will not trigger the blocking. However, when the Blocker Tag detects that the private bit is being read, it can start to jam the reader.

Certain drawbacks exist with Blocker Tags. Blocker Tags can be used maliciously to jam readers. To guard against this kind of attack, readers can easily detect the presence of Blocker Tags by, for instance, checking for the presence for a certain number of random tags. The probability that the randomly chosen tags are present is extremely low and if they report their presence, it's likely that a Blocker Tag is being used. Another drawback is that the customer would have to choose to use the Blocker Tags. It would be better to have a system that activates privacy protection by default.

2.1.2 Soft Blocking

Soft Blocking, as proposed by Juels and Brainard in [14], allows for certain tags to be marked as private and unreadable, but unlike the Blocker Tags discussed in Section 2.1.1, the privacy control is implemented not in the tag but in a software or firmware module at the reader level. The software or firmware module is referred to by the authors as a TaPA (Tag Privacy Agent). The TaPA filters "tag data output by a reader prior to their transmission to other parts of the RFID system, primarily back-end applications" [14]. What's required is that all readers comply with a set of privacy control rules.

The system works by having different types of tags that invoke different behaviors from readers that comply with the privacy control rules. Tags can be marked as either public or private by flipping a privacy bit in their tag's ID. Additionally, different levels of privacy can be established with multiple privacy bits that allow varying levels of access to different readers. The authors also discuss the existence of "Blocker Tags". If a reader detects the presence of a "Blocker Tag", it will not read private tags. Otherwise, it will read both private and public tags. An alternative implementation discussed by the authors is to use "Unblocker Tags". Private tags can only be read in the presence of an "Unblocker Tag", which allows users to opt-in to reduced privacy instead of having it be the default option.

A question quickly arises about how the consumer can be sure that all readers comply. This is a genuine concern. Reader manufacturers can all choose to or be forced to comply with the regulations. As well, purchasers of the readers can choose to only buy readers that comply. However, this means that the customer must simply trust that the readers comply. This is not out of the question as a certain level of trust is needed and granted with most credit and debit card transactions. However, it is possible for readers to be constructed or modified to read private data. A partial solution is to detect non-compliant readers as the reading of private data can be detected by passively listening to the communications of the reader.

The major benefits of Soft Blocking are that simple passive tags can be used and

that the privacy control is available by default.

2.1.3 Encryption of Tag Data

Another method that is gaining popularity in commercial settings is to encrypt the data that is written to the RFID tag. This is especially useful in situations where the tag data or ID contains a person's account information or personally identifiable data. Having this unencrypted data read by a non-trusted reader may pose a serious privacy concern. If instead, the ID that is stored on the tag is encrypted, a reader that cannot decrypt the data will not pose a serious privacy risk.

What is appealing about this privacy control method is that the encryption can be done completely on the reader side. The reader can encrypt the data or tag ID using its choice of encryption schemes and write the encrypted data to a standard tag. This allows the tag to stay simple, cheap, and passive. For the reader, which may already perform a significant amount of computation, the encryption and decryption process does not necessarily increase the computational load to any great degree.

Encrypting tag data is a privacy control method that is already implemented in Philips' MiFare and DESFire and Texas Instruments' HFI tags, where the encryption algorithms and tags are proprietary. Skyetek is another RFID company that is introducing a line of readers that will allow encryption of data on standard RFID tags [18].

It should be mentioned that, if the tag data is statically encrypted by the reader, the tag is still susceptible to being tracked by a system of readers beacause it maintains the same ID. Even if the tag ID is dynamically encrypted by the tag in response to a challenge, as is the case with the TI's TIRIS (Texas Instruments Registration and Identification System), tracking is still possible. It is possible because the tag responds with the same answer to a given challenge and a reader can make use of this to identify the tag [13].

2.1.4 Distance Detection

Having a tag respond only to readers that are in close proximity offers another means of privacy control. The consumer can control which readers can communicate with a tag by controlling the communication distance and the physical location of the tag. Similarly it is less likely that random or unseen readers are within the communication range.

A number of methods exist for a tag to estimate the distance to a reader. Fishkin and Roy proposed a method of estimation using a correlation between the signal-to-noise ratio of a reader query and the physical distance to the reader [15]. They also lay out the idea of using Distance Detection for privacy control. Another possible method is to detect the signal strength of communication from the reader or limiting the transmission signal strength of the tag.

2.1.5 Pseudonyms

Juels proposes the use of Pseudonyms in [3] as a means of preventing unwanted location tracking. A tag with Pseudonyms would communicate not a single unique serial number but one of several serial numbers and a reader trying to track the movement of a single serial number would be thwarted. The tag's owner and trusted readers would know all of the pseudonyms of the tag and could associate them to a particular tag and its associated information in a database. However, not all readers would have access to this information.

Pseudonyms would not prevent tracking if all the pseudonyms of a tag were known. A reader may try to capture all the pseudonyms by repeatedly querying the tag. A tag may implement a long delay before sending a different serial number in order to lower the chances of a reader knowing all the pseudonyms. Another option is to allow the tags to be reprogrammed so that its entire set of pseudonyms changes. Again, this option does not allow for the use of simple passive tags.

2.1.6 Hash Chains

Ohkubo, Suzuki, and Kinoshita propose an approach that uses Hash Chains [17]. The tag and reader agree upon an algorithm that generates a sequence of serial numbers. Only those who know the algorithm know the sequence. Thus only the readers that know the algorithm can associate the serial numbers in the sequence with the identity of the tag.

Since the actual identification of the tag is not publicly known, the Hash Chain method prevents unwanted identification of tags. Also, since the serial number of the tag changes, it prevents tracking of the tag. This level of protection only works if the algorithm is unknown to unwanted readers. It is possible that the algorithm can be determined from reading multiple serial numbers in the sequence. However, the algorithm can be complex enough and sequence long enough that determining the algorithm from reading a subset of the sequence would be difficult.

For the Hash Chain method of privacy control the reader must be able to associate the current serial number given by the tag with the tag itself. This can be done if the reader knows where in the sequence the tag is. There are many ways to achieve this synchronization. One way is to only allow the tag to change its serial number to the next in the sequence after it has been read by one of its trusted readers. This way, the trusted readers can always be synchronized with the tag. If there exist trusted readers that are not part of the same system, these readers may have to talk to each other to ensure synchronization. This method would ensure that, for any number of queries from a non-trusted reader, the tag would give the same ID. One problem that having the same ID communicated to non-trusted readers is that it allows for tracking of the tag. However, tracking can be made more difficult if the tag has Pseudonyms (Section 2.1.5) for each point in the sequence. Another option, which works well with very controlled and regular patterns of reader/tag communication, is to change the serial number infrequently enough so that the reader would always have communicated with the tag before it changes its serial number. This forgoes the added complexity of having the reader initiate the changing of the tag's serial number. Yet another option

is to designate a few bits to communicate what position in the sequence the tag is in. This would allow a reader that knows the algorithm to immediately synchronize with the tag even if the tag had been read and progressed down the serial number sequence between readings. Having the position in the sequence communicated by the tag provides more information that can be used to crack the algorithm. However, the algorithm may still provide enough protection and the position information itself may be encoded with another algorithm.

I have discussed the possibility of using a secret algorithm to ensure privacy. However, there are variations on the Hash Chain that do not require the algorithm itself to be secret. The central principle is that there is some secret kept between the tag and trusted readers. This secret can be, for instance, a secret number that the algorithm uses to generate the sequence. In fact, this approach may be preferable because if only the algorithm is secret, cracking the algorithm will grant access to all tags using the algorithm. However, if each tag has a different secret number, cracking a secret number will only grant access to a single tag. In addition, making the algorithm public will allow people to test their applications and tags with the algorithm.

2.1.7 Private Databases

Many consumer and commercial applications of RFID require the association of the serial number communicated by the tag with information about the tag contained in an external database. For example, Wal-Mart wants to know that the string of bits its RFID reader is picking up refers to a crate of Dr. Peppers that have a remaining shelf life of 2 months. A consumer's smart medicine cabinet wants to know that the serial number it gets from one of the tags inside it contains MerckMed-A and it shouldn't be mixed with PfizerPill-B so that it can give a warning when MerckMed-A is being taken out of the cabinet to be used. These associations are made by a database that contains both the information about the tag and the tag's serial number.

In settings where privacy is not a major concern, the database can be public and anyone who can read the tag and get information about the tag. In many commercial

settings, a public database would be appropriate and beneficial. A public database allows for the possibility of a standardized and open database that is populated and updated by many manufacturers so each company does not have to create and manage its own database. Further improving the ease by which RFID tags provide information about the tagged item, EPC dictates a well known and predictable numbering scheme by which information, such as the item's brand and model, is encoded by specific sequences in the tag's ID.

However, for many consumers, more privacy is desired and increased privacy can be offered with a private database. Only those with access to private database would have access to information about the tag.

The key is to allow a tag to assume a different form in different settings. Where a public database is useful, the tag can communicate its serial number that's associated with the public database. When more privacy is desired, the tag can communicate a serial number associated only in a private database. This can easily be accomplished by rewriting the tag's ID. A customer may take items home that have tags with public serial numbers. Once home a device may rewrite the tag's ID and associate the tag's new private ID with its public ID or with information about the tag. The rewriting device may be too expensive for most homes, and in such a case, the rewriter may be available at the point of purchase. Upon buying the tagged items, the consumer may rewrite the tags at the store and the associations between the tag's public ID with its new private ID can be stored on a memory card to be taken home or communicated to the consumer's home computer. Afterwards the store can destroy its association information or store it securely. Once the association is made between the tag's new ID and the tag's information, the consumer can control access to the tag's information by controlling who has access to the private database.

Since the tag still has an ID, it is possible to track the tag even though the information about the tag is unknown. This threat can be reduced by using Pseudonyms (Section 2.1.5) or by rewriting the tags again from time to time. One of the benefits of the Private Database approach is that the tags need only to be rewritable and can be kept fairly simple.

2.2 Selected Privacy Control Methods

Of the approaches to privacy control discussed, the methods I chose to implement are Hash Chains, Pseudonyms, and Private Databases.

Hash Chains were chosen because they provide a high level of security whereby potentially only readers that know the algorithm for deriving the sequence of tag ID numbers can know the true identity of the tag. Private Databases were chosen because they can provide security without sacrificing the simplicity and cost efficiency of the tags. The tags need only be rewritable for this form of privacy control, which can ensure that only RFID systems that have access to the Private Databases can associate the tag's new ID with the tag's information. Pseudonyms were chosen as a simple method of preventing the tracking of RFID tags that, though it cannot it be implemented with standard tags, requires only minimal additional computation. In addition, Pseudonyms can be easily added t o a tag that uses Hash Chains as a variant that offers a relatively high degree of privacy control while also preventing tag tracking.

Soft Blockers were not chosen because they require a level of trust of the readers that may not be realistic and takes away from the consumer the ability to control their level of privacy.

Blocking Tags and Distance Detection are also good methods, but Hash Chains, Pseudonyms, and Private Databases were chosen because they offer a different and wider range of features (i.e. high security and simple tag implementation) to explore. That said, a simple form of Distance Detection may be used to control data leakage as tags pass from public to private scenarios. For example a tagged item going from the store to home before privacy control is activated can be set to only respond to close readers so that unauthorized readers will find it difficult to query the tag.

Chapter 3

Hardware Design

3.1 Overview

A number of privacy control methods can be accomplished with currently available and standard EPC tags. For Private Databases, for instance, the privacy control can be accomplished in software alone. However, for other methods such as Hash Chains and Pseudonyms, the tag itself must be more powerful than the standard EPC tags. To implement these methods, an RFID tag was made that allowed for more computational power and flexibility than standard RFIDs. To design and fabricate an RFID tag similar to modern EPC tags, which have small chips that are on the order of 1 mm on either side, would be too time consuming and costly. Therefore, the RFID created is an RFID tag emulator that behaves like an RFID tag, but is larger and more practical to prototype with. The tag emulator allows for the different privacy control methods to be implemented and tested quickly with the understanding that they can be included in a semiconductor IC if adopted for RFIDs on a larger scale.

The RFID tag emulator is a device that responds to queries from an RFID reader in accord with one or more RFID communication protocols. This means that when a reader sends a signal to the tag emulator to request its ID, the tag responds with the correct digital modulation, bit rate, etc. However, the tag emulator will have a more powerful microprocessor and allows us to easily update or change its firmware. The central components of the tag emulator are a microprocessor to implement the

RFID communication protocol and the privacy control methods, wireless transmitter and receiver circuitry, and an antenna. In designing the tag emulator, I was able to build upon and existing EPC Generation 1 tag emulator designed by Rich Redemske for his Master of Engineering thesis [Redemske]. Redemske's emulator implements the communication protocol for EPC Gen1 RFID tags. Thus, in this thesis we were able to concentrate on implementing the privacy control methods in firmware on a separate microcontroller and circuit board and use Redemske's emulator to handle the lower level communications with the RFID reader.

Note that, although we are using EPC as a starting place, we will not limit our tag emulator to the functionally of an EPC tag. An EPC tag only communicates its unique identifier and does not allow for any privacy control other than a kill bit, which will disable the tag entirely. This kill bit is intended to be used when the tag is given to the consumer, but, when it renders the tag safe from privacy attacks, it also renders the tag useless to the consumer for RFID applications. Instead of a kill function, the tag simulator will use the hash chains and private database approaches to privacy control. Also, under the EPC standard, data is only written once to the tag. Any changes to the information related to the tag are written to the EPC Network. The tag simulator will allow for the tag ID to be changed to comply with the requirements of the different forms of privacy control.

3.2 Tagsense RFID Tagmodem

In his Master of Engineering Thesis entitled "An Electromagnetic Measurement Tool for UHF RFID Diagnostics" [21], Rich Redemske describes the design and construction of a passive RFID UHF (i.e. Ultra High Frequency) emulation tag. His tag implements that Auto-ID Center/EPCglobal Generation 1 RFID passive UHF tag protocol and consisted mainly of an antenna, transmit and receive circuitry, a microcontroller, and circuitry that allowed it to measure and display the power level of received signals from a reader.

The EPC Gen 1 tag emulator used in this thesis was a product from Tagsense

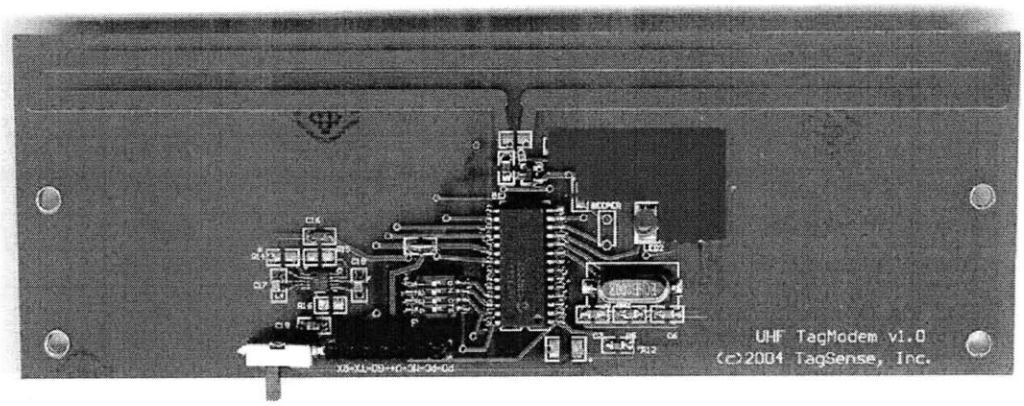

Figure 3-1: Tagsense UHF Tagmodem. Taken from [20]

called the UHF Tagmodem that was based on Redemske's Tag Emulator and is shown in Figure 3-1. According to the user manual [20], the "UHF Tagmodem enables communication between any RS-232 device and a UHF EPC RFID Reader. The current version of the Tagmodem uses EPC protocol Class 1 Generation 1, informally known as the 'Alien Protocol' used by Alien Technology RFID readers". In essence, the Tagmodem performs the simple but important task of taking a 64-bit ID through serial communication, storing it, and sending out that ID when queried by a reader.

3.3 Privacy Control Mainboard

The firmware that dictates the operation of the tag emulator and directs the privacy control behavior is stored on the Tag Emulator Mainboard. To reduce size and power consumption, the mainboard was kept relatively simple and is comprised of a microcontroller, a power supply, and I/O that allows for communication with the Tagsense Tagmodem, the microcontroller programmer, and a serial port for debugging on a computer.

Table 3.1: Microprocessors Considered for the RFID Tag Emulator

Microprocessor	Speed	Memory	USART	Package	I/O pins
Atmel ATmega48	$\leq$ 20 MIPS	4 kB	Yes	TQFP 32	$\leq$ 23
Atmel ATmega88	$\leq$ 16 MIPS	8 kB	Yes	TQFP 32	$\leq$ 23
Atmel ATmega168	$\leq$ 20 MIPS	16 kB	Yes	TQFP 32	$\leq$ 23
Atmel ATmega644	$\leq$ 20 MIPS	64 kB	Yes	TQFP 44	$\leq$ 32
Atmel ATmega128	$\leq$ 16 MIPS	128 kB	Yes	TQFP 64	$\leq$ 53
Atmel ATtiny44	$\leq$ 20 MIPS	4 kB	Yes but harder	SOIC 14	$\leq$ 12
Atmel ATtiny84	$\leq$ 20 MIPS	8 kB	Yes but harder	SOIC 14	$\leq$ 12

3.3.1 Microprocessor

In choosing a microcontroller, the main considerations were that it be fast and powerful enough to implement the chosen privacy control methods, that it be relatively small so that the mainboard PCB could be small and easy to design, and that it be quick and cheap to develop with.

In terms of the ease and cost of development, the Atmel AVR family of microcontrollers was attractive because the microcontroller itself and its programmer are both inexpensive and there are good free compilers and development environments. In addition to how inexpensive and readily available the AVR development tools are, and partly because of it, the AVR have become very popular and there are large libraries of code, lots of support, and tutorials that facilitates coding and debugging. In addition, there are many AVRs to choose from so it should be possible to find something small and powerful enough.

After deciding on Atmel AVRs, I looked at a number of different microcontrollers that had the required features. Table 3.1 shows several features of the microprocessors considered for the tag emulator. In keeping with the consideration that the privacy control methods should not be computationally intensive in order to keep the complexity and power consumption of the RFID tag low, all of speed and computation requirements were not very limiting. Thus, any of the microcontrollers listed in Table 3.1 should be adequate. The benefits of the ATtiny microcontrollers are that they are

small and cheap. However, those benefits were weighed against limited number of I/O pins on the ATtiny microcontrollers. Extra I/O pins in excess of the number required for the final version Tag Emulator are useful for debugging during the development stage. All of the ATmega microcontrollers had sufficient I/O pins and flash memory. Physically, the smallest ATmega microcontrollers are the ATmega48, ATmega88, and ATmega168, which all came in a TQFP 32 surface mount package. The ATmega168 was chosen because it had the most flash memory of the smallest ATmega microcontrollers and would allow for more complex behavior to be programmed into the Tag Emulator without sacrificing size.

3.3.2 Power

The power requirements for the Privacy Control Mainboard are low and a simple voltage regulator works as the power supply. Initially the mainboard was developed on a breadboard and a National Semiconductor LM340T5 7805 5V regulator was used. The breadboard version of the mainboard could be used either with a battery or an AC adapter if it needed to be on for a long time during development.

The PCB version of the Privacy Control Mainboard runs on 3V and uses a CR2032 coin battery. No voltage regulator is used but a 15 uF tantalum capacitor is used.

3.3.3 I/O

The Privacy Control Mainboard communicates with the Tagsense Tagmodem, the microcontroller programmer, and has a serial port for communicating with a computer when debugging.

The Tagmodem sends its ID to a reader when queried and it receives a new ID through a 4-pin serial interface. This 4-pin serial interface, which consists of separate transmit, receive, power, and ground lines, operates at 3V. Therefore, if the microcontroller is operating at 3V, the Tagmodem can be connected directly to the ATmega168's USART pins without an additional driver/receiver IC (eg. MAX232 or MAX233).

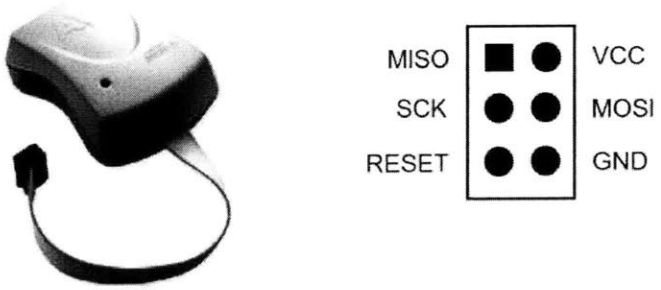

Figure 3-2: Atmel AVR ISP and 6-pin header. Taken from [5] and [6]

For debugging purposes, it is useful to let the Mainboard communicate directly with a computer through a serial port. The ATmega168 has a serial programmable USART (Universal Synchronous-Asynchronous Receiver/Transmitter), which when combined with a driver/receiver IC allows for serial communication with a computer serial port. In the breadboard prototype, a MAX233 was used.

The Mainboard allows for reprogramming of the ATmega168 without taking it out of its final circuit using an In-System-Programmer. There are several types of In-System-Programmers available. The one used for the mainboard was the Atmel AVR ISP, which has a 6-pin connector, both shown in Figure 3-2. The Mainboard has a 6-pin receptacle for the AVR ISP that connects to the programming pins on the ATmega168.

3.3.4 Physical Characteristics and PCB Design

We wanted to make a PCB (Printed Circuit Board) version of the Tag Emulator mainboard because it would be more easily reproduced, rugged, compact, and also less susceptible to noise. The main design features of the PCB were:

- can be soldered by hand quickly and relatively easy
- small size . less than 2in by 2in

- powered by standard coin battery . i.e. all components can operate below 3V
- can connect directly to Tagsense Tagmodem
- can connect directly to Atmel AVRISP for programming

To keep the size down, small surface mount components were used were possible, thin copper traces were used for signal lines, and care was taken to lay out the PCB in a relatively compact fashion. To make the soldering process easier, the case size of surface mount components were kept at or above the 0805 (0.08in by 0.05in) size and a minimum of 0.1 inch spacing was maintained between adjacent components. To allow the mainboard to run off a 3V coin battery, making for a smaller overall device, a microcontroller that can handle less than or equal to 3V was required. The Atmel ATmega168 can operate with a 2.7V-5.5V supply voltage, which should suffice. However, the 2.7V minimum was close enough to the 3V battery voltage that a low battery or a full load may cause the supply voltage to drop below the minimum requirement for reliable operation. So instead of using the ATmega168, I chose to use the ATmega168V, which operates with a supply voltage between 1.8V and 5.5V [2]. The ATmega168V only runs up to 10MHz, less than the 20MHz of the ATmega168. However, the Tag Emulator mainboard only uses the 8MHz internal oscillator of the microcontroller and so the decrease in maximum speed was not an issue. 0.1inch pitch pin headers were added to the PCB to allow for it to connect directly with the AVRISP and the Tagsense Tagmodem. LEDs and a 2-position switch were added to allow for optional debugging, I/O, and the ability to change the mainboard's behavior without needing to reprogram it.

PCB layout was done in EAGLE (Easily Applicable Graphical Layout Editor) version 4.11. The schematic for the Tag Emulator mainboard is shown in Figure A-1 in Appendix A. The board layout is shown in Figure 3-4. In laying out the PCB, basic precautions were taken to reduce noise. A ground plane was placed behind the microcontroller. Signal lines on different layers crossed at right angles and signal lines did not run parallel to each other in close proximity. The PCB was printed by Advanced Circuits. The finished mainboard PCB is a 1.5in by 1.5in 2-layer board.

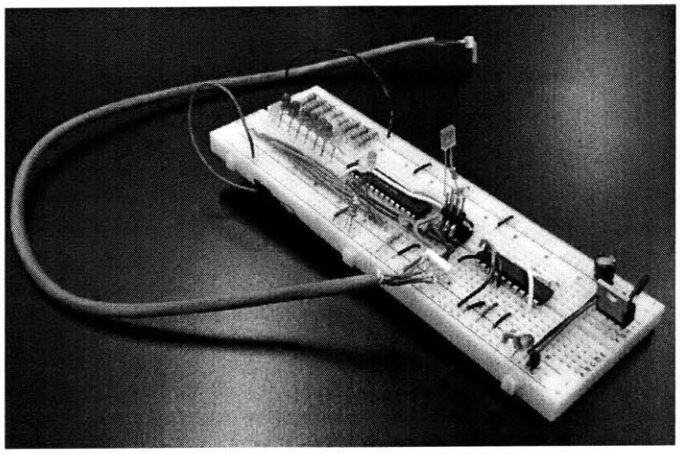

Figure 3-3: Tag Emalator Protoboard

The first Tag Emulator mainboard was prototype on a protoboard and is shown for comparison in Figure 3-3.

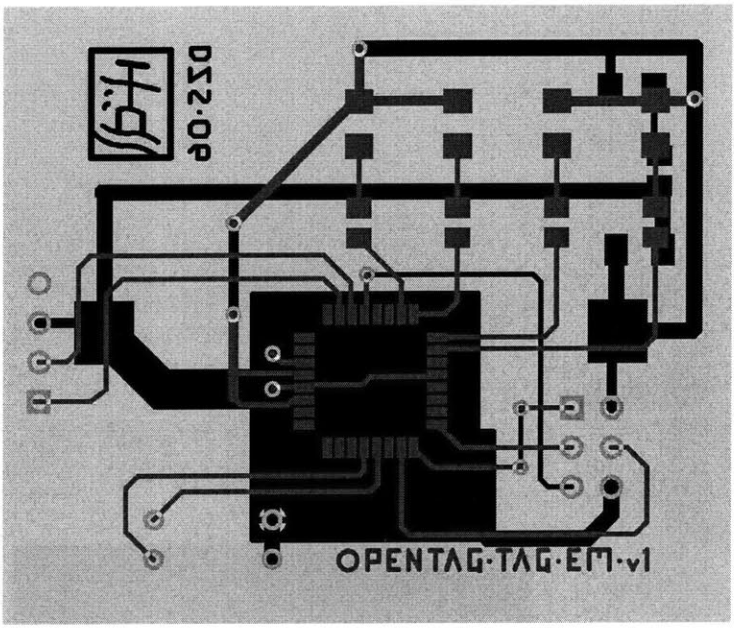

Figure 3-4: Tag Emalator PCB, Top and Bottom Layers. Drawn in Eagle 4.11

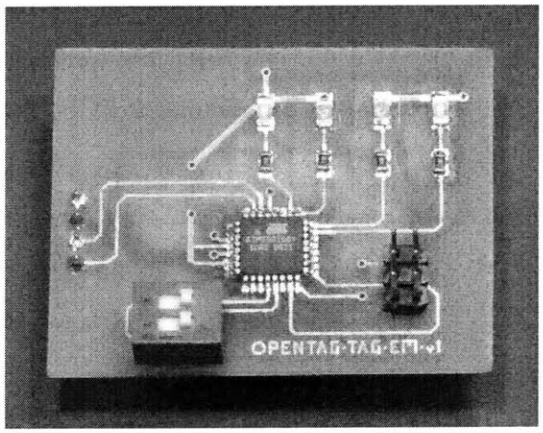

Figure 3-5: Tag Emulator PCB - Front View

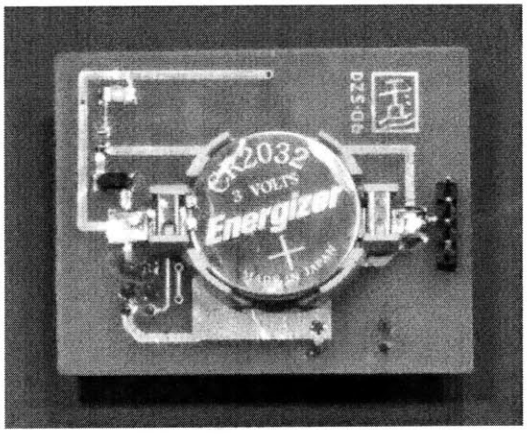

Figure 3-6: Tag Emulator PCB - Back View

Chapter 4

Implementation of Selected Methods for Privacy Control

4.1 Private Databases

Private Databases is a simple privacy control method that can be implemented using current Gen2 EPC tags and readers. The idea is to change the tag ID from one that is listed in a public database where any reader can gain access to information about the tagged item or from one where information about the tagged item is stored in the ID itself, such as a manufacturer code, production date, or a customer account number. To improve privacy, the ID is changed to one that is meaningless to non-trusted readers and software and the association between the ID and tag information can be made on a private database. The ability to write tag IDs is a feature of the Gen2 EPC protocol and so Private Databases can be implemented on the reader side alone with standard hardware, which is one of its main benefits.

The Private Databases method was implemented in reader-side software written in Python to work with the Thingmagic Mercury4 RFID reader. The source code is available in privateDBReader.py in Appendix B.

The main elements of the reader software were functions that set up ID queries using commands to the Mercury4 reader and that look for the IDs that were read amongst the set of stored IDs for which tag information was available. A visual

interface was included so that any ID that was recognized showed up with an image of the tagged item and any ID that was not recognized would show up with a question mark.

Tags could be written to using the Thingmagic Tesla MercuryOS 2.3 web-based user interface and then the new IDs along with tag information were entered into the reader software. In the supplied software only a few tags were entered for demonstration purposes and the data structure used for storing information was a dictionary. However, many IDs can easily be entered and other data storage methods such as a SQL database could be used. In practice the tags could be written with their new IDs upon purchase by a consumer or at home and the tag writing process can be automated by commands to the reader.

4.2 Pseudonyms

The method of using pseudonyms was one of the privacy control methods implemented on the Tag Emulator. Pseudonyms is a method that requires an upgrade in the capabilities of passive RFID tags and thus could not be done through changes in the reader software alone. However, it is a very simple method that requires little computation and little memory.

The implementation for Pseudonyms for the Tag Emulator was done in C for the Atmel ATmega168V microcontroller. The source code is shown in pseudonyms.c in Appendix B.

In this implementation of Pseudonyms, the Tag Emulator works by sequentially moving through a list of ten different IDs. Every time a reader reads the Tag Emulator's ID, it changes its ID, returning to the beginning of the list when it reaches the end. This accomplishes the task of presenting not one but many IDs to readers so that readers outside of the trusted set of readers would not be able to track a single ID as it progresses through a system of readers distributed throughout a given area.

The implementation of the Pseudonyms privacy control method has two modes of operation. In one, the Tag Emulator progresses through a series of ten IDs that are in

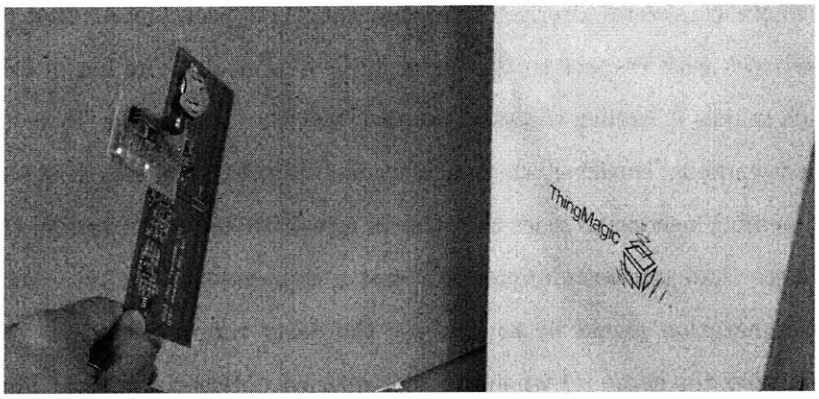

Figure 4-1: Tag Emulator in Action

sequential order as shown in Figure 4-3, where the display shown is the Query page of the Thingmagic Mercury4 RFID reader's Tesla MercuryOS 2.3. This is mainly for demonstration purposes, and it is possible and not difficult for readers outside of the trusted set of readers to detect all the IDs and associate them all to a single tag because they are related. In another mode of behavior that is more realistic, the Tag Emulator progresses through a list of ten random hard-coded IDs. This behavior is shown in Figure 4-4. In this mode, it would be much harder for a non-trusted reader to associate with a single tag. The different modes of operation are selected for by a switch on the Tag Emulator. Compare the behavior of both modes of operation with the behavior of a standard EPC RFID, shown in Figure 4-2, which only displays a single tag ID.

Another security measure added was a delay between possible successive reads of the Tag Emulator. This prevents a reader from rapidly querying a tag and getting all of its pseudonyms.

There are several alternate implementations of pseudonyms that have benefits and drawbacks with respect to the given implementation. The list of IDs can be longer, which makes it harder to associate all IDs with a single tag. In fact, the list can be almost endless. However, there is a tradeoff between the length of the list of IDs and the memory needed to store the IDs. In a realistic implementation, there would likely be more than ten pseudonyms, but not a tremendous amount. Another alternative implementation would be to increase the delay time between ID changes. The tag could offer the same ID whenever it is queried between changes. This would make it harder for readers to query a tag for all its pseudonyms within a small or reasonable amount of time. Again there is a tradeoff: the longer the delay, the more likely that a system of readers will be able to see the same ID moving from place to place and track a tag's, and consequently an individual's, movements. The optimal choice of delay time and number of pseudonyms can depend greatly on the application.

Another feature of Pseudonyms is that it is easily combined with other privacy control methods. Pseudonyms can be an extension of Private Databases in that the pseudonyms can be associated with information about the tag only within a private

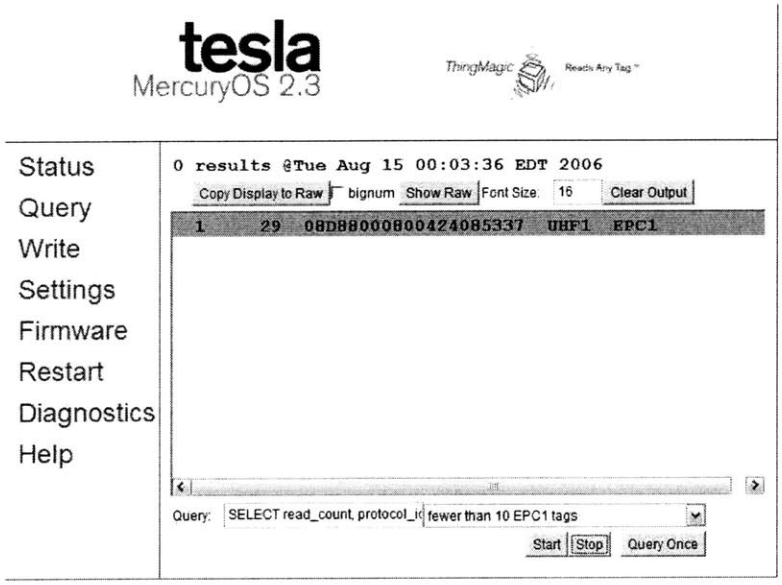

Figure 4-2: Regular EPC Tag Reading: Single ID

database and readers outside the trusted network would not have access to that information. Similarly, Pseudonyms can be used in conjunction with Hash Chains.

The reader-side software for the Pseudonyms method was written in Python to work the Thingmagic Mercury4 reader and the source code is available in pseudonymsReader.py in Appendix B. The reader-side software for Pseudonyms is the same as that for Private Databases, except that all of the pseudonym IDs for a given tag are stored and they are all associated with information about the tag.

4.3 Hash Chains

Like Pseudonyms, the Hash Chains privacy control method requires a tag with capabilities beyond those of the current standard EPC tags, and, like pseudonyms, it

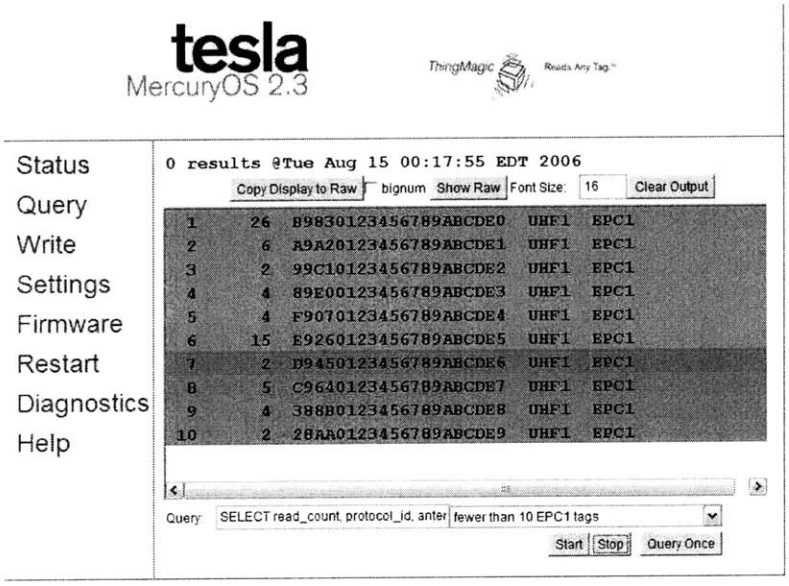

Figure 4-3: Pseudonyms in Action: 10 Sequential Pseudonyms

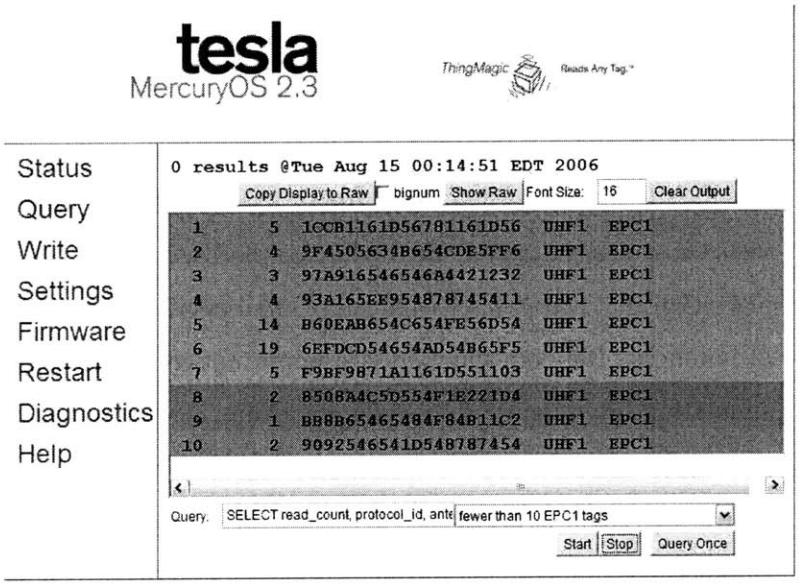

Figure 4-4: Pseudonyms in Action: 10 Random Hardcoded Pseudonyms

was implemented on the Tag Emulator. In the Hash Chains method, the basic idea is that an RFID tag progresses through a sequence of IDs that is known by trusted readers. A trusted reader is then able to associate any ID in the sequence with the physical tag. What separates Hash Chains from Pseudonyms is that each ID in the sequence is not stored in the tag but rather it is generated by the tag using an algorithm known both to the tag and the reader. In theory this means that the sequence of IDs can be arbitrarily long and even infinitely long without having the memory requirements of the tag grow in proportion. However, as will be explained in the analysis of the algorithms that generate the sequence, generating longer sequences often requires more memory. Hash Chains, in general, also require more computational power than pseudonyms, which can prove to be prohibitive for a small, low-power, and inexpensive tag.

In implementing the Hash Chains method, first a class of algorithms and then a specific algorithm need to be chosen. Pseudo random number generators (PRNG) were chosen as the type of algorithm for the Hash Chain. A PRNG is a deterministic algorithm that generates a sequence of numbers that approximates a random sequence of numbers. Random numbers are important for applications such as Monte Carlo simulation, cryptography, genetic algorithm, and circuit testing [22]. However, in most cases, deterministic machines, such as a computer, running deterministic algorithms are used to generate these random numbers. Since deterministic machines running deterministic algorithms can only generate deterministic sequences of numbers, the numbers cannot be truly random and so the generators are referred to as PRNGs. A given PRNG, given the same starting value (or seed), and supplied with the same parameters, will always generate the same sequence of numbers.

PRNGs are an attractive candidate for RFID Hash Chains for several reasons. The sequence of IDs that an RFID tag implementing the Hash Chain method progresses through must not only be known to the tag, but also to trusted readers. Therefore, the fact that PRNGs are deterministic and not truly random is a critical feature required by the Hash Chain method. It is important that both a reader and a tag, running the same Hash Chain algorithm and starting with the same seed, must generate the

Table 4.1: Features of Different PRNGs

PRNG	Primary Applications	Features
Linear congruential generator	simple non-cryptographic applications	fast, simple, small
Lagged Fibonacci generator	simple non-cryptographic applications	fast, simple, small
Linear feedback shift register	non-cryptographic applications and low quality cryptography	easy to implement in hardware, can be very fast
Blum-Blum-Shub generator	cryptography	slow, secure
Fortuna	cryptography	secure
Mersenne twister	non-cryptographic applications	fast, long period, good randomness
Yarrow	cryptography	secure, royalty-free
ISAAC	cryptography	secure

same sequence of numbers and a PRNG does just that.

Another important feature of a Hash Chain algorithm is that a reader outside of the set of trusted readers would not be able to tell that a tag is progressing through a sequence of IDs and would not be able to determine which IDs belong to that sequence. Using a sequence of IDs that appears random, that is with no discernable pattern, is one way to accomplish this. Much effort has been spent on creating PRNGs that generate a sequence of numbers that appears random. There exist PRNGs that approximate randomness to a very high degree. As well, there are PRNGs that approximate a random sequence of numbers to varying degrees to suite different requirements, many which are highly optimized to run efficiently and many of which have good documentation on their performance, strengths, and weaknesses.

After choosing PRNGs as the class of algorithms to use for the Hash Chains implementation, the choice of a specific PRNG was made. Several PRNGs were looked into and they, along with a subset of their features, are listed in Table 4.1.

The two main requirements of a PRNG are that it generates numbers efficiently in terms of time and memory and that it be secure [23]. For an RFID tag meant for ubiquitous tagging in particular, the efficiency requirement is important because of

the tag's limitations in size, memory, and cost. Speed may also be important factor in RFID protocols and applications where rapid reading of tags is required. Therefore, most of the cryptographically secure PRNGs are prohibitively inefficient. Of the remaining choices, the two that seemed most appropriate were the Linear Congruential Generator (LCG) and the Mersenne Twister. The LCG can be extremely fast and only requires the storage of up to five variables the size of the number generated. The Mersenne Twister is also very fast but is a much better approximation of a random number than the LCG.

The LCG algorithm, Equation 4.1, is a very simple equation whose behavior depends greatly on the parameters A, B, and M.

$$V_{j+1} = (A \cdot V_j + B) mod M \tag{4.1}$$

LCGs have a period of at most M before they repeat themselves and sometimes much less. In order maximize the period of a LCG, the following requirements are made on the parameters [12]:

1. B and M are relatively prime
2. $A - 1$ is divisible by all prime factors of M.
3. $A - 1$ is a multiple of 4 if M is a multiple of 4
4. $M > max(A, B, V_0)$
5. $A > 0$, $B > 0$

By comparison the Mersenne Twister has a period of $2^{19937} - 1$, much greater than any M used for an LCG. However, it requires about 2kB of memory to run. If that memory is available, it runs much faster than any LCG that comes even close to attaining the same period and it better approximates a random sequence than LCGs.

Though both LCGs and the Mersenne Twister would likely be appropriate choices for the Hash Algorithm, LCGs were chosen because of how efficient they are for small number generation. LCGs are the best option for many embedded systems and for a

very low-power, low-memory RFID chip, it may be the only viable option before the introduction of much more efficient and powerful RFID chips.

The Hash Chains method using LCGs was implemented in C for the Atmel Atmega168V microcontroller and the source code is provided in the file lcg_pcb.c in Appendix B.

Since the Atmega168V is an 8-bit processor, the LCG was used to generate 8-bit sequences. To make up the 64-bit EPC ID required by the tag, eight sequences were generated independently and concatenated. To generate an 8-bit number, M from Equation 4.1 needs to be less than or equal to 256. Since the numbers generated by the LCG cannot exceed M, M was set to be 256 to ensure that the full range of possible IDs could be generated, better approximating a random sequence of IDs. Moreover, LCGs where M is a power of 2 are the fastest form of LCGs because the calculation of $mod M$ is fast when M is a power of 2.

The period of a single 8-bit sequence generated by the LCG has a maximum period of 256 and it would not take long for a reader to read a tag 256 times. However, since each individual 8-bit segment of the tag ID has a 256 and there are eight independently generated 8-bit segments, the maximum period of the entire ID is $8^8 = 16777216$. EPC Gen2 theoretically allows for read speeds up to 1600 tags per second (640kbps) [8], about 10 times the speed of EPC Gen1. Thus, at its greatest theoretical throughput, a Gen2 reader would be able to read 16777216 IDs in $16777216 \div 1600 = 10485.76$ seconds (about 3 hours) and it would take a Gen1 reader about 10 times as long. It is possible for a reader to read through all the tag IDs, but it is unlikely. So, as for period, the LCG generated sequence of IDs is probably sufficient. If the tag were to implement a delay between changes in its ID, the time required to read the entire sequence, or enough of it to analyze and predict its progression, would be further enlarged. It is also possible that the tag will be able to detect when it moves in and out of the read range of a given reader. The tag can then change its ID only when it moves out of the read range of a reader [13]. Thus a single reader will only see one ID within the sequence of IDs of a given tag, significantly increasing the difficulty of analyzing the sequence of IDs.

In this implementation of the Hash Chains method, the sequence of IDs is private even when the IDs, algorithm, and M from Equation 4.1 are publicly known. What needs to be kept private, though, are the parameters A and B. These parameters constitute the secret information between the tag and trusted readers. They can be hard-coded into a tag and known to a reader, or they can be writeable only by trusted readers.

The reader-side software for the Hash Chains method was written in Python to work the Thingmagic Mercury4 reader and the source code is available in lcgReader.py in Appendix B.

In a similar fashion to the reader-side software for Private Databases and Pseudonyms, the reader in the Hash Chains method constantly reads tags in its vicinity and when IDs that are listed amongst its set of stored IDs are read, it displays an image of the tagged item. Otherwise it displays a question mark. The difference is that, to synch up with a tag that has implemented the Hash Chains method, the reader software must be able to recognize any one of the IDs within a sequence of IDs generated by a single tag as IDs that are associated with that tag. To accomplish this, the reader software knows the algorithm (LCG in this case), the parameters (A, B, M), and the initial or seed ID. The software uses the LCG to generate a set of IDs, in this case 100, and it tries to match the IDs read with any of the 100 IDs in the generated list. If the ID is matched, the reader then generates a new set of 100 IDs starting from the latest ID read. The number of IDs generated at a time can easily be raised if the tag is expected to progress through its sequence of IDs quickly since the reader software list not greatly limited in storage. Alternative behaviors for the reader-side software are possible and are explored in Section 5.3.

Chapter 5

Results and Discussion

5.1 General Comments on the Implemented Privacy Control Methods

Three privacy control methods were prototyped: Private Databases, Pseudonyms, and Hash Chains. The design of the methods was greatly influenced and constrained by the need to limit cost, size, and power consumption in a tag meant for ubiquitous item-level tagging. For the most part, though, the implementation of the methods was simple and straightforward.

The implementation of Private Databases functioned by having a local data structure that stored associations between an ID and information about the tagged item. It made use of the rewriting feature of EPC tags. Once the data structure was populated with the associations of the tags it needed to recognize, the system operated like any other RFID tag-reader system. The improvement in privacy comes from the fact that the storage of associations was local and not publicly available. This privacy control method, though simple and ineffective against tracking privacy concerns, represents the easiest, most inexpensive, and most likely privacy control method to be implemented in a practical system. Since this method relies on the rewriting of tags, it is crucial that there be controls on who is allowed to write the tags. An unexpected or unwanted write to the tag can easily render the tag ID unrecognizable

to the local system. It may be reasonable to assume the tag will not likely be written to by unwanted devices because the writing of tags requires the tag to be close to the writing device, significantly closer than for a read. This proximity requirement is a minimal security measure. However, more secure measures may be required. This is especially true because, if a tag at a store or other commercial setting is writable, it is possible to rewrite those tags and disrupt RFID system and applications. One possible security measure is to have a proximity detector on the tag that requires that writes must occur at very close ranges or only when the tag and writer are touching. This reduces the probability of undesired writing. It is also possible to control access to writing through a password or challenge response authentication.

In the Pseudonyms implementation, the tag cycles through a limited set of IDs that is associated with information about the tag in a local data structure. It is an expansion upon the method of Private Databases. The two main differences are that it takes measures to prevent tag tracking and that it requires a tag that is more powerful than current EPC tags. The more powerful tag was prototyped with the Tag Emulator, but would require a minimal amount of additional computation in comparison to a standard EPC tag. The more secure variations of Pseudonyms require the tag to change IDs in between reads. This can mean that the tag cannot be a simple passive tag but rather it must be a semi-active or active tag. However, the additional energy required is minimal and can be stored in a capacitor, though it would still increase the size and cost of the tag chip. If the ID only changes when read by a trusted reader or only changes as it leaves or enters the read range of a reader, it is possible to design a passive tag that implements Pseudonyms that gets its power from the reader.

In the Hash Chains implementation, the tag progresses through a sequence of IDs generated by a PRNG. The algorithm, parameters, and seed ID are known to both the reader-side software and the tag so they both know the progression of the ID sequence. The Hash Chains method is an expansion upon the Pseudonyms method where the sequence of IDs the tag progresses through is generated instead of stored and can be much longer and can be more difficult to analyze. There is a tradeoff

between the Pseudonyms and Hash Chains methods in that Pseudonyms would normally require more memory on the tag to implement and Hash Chains would require more computational power. However, Hash Chains have the potential to be much more secure. It is possible, if the tag progresses through the sequence only when read by a reader or when entering or leaving the read range, that Hash Chains can be implemented on a passive tag. However, given, the increased computation required, it is more difficult than for Pseudonyms. There are several PRNGs that can be used that require vastly different degrees of computational power and whether the tag would need to be passive, active, or semi-active would depend upon the amount of security needed and the PRNG chosen. The security of the PRNG depends to a large extent on how random the generated sequence appears and the randomness of the LCG implemented is explored in Section 5.2.

5.2 Testing the LCG

To test the Linear Congruential Generator in the Hash Chains implementation, I used John Walker's program: ENT . A Pseudorandom Number Sequence Test Program. [24]. ENT applied various tests to a file containing a one megabyte sequence of 8-bit numbers generated by the LCG. The code used to generate the sequence can be found in Appendix C. The results of the ENT test are given in Table 5.1.

A more detailed explanation of the tests is given in Appendix C.

The LCG did not perform well in the Chi square test. Whereas a result between 10% and 90% would indicate a completely random sequence of numbers, the LCG scored 99.99%, indicating that is it significantly non-random with respect to a Chi square test. This is not a surprising result as many PRNGs, including the Unix rand() function, have similar results for the Chi square test [24]. In the remaining four tests, Entropy, Arithmetic mean value, Monte Carlo value, and Serial correlation, the LCG performed very well

Table 5.1: PRNG Analysis of LCG using ENT

Test	Results	Ideal Results
Entropy	8.000000 bits per byte. Optimum compression would reduce the size of this 1000000 byte file by 0%.	8 bits per byte
Chi square distribution for 1000000 samples	distribution is 0.01, and randomly would exceed this value 99.99% of the times	Varies. Truly random is between 10% and 90%
Arithmetic mean value of data bytes	127.4996	127.5 = random
Monte Carlo value for Pi	3.125004500	pi (error 0.53 percent)
Serial correlation coefficient	0.013502	totally uncorrelated = 0.0

5.3 Reader-Side Behavior for the Hash Chains Method

For Private Databases and Pseudonyms, the reader's behavior is simple and straightforward. It only needs to compare the ID it receives from a tag to its list of IDs. For Private Databases, if the ID matches one of the IDs in its database, it can match the ID up with the tag info. For Pseudonyms, the reader need only remember a set of IDs for each tag. However, for Hash Chains, the reader cannot easily remember all of the IDs of a tag. In fact, if a good PRNG is used the period of the ID sequence will be very long. In the case of the LCG implementation used here, the period can be as great as 16777216. Other PRNGs may have periods that are longer still. It may be impractical or even impossible for a reader to remember all the sequences of each of the tags it reads and to search through those sequences to match and ID to a tag. Luckily there exist other options.

One possibility, if a tag is not expected to progress very far down an ID sequence, is that the reader can generate and store a small subset of the sequence and search through that segment when it receives ID from tag. This is the case with the Hash Chain method implemented on the Tag Emulator. However, if the initial (i.e. seed)

IDs and the PRNG parameters of the tags are all different for security reasons, then the reader must search through segments of ID sequences for all of the tags it's keeping track of. Again this may become impractical. It is possible to generate only a very small subset of the sequences, then search and if a match is not found, generate another subset of the sequences. This raises the question of when a reader should stop looking down the hash chain and decide that the tag is an unrecognized. This is a major problem of this particular reader-side implementation. It is possible to progress down a sequence of IDs far enough that the chances of a false negative (declaring a recognized tag unrecgonizable) are acceptably low. However, memory, time, or computational constraints may not allow this. Additionally, in some applications, no level of false negatives may be acceptable and, in such a case, an alternative implementation would be needed.

Another option is to incorporate a counter in the tag that counts what position along the sequence it is at. This counter can be stored in separate memory accessible by the reader or it can be stored in the ID itself. If it is stored in the ID, it may be more obvious to a reader outside the trusted set of readers that the tag is implementing Hash Chains and it may lead to simpler analysis of the sequence. However, before the sequence is cracked, a non-trusted reader would not be able to determine the future progression of the sequence and would not necessarily be able to tell one tag from another. Additionally, the bits of the counter can be located in non-consecutive or changing positions along the n-bit ID of the tag so that they are not easily discernable as bits of a counter. A variation on the counter is to reset the counter bits and the seed ID on the reader and tag every time the tag is read and recognized by a trusted reader. The counter reset and new seed ID would be communicated to other readers on the network of trusted readers. This requires synchronization of the readers and write-access to the counter bits by all trusted readers, but it can significantly reduce the amount of time and computation needed to recognize a tag [13].

If the tag is read frequently by readers within its trusted set readers, then it may be practical to change the ID of the tag only when it is read by a trusted reader. This way the reader will know exactly what the tag ID will be next time. Problems arise

if there is a long span of time between reads by a trusted reader where the tag ID stays the same and it is vulnerable to tracking. With this approach it is also possible to run the hash algorithm on the reader side where computation power and memory are not as limited and high quality and secure algorithms can be used.

5.4 Evaluation of Hardware Design

The Tag Emulator was designed with the competing interests of having the PCB board simple and small yet still being flexible, powerful, and large enough for easy and fast prototyping. After building and working with the Tag Emulator, it was apparent that it was more powerful than required and larger than necessary for easy construction.

The microprocessor was overpowered for the LCG algorithm used in the most computationally intensive of the implemented privacy methods, and it may have been possible to build the tag emulator using a much smaller microcontroller at the level of an Atmel ATtiny AVR. This would decrease the size and cost of the final board. An ATmega would be more suitable if an algorithm like the Mersenne twister PRNG was used.

The spacing and size of the additional capacitors, resistors, and LEDs and the spacing of the headers and switches was also more than required for easy assembly and soldering. A revised board could be constrained in size only by the size of the battery and the battery itself could be smaller, making the revised board about one third of the size of the current Tag Emulator.

5.5 Future Work

5.5.1 EPC Generation2

Amongst the privacy control methods implemented, only Private Databases used the EPC Gen2 protocol. Pseudonyms and Hash Chains used the Tagsense Tag Modem, which uses Gen1. Gen2 has several benefits over Gen1 including faster reads and

writes, dense reader operation for environments with many readers, improved and more reliable read algorithms, and optional memory [8]. In addition it is fast becoming the accepted RFID standard and is already widely used.

The transition from Gen1 to Gen2 can be made by using a device similar the Tagsense Tag Modem that uses Gen2 instead of Gen1 or by designing a Tag Emulator that encompasses the capabilities of the Privacy Control Mainboard and the Tagsense Tag Modem in that it handles both the privacy methods and the EPC protocol. It can be designed to work with Gen2 or any RFID standard of the time.

5.5.2 Combined Privacy Control Mainboard and Tag Emulator

Developing a device that combines the capabilities of the Privacy Control Mainboard and Tag Emulator would allow for better control over the privacy methods implemented. In redesigning the board, information such as the power of the reader signal can be measured. This can be used to determine if a reader is close enough to allow reading or writing of the tag or it can be used to signal when to change the ID when using Pseudonyms or Hash Chains. If a reader-writer pair is developed, the communications protocol itself can be altered to allow for challenge-response authentication, communications of private data such as algorithm parameters, and other additional functions. The Tag Emulator could be smaller, less expensive to make, and more rugged. It can also be designed to work with EPC Gen2 or other RFID standards as described in Section 5.5.1.

Appendix A

Tag Emulator PCB

Table A.1: RFID Tag Emulator Parts List

Manufacturer	Model	Description	Package
Atmel	ATMEGA168V	AVR MCU 16K	32TQFP
Panasonic-SSG	LNJ316C8TRA	Green LED	1206
Grayhill	78B02ST	2 pos switch	custom thr hole
MPD	BU2032-SM-HD-G	2032 coin batt hldr	custom SMT
Rohm	TCA1A156M8R	Tant 15uF Cap	1206
Panasonic	ERJ-P06J301V	300ohm Res	0805
Tyco	4-103186-0-10	2-row hdr	.1" thr hole
Molex	22-28-4360	1-row hdr	.1" thr hole

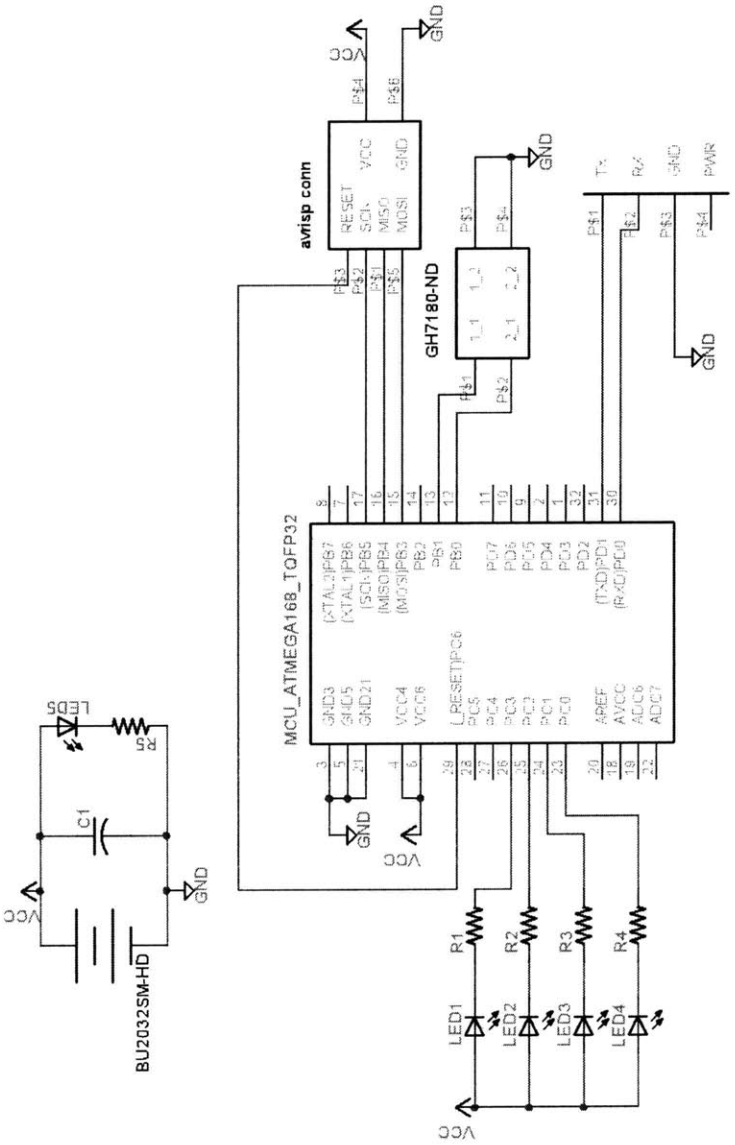

Figure A-1: Tag Emalator PCB Schematic

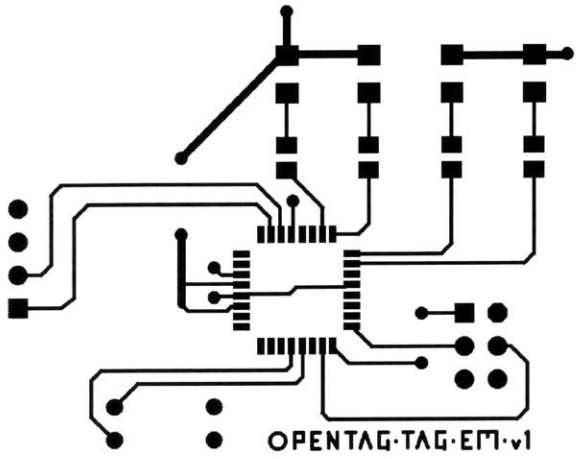

Figure A-2: Tag Emalator PCB. Top Copper Layer. Printed by Advanced Circuits.

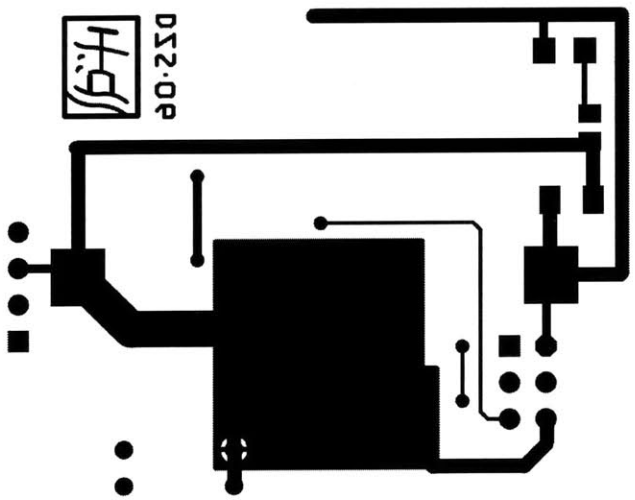

Figure A-3: Tag Emalator PCB. Bottom Copper Layer. Printed by Advanced Circuits.

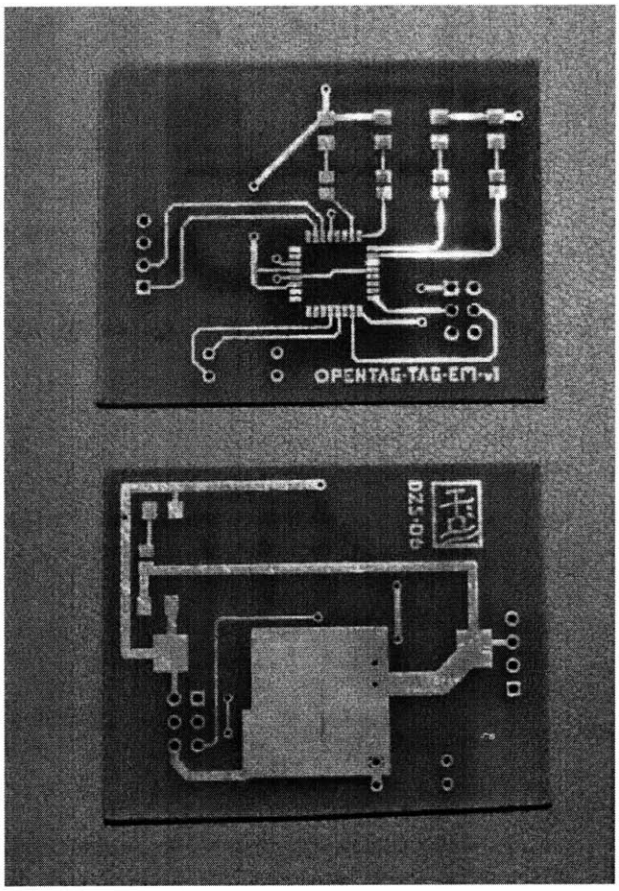

Figure A-4: Tag Emulator PCB - Bare

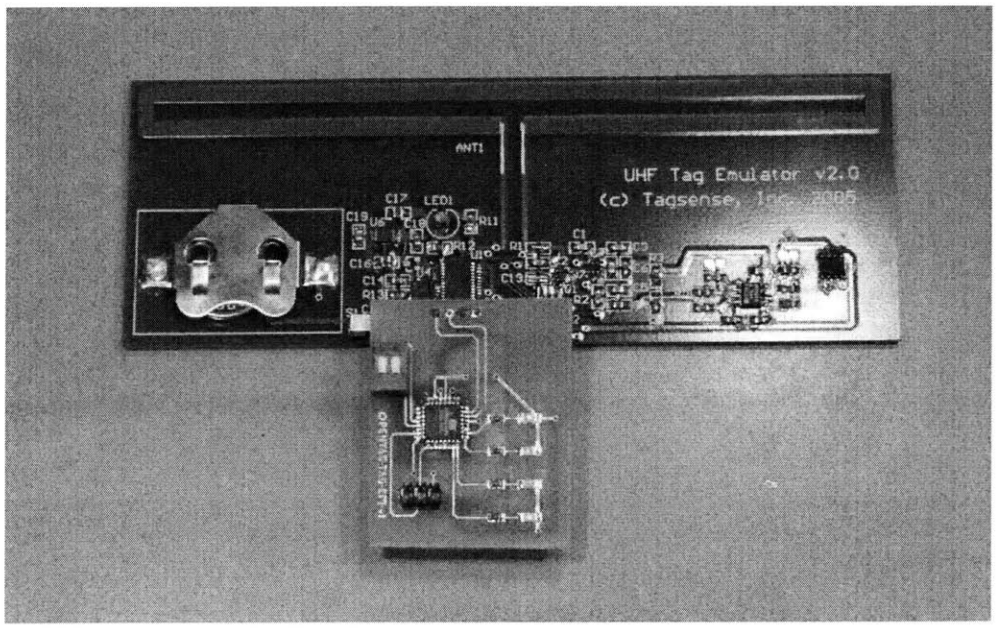

Figure A-5: Tag Emulator on Thingmagic Tagmodem

Appendix B

Tag Emulator Source Code

The following contains code for both the tag side and reader side implementations of the Private Databases, Pseudonyms, and Hash Chains privacy control methods. The tag side C code is listed first, followed by the reader side Python code.

The formatting of the code was done using Lgrind for the C code and SciTE for the Python code. Although the code was reviewed after formatting, it was altered from its original form to fit within the confines of the thesis format and may contains errors as a result. Please review carefully before using.

B.1 Code - pseudonyms_pcb.tex

This section contains the C code for the Pseudonyms privacy control method that is written for the Tag Emulator running an Atmel ATmega168V.

```
// pseudonyms_pcb.c
// Modifited from AVR306: Using the AVR UART in C
// Routines for polled USART on the Atmega168V
// Last modified: 2006-08-16
// Modified by: Danny Shen

/* Includes */
#include <avr/io.h>
#include <string.h>
#include <util/delay.h>

/* Prototypes */
void USART0_Init( unsigned int baudrate);
unsigned char USART0_Receive( void );
void USART0_Transmit( unsigned char data);
void USART0_Transmit_String(char data[]);
char* key2string(int key);
char* key2string_r(int key);

/* Main - a simple test program*/
int main( void ){

    /* Set the baudrate to 38,400 bps using 8.0MHz internal clock */
        USART0_Init( 12 );

        DDRB = 0b00000000; //portb set to inputs
        PORTB = 0xFF; //enable internal pull-up resistors

        DDRC = 0b00001111; //set bits 0-4 of portc to output
        PORTC = 0b11111110; //turns on LED at pc0

        for(;;){/* Forever */

                //testing switches
                if (PINB & 0x01){//if switch 1 is off
                        PORTC &= 0b11110111;//turn on LED at pc3
                }
                else{//if switch 1 is off
                        PORTC |= 0b00001000;//turn off LED at pc3
                }

                for(int i=0; i<10;)
                {
                        char char_in = USART0_Receive();
                        //PORTB |= 0b00000100;//turn off LED at pb2

                        if(char_in == '>'){ //if tag emulator is ready
```

```
                        //acknowledge ready to send
                        PORTC &= 0b11111101;//turn on LED at pc1
                        PORTC |= 0b00000001;//turn off LED at pc0


                        //determine which mode of pseudonyms are used
                        if (PINB & 0x01){//if sw1 is off, ordered IDs
                                USART0_Transmit_String(key2string(i));
                                _delay_loop_2(80000);
                        }
                        else{//do random IDs
                                USART0_Transmit_String(key2string_r(i));
                                _delay_loop_2(80000);
                        }
                        i++;

                        //no longer ready to send
                        PORTC |= 0b00000010;//turn off LED at pc1
                        PORTC &= 0b11111110;//turn on LED at pc0
                }//end if

                else if (char_in == 'q'){//good response
                        PORTC &= 0b11111011;//turn on LED at pc2
                        _delay_loop_2(300000);
                }//end else if

        }//end for
}//end for

return 1;

}//end main


/* Initialize UART */
void USART0_Init( unsigned int baudrate ){

        /* Set the baud rate */
        UBRR0H = (unsigned char) (baudrate>>8);
        UBRR0L = (unsigned char) baudrate;

        /* Enable UART receiver and transmitter */
        UCSR0B = ( ( 1 << RXEN0 ) | ( 1 << TXEN0 ) );

        /* Set frame format: 8 data 1 stop */
        UCSR0C = (3<<UCSZ00);
```

```
}//end USART0_Init


/* Read and write functions */
unsigned char USART0_Receive( void ){

        /* Wait for incomming data */
        while ( !(UCSR0A & (1<<RXC0)) )
                ;
        /* Return the data */
        return UDR0;

}//end USART0_Receive


/* transmit a single char via USART */
void USART0_Transmit( unsigned char data ){

        /* Wait for empty transmit buffer */
        while ( !(UCSR0A & (1<<UDRE0)) )
                ;
        /* Start transmittion */
        UDR0 = data;

}//end USART0_Transmit


/* transmit all chars in a data */
void USART0_Transmit_String(char data[]){

        char *cp;

        cp = data;
    while(*cp != 0){
        USART0_Transmit(*cp);
        cp++;
    }

}//end USART0_Transmit_String


/* returns sequential list of IDs based on key*/
char* key2string(int key){

        switch(key){
        case 0: return "E0123456789abcde0\r";
        case 1: return "E0123456789abcde1\r";
```

```
        case 2: return "E0123456789abcde2\r";
        case 3: return "E0123456789abcde3\r";
        case 4: return "E0123456789abcde4\r";
        case 5: return "E0123456789abcde5\r";
        case 6: return "E0123456789abcde6\r";
        case 7: return "E0123456789abcde7\r";
        case 8: return "E0123456789abcde8\r";
               default : return "E0123456789abcde9\r";
        }

}//end key2string


/* returns non-sequential list of IDs based on key*/
char* key2string_r(int key){

  if(key == 0) return "E05634b654cde5ff6\r";
  else if(key == 1) return "E16546546a4421232\r";
  else if(key == 2) return "E65ee954878745411\r";
  else if(key == 3) return "Eab654c654fe56d54\r";
  else if(key == 4) return "Ecd54654ad54b65f5\r";
  else if(key == 5) return "E9871a1161d551103\r";
  else if(key == 6) return "Ea4c5d554f1e221d4\r";
  else if(key == 7) return "E65465484f84b11c2\r";
  else if(key == 8) return "E546541d548787454\r";
  else return "E1161d56781161d56\r";

}//end key2string_r
```

B.2 Code - lcg_pcb.c

This section contains the C code for the Hash Chains privacy control method that is written for the Tag Emulator running an Atmel ATmega168V. The Linear Congruential Generator (LCG) is the algorithm used to generate the sequence of IDs.

```
// lcg_pcb.c
// Modifited from AVR306: Using the AVR UART in C
// Routines for polled USART on the Atmega168V
// Last modified: 2006-08-16
// Modified by: Danny Shen

/* Includes */
#include <avr/io.h>
#include <string.h>
#include <util/delay.h>

/* Prototypes */
void USART0_Init( unsigned int baudrate);
unsigned char USART0_Receive( void );
void USART0_Transmit( unsigned char data);
void USART0_Transmit_String(char data[]);
char* key2string(int key);
char* key2string_r(int key);


/* Main - Implements Linear Congruential Generator */
int main( void ){

  // Set baudrate to 38,400bps with 8.0MHz internal clock
  USART0_Init( 12 );

  DDRB = 0b00000000; //portb set to inputs
  PORTB = 0xFF; //enable internal pull-up resistors

  DDRC = 0b00001111; //set bits 0-4 of portc to output
  PORTC = 0b11111110; //turns on LED at pc0


  int ID_SIZE = 8; //number of bytes in ID

  //elements that define behavior of LCG algorithm
  int m=256;
  int a[ID_SIZE]={37, 17, 1, 121, 209, 9, 73, 157};
  int b[ID_SIZE]={23, 199, 31, 223, 167, 5, 103, 251};

  int v_new[ID_SIZE];//array of ID bytes
  char id[2*ID_SIZE];//char array of ID

  //initial ID
  //can be given a value here
  //or can be input from RFID Reader
  int v_old[ID_SIZE];
```

```
v_old[0] = 1;
v_old[1] = 129;
v_old[2] = 67;
v_old[3] = 28;
v_old[4] = 250;
v_old[5] = 157;
v_old[6] = 94;
v_old[7] = 161;

for(;;){/* Forever */

  char char_in = USART0_Receive();
  //PORTB |= 0b00000100;//turn off LED at pb2

  if(char_in == '>'){ //if tag emulator is ready
    //USART0_Transmit_String("-- > sent --\n");

    //acknowledge ready to send
    PORTC &= 0b11111101;//turn on LED at pc1
    PORTC |= 0b00000001;//turn off LED at pc0

    //apply LCG to each byte and store new in id
    for(int i=0; i<ID_SIZE; i++){
      v_new[i] = (a[i] * v_old[i] + b[i]) % m;

      v_old[i] = v_new[i];

      //extract characters from byte
      //adding 48 to int gives char
      id[2*i+1] = v_new[i]%16 + 48;
      id[2*i] = v_new[i]/16 + 48;
    }


    USART0_Transmit_String(id);
    _delay_loop_2(80000);

    //no longer ready to send
    PORTC |= 0b00000010;//turn off LED at pc1
    PORTC &= 0b11111110;//turn on LED at pc0
  }//end if

  //turn on led to indicate a good response
  else if (char_in == 'q'){
    PORTC &= 0b11111011;//turn on LED at pc2
    _delay_loop_2(300000);//leave LED at pc2 on
  }//end else if
```

```
  }//end for

  return 1;

}//end main


/* Initialize UART */
void USART0_Init( unsigned int baudrate ){

  /* Set the baud rate */
  UBRR0H = (unsigned char) (baudrate>>8);
  UBRR0L = (unsigned char) baudrate;

  /* Enable UART receiver and transmitter */
  UCSR0B = ( ( 1 << RXEN0 ) | ( 1 << TXEN0 ) );

  /* Set frame format: 8 data 1 stop */
  UCSR0C = (3<<UCSZ00);

}//end USART0_Init


/* Read and write functions */
unsigned char USART0_Receive( void ){

  /* Wait for incomming data */
  while ( !(UCSR0A & (1<<RXC0)) )
    ;
  /* Return the data */
  return UDR0;

}//end USART0_Receive


/* send a single char through USART */
void USART0_Transmit( unsigned char data ){

  /* Wait for empty transmit buffer */
  while ( !(UCSR0A & (1<<UDRE0)) )
    ;
  /* Start transmittion */
  UDR0 = data;

}//end USART0_Transmit
```

```
/* send a whole ID */
void USART0_Transmit_String(char data[]){

  char *cp;

  cp = data;

  USART0_Transmit('E');//start char

  while(*cp != 0){
    USART0_Transmit(*cp);
    cp++;
  }

  USART0_Transmit('\r');//end char

}//end USART0_Transmit_String
```

B.3 Code - privateDBReader.py

This section contains the Python code for the Private Databases privacy control method that is written to work for the Thingmagic Mercury4 reader. This and all the other reader-side Python code was written with help from Brent Fitzgerald at the Physical Language Workshop at the MIT Media Lab.

```
    #!/usr/bin/python

#privateDBReader.py
#reader side implementation of Private Databases


from  readtags  import  multirange

import  sys ,  random ,  math ,  pygame ,  re ,  urllib ,  xmlrpclib
from  pygame . locals  import  *
import  Image ,  ImageFile
from  threading  import  Thread
import  sys ,  time ,  telnetlib ,  socket


host  =  '18.85.16.183'

#dict linking IDs to tagged items
tag_data  =  { "0x0123456789ABCDEF0123456A"  :
                        pygame . image . load ( "CD_boltonCD.jpg" ),
                    "0x1234567890ABCDEF0000DBDD"  :
                        pygame . image . load ( "tolkien.jpg" )
                    }

notfoundimg  =  pygame . image . load ( "notfound.png" )




#query the Mercury4 reader at IP
#address host and port 8080

def  query ( host ,  q ,  runtime = 1 ,  port = 8080 ):

    tn  =  telnetlib . Telnet ()

    print  '-- Constructed query "%s"'  %  ( q )
    print  "-- Connecting to %s..." % host
    try :
        tn . open (  host ,  port  )
    except  socket . error :
        raise  "Error 2:  " + host + " sql server is down"
    print  "-- connected."

    print  '-- Sending query'
```

```
    tn . write ( q )

    start = time . time ()
    count = 0

    # result set
    #   - key: (tag id, antenna_id),   value: read_count
    results = {}

    # Process search results as they come in
    while time . time ()- start <= runtime :
        output = tn . read_until ( "\n" , 1 )

        # Process line
        if output and output != "\n" :
                output = output . strip ()
                output = output [ 0 : len ( output )- 4 ]
                try :
                        if results . has_key ( output ):
                                results [ output ] += 1
                        else :
                                results [ output ] = 1

                except ( ValueError ):
                        print "Can't parse line \"%s\"" % output

    # Stop search
    tn . write ( "SET auto=OFF;" )
    tn . close ()
    print "-- Connection closed, done!"

    return results



def main ():

        pygame . init ()

        modes = pygame . display . list_modes ()
        mode = modes [ 4 ]
        screen = pygame . display . set_mode ( mode )
        width = mode [ 0 ]
        height = mode [ 1 ]
```

```
        done = 0

        clock = pygame . time . Clock ()
        while not done :
                clock . tick ( 30 )
                screen . fill (( 0 , 0 , 0 ))

                results = query ( host , "SELECT id from tag_id;" , 2 )

                print results
                x = 10
                y = 10

                for r in results . keys ():
                        if tag_data [ r ]:
                                i = tag_data [ r ]
                        else :
                                i = notfoundimg . convert_alpha ()
                        r = pygame . Rect (( x , y ), i . get_size ())
                        screen . blit ( i , r )
                        x += i . get_width ()

                pygame . display . update ()
                for e in pygame . event . get ():
                        if e . type == QUIT or \
                            ( e . type == KEYUP and e . key == K_ESCAPE ):
                                done = 1
                                break

if __name__ == '__main__' :
                main ()
```

B.4 Code - pseudonymsReader.py

This section contains the Python code for the Pseudonyms privacy control method that is written to work for the Thingmagic Mercury4 reader.

```
    #!/usr/bin/python

#pseudonymsReader.py
#reader side implementation of Pseudonyms


from  readtags  import  multirange

import  sys ,  random ,  math ,  pygame ,  re ,  urllib ,  xmlrpclib
from  pygame . locals  import  *
import  Image ,  ImageFile
from  threading  import  Thread
import  sys ,  time ,  telnetlib ,  socket


host  =  '18.85.16.183'

#dict linking IDs to images of tagged items
tag_data  =  { "0123456789abcde0"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde1"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde2"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde3"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde4"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde5"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde6"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde7"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde8"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "0123456789abcde9"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "05634b654cde5ff6"  :  \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "16546546a4421232"  :  \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "65ee954878745411"  :  \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "ab654c654fe56d54"  :  \
```

```
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "cd54654ad54b65f5"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "9871a1161d551103"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "a4c5d554f1e221d4"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "65465484f84b11c2"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "546541d548787454"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" ),
                    "1161d56781161d56"   :   \
                        pygame . image . load ( "Book_tolkien.jpg" )
                }

#image to load if no ID not recognized
notfoundimg  =  pygame . image . load ( "notfound.png" )



#query the Mercury4 reader at IP
#address host and port 8080

def  query ( host ,  q ,  runtime = 1 ,  port = 8080 ):

    tn  =  telnetlib . Telnet ()

    print  '-- Constructed query "%s"'  %  ( q )
    print  "-- Connecting to %s..." % host
    try :
        tn . open (  host ,  port  )
    except  socket . error :
        raise  "Error 2:  " + host + " sql server is down"
    print  "-- connected."

    print  '-- Sending query'
    tn . write  ( q )

    start  =  time . time ()
    count  =  0

    # result set
    #  - key:  (tag id, antenna_id),  value:  read_count
    results  =  {}
```

```
    # Process search results as they come in
    while   time .  time ()- start   <=   runtime :
        output   =   tn .  read_until ( "\n" , 1 )

        # Process line
        if   output   and   output   !=   "\n" :
                output   =   output .  strip ()
                output   =   output [ 0 :  len ( output )- 4 ]
                try :
                        if   results .  has_key ( output ):
                                results [ output ]   +=   1
                        else :
                                results [ output ]   =   1

                except   ( ValueError ):
                        print   "Can't parse line \"%s\"" % output

    # Stop search
    tn .  write ( "SET auto=OFF;" )
    tn .  close ()
    print   "-- Connection closed, done!"

    return   results



def   main ():

        pygame .  init ()

        modes   =   pygame .  display .  list_modes ()
        mode   =   modes [ 4 ]
        screen   =   pygame .  display .  set_mode ( mode )
        width   =   mode [ 0 ]
        height   =   mode [ 1 ]
        done   =   0

        clock   =   pygame .  time .  Clock ()
        while   not   done :
                clock .  tick ( 30 )
                screen .  fill (( 0 , 0 , 0 ))

                results   =   query ( host ,   "SELECT id from tag_id;" ,   2 )
```

```
                print   results
                x   =   10
                y   =   10

                for   r   in   results .  keys ():
                        #note that substring of result key is used
                        #this is because of how data from Mercury4
                        #reader is formatted
                        if   tag_data [ r [ 2 :  18 ]]:
                                i   =   tag_data [ r [ 2 :  18 ]]
                        else :
                                i   =   notfoundimg .  convert_alpha ()
                        r   =   pygame .  Rect (( x , y ),   i .  get_size ())
                        screen .  blit ( i , r )
                        x   +=   i .  get_width ()

                pygame .  display .  update ()
                for   e   in   pygame .  event .  get ():
                        if   e .  type   ==   QUIT   or  \
                            ( e .  type   ==   KEYUP   and   e .  key   ==   K_ESCAPE ):
                                done   =   1
                                break

if   __name__   ==   '__main__' :
                 main ()
```

B.5 Code - lcgReader.py

This section contains the Python code for the Hash Chains privacy control method that is written to work for the Thingmagic Mercury4 reader.

```
    #!/usr/bin/python

#lcgReader.py
#reader side implementation of Hash Chains
#using the Linear Congruential Generator


from  readtags  import  multirange

import  sys ,  random ,  math ,  pygame ,  re ,  urllib ,  xmlrpclib
from  pygame . locals  import  *
import  Image ,  ImageFile
from  threading  import  Thread
import  sys ,  time ,  telnetlib ,  socket


host  =  '18.85.16.183'

#dict linking IDs to tagged items
tag_data  =  { "0123456789abcde9"  :  \
                        pygame . image . load ( "CD_bolton.jpg" ),
                    "05634b654cde5ff6"  :  \
                        pygame . image . load ( "Book_tolkien.jpg" )
                    }

notfoundimg  =  pygame . image . load ( "notfound.png" )


#lcg algorithm parameters
m  =  256
lsta  =  [ 37 ,  17 ,  1 ,  121 ,  209 ,  9 ,  73 ,  157 ]
lstb  =  [ 23 ,  199 ,  31 ,  223 ,  167 ,  5 ,  103 ,  251 ]

#initial ID
lst_initialID  =  [ 1 , 129 , 67 , 28 , 250 , 157 , 94 , 161 ]

# number of IDs that will be generated
# by reader using LCG
numids  =  100



#generates a dict of IDs using the LCG
#and parameters that is numids long
```

```
def   generate_IDs ( lst_oldID ):

        strid   =   ""

        #form ID string out of lst_oldID
        for   i   in   range ( len ( lst_oldID )):
                #concatenate ints into id string
                strid   +=   str ( lst_oldID [ i ]// 16 )
                strid   +=   str ( lst_oldID [ i ]% 16 )

        #input string of vold as first ID in tag_data
        tag_data   =   { strid   :    \
                                pygame .  image .  load ( "CD_boltonCD.jpg" )
                        }


        #input strings of generated IDs
        for   i   in   range ( numids ):
                strid   =   ""

                #apply LCG to each element in the list
                for   i   in   range ( len ( lst_oldID )):

                        #lcg algorithm
                        lst_newID [ i ]   =   ( lsta [ i ] * lst_oldID [ i ] + lstb [ i ]) % m
                        lst_oldID [ i ]   =   lst_newID [ i ]

                        #concatenate ints into id string
                        strid   +=   str ( lst_newID [ i ]// 16 )
                        strid   +=   str ( lst_newID [ i ]% 16 )



#query the Mercury4 reader at IP
#address host and port 8080
def   query ( host ,   q ,   runtime = 1 ,   port = 8080 ):

    tn   =   telnetlib .  Telnet ()

    print   '-- Constructed query "%s"'   %   ( q )
    print   "-- Connecting to %s..." % host
    try :
        tn .  open (   host ,   port   )
    except   socket .  error :
        raise   "Error 2:  " + host + " sql server is down"
```

```
    print   "-- connected."

    print   '-- Sending query'
    tn . write   ( q )

    start   =   time . time ()
    count   =   0

    # result set
    #   - key:  (tag id, antenna_id),   value:  read_count
    results   =   {}

    # process search results as they come in
    while   time . time ()- start   <=   runtime :
        output   =   tn . read_until ( "\n" , 1 )

        # process line
        if   output   and   output   !=   "\n" :
                output   =   output . strip ()
                output   =   output [ 0 :  len ( output )- 4 ]
                try :
                        if   results . has_key ( output ):
                                results [ output ]   +=   1
                        else :
                                results [ output ]   =   1

                except   ( ValueError ):
                        print   "Can't parse line \"%s\"" % output

    # Stop search
    tn . write ( "SET auto=OFF;" )
    tn . close ()
    print   "-- Connection closed, done!"

    return   results




def   main ():

        pygame . init ()
        generate_IDs ( lst_initialID )
```

```
modes = pygame . display . list_modes ()
mode = modes [ 4 ]
screen = pygame . display . set_mode ( mode )
width = mode [ 0 ]
height = mode [ 1 ]
done = 0

clock = pygame . time . Clock ()
while not done :
        clock . tick ( 30 )
        screen . fill (( 0 , 0 , 0 ))

        results = query ( host , "SELECT id from tag_id;" , 2 )

        print results
        x = 10
        y = 10

        for r in results . keys ():

                #note that substring of result key is used
                #this is because of how data from Mercury4
                #reader is formatted
                rsub = r [ 2 : 18 ]

                #if the ID is found
                if tag_data [ rsub ]:
                        i = tag_data [ rsub ]

                        lstID = [ 0 ]
                        #create an ID list from string
                        for i in range ( 8 ):
                                hbit = int ( rsub [ 2 * i ]) #low bit
                                lbit = int ( rsub [ 2 * i + 1 ]) #high bit
                                lstID . append ( 16 * hbit + lbit )

                        #generate new list of IDs
                        #starting from the latest ID
                        generate_IDs ( lstID )

                else :
                        i = notfoundimg . convert_alpha ()

                r = pygame . Rect (( x , y ), i . get_size ())
                screen . blit ( i , r )
```

```
                x   +=   i .  get_width ()

        pygame .  display .  update ()
        for   e   in   pygame .  event .  get ():
                if   e .  type   ==   QUIT   or  \
                    ( e .  type   ==   KEYUP   and   e .  key   ==   K_ESCAPE ):
                        done   =   1
                        break

if   __name__   ==   '__main__' :
         main ()
```

Appendix C

Testing of Pseudo Random Number Generators

C.1 ENT - A Pseudorandom Number Sequence Test Program

The following is the description of John Walker's ENT program that was used to analyze the Linear Congruential Generator and was taken from [24].

This page describes a program, ent, which applies various tests to sequences of bytes stored in files and reports the results of those tests. The program is useful for those evaluating pseudorandom number generators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a file is of interest.

C.2 Overview

ENT performs a variety of tests on the stream of bytes in infile (or standard input if no infile is specified) and produces output as follows on the standard output stream:

- Entropy = 7.980627 bits per character.
- Optimum compression would reduce the size of this 51768 character file by 0 percent.
- Chi square distribution for 51768 samples is 1542.26, and randomly would exceed this value 0.01 percent of the times.
- Arithmetic mean value of data bytes is 125.93 (127.5 = random). Monte Carlo value for Pi is 3.169834647 (error 0.90 percent). Serial correlation coefficient is 0.004249 (totally uncorrelated = 0.0).

C.3 Description of Calculated Values

C.3.1 Entropy

The information density of the contents of the file, expressed as a number of bits per character. The results above, which resulted from processing an image file compressed with JPEG, indicate that the file is extremely dense in information–essentially random. Hence, compression of the file is unlikely to reduce its size. By contrast, the C source code of the program has entropy of about 4.9 bits per character, indicating that optimal compression of the file would reduce its size by 38

C.3.2 Chi-square Test

The chi-square test is the most commonly used test for the randomness of data, and is extremely sensitive to errors in pseudorandom sequence generators. The chi-square distribution is calculated for the stream of bytes in the file and expressed as

an absolute number and a percentage which indicates how frequently a truly random sequence would exceed the value calculated. We interpret the percentage as the degree to which the sequence tested is suspected of being non-random. If the percentage is greater than 99% or less than 1%, the sequence is almost certainly not random. If the percentage is between 99% and 95% or between 1% and 5%, the sequence is suspect. Percentages between 90% and 95% and 5% and 10% indicate the sequence is "almost suspect". Note that our JPEG file, while very dense in information, is far from random as revealed by the chi-square test.

Applying this test to the output of various pseudorandom sequence generators is interesting. The low-order 8 bits returned by the standard Unix rand() function, for example, yields:

- Chi square distribution for 500000 samples is 0.01, and randomly would exceed this value 99.99 percent of the times.
- While an improved generator [Park & Miller] reports:
- Chi square distribution for 500000 samples is 212.53, and randomly would exceed this value 95.00 percent of the times.
- Thus, the standard Unix generator (or at least the low-order bytes it returns) is unacceptably non-random, while the improved generator is much better but still sufficiently non-random to cause concern for demanding applications. Contrast both of these software generators with the chi-square result of a genuine random sequence created by timing radioactive decay events.
- Chi square distribution for 32768 samples is 237.05, and randomly would exceed this value 75.00 percent of the times.

C.3.3 Arithmetic Mean

This is simply the result of summing the all the bytes (bits if the -b option is specified) in the file and dividing by the file length. If the data are close to random, this should be about 127.5 (0.5 for -b option output). If the mean departs from this value, the values are consistently high or low.

C.3.4 Monte Carlo Value for Pi

Each successive sequence of six bytes is used as 24 bit X and Y co-ordinates within a square. If the distance of the randomly-generated point is less than the radius of a circle inscribed within the square, the six-byte sequence is considered a "hit". The percentage of hits can be used to calculate the value of Pi. For very large streams (this approximation converges very slowly), the value will approach the correct value of Pi if the sequence is close to random. A 32768 byte file created by radioactive decay yielded:

- Monte Carlo value for Pi is 3.139648438 (error 0.06 percent).

C.3.5 Serial Correlation Coefficient

This quantity measures the extent to which each byte in the file depends upon the previous byte. For random sequences, this value (which can be positive or negative) will, of course, be close to zero. A non-random byte stream such as a C program will yield a serial correlation coefficient on the order of 0.5. Wildly predictable data such as uncompressed bitmaps will exhibit serial correlation coefficients approaching 1. See [Knuth, pp. 64-65] for more details.

C.4 Code - Lcg_bytes.java

```
// Lcg_bytes.java
// Sequence of bytes generated by
// Linear Congruential Generator
// Last modified: 2006-08-21
// Modified by: Danny Shen

import java.io.*;

/* generates a 1 MB sequence using an 8-bit LCG algorithm */
public class Lcg_bytes {
    public static void main (String argv[]) {

        //number of elements in sequence
        int size = 1000000;

        byte[] ba = new byte[size];//byte array of sequence

        //parameters of LCG
        int a=37;
        int b=23;
        int m=256;

        //variables used by LCG
        int v_old = 1;
        int v_new;

        //generate sequence
        for (int i = 0; i<size; i++){
            //lcg algorithm
            v_new = (a * v_old + b) % m;
            v_old = v_new;

            //store generated number in ba as a byte
            Integer integ = new Integer(v_new);
            ba[i] = integ.byteValue();
        }

        try {//write to file for ent test
            File file = new File("lcg_seq.txt");
            BufferedOutputStream bos
               = new BufferedOutputStream(new FileOutputStream(file));

            bos.write(ba);
            bos.flush();
            bos.close();
        }//end try
```

```
        catch(Exception e) {
        }//end catch

    }//end main

}//end Lcg_bytes
```

Appendix D

Glossary of Acronyms

AVRISP	AVR In-System Programmer
EAGLE	Easily Applicable Graphical Layout Editor
EPC	Electronic Product Code
IC	Integrated Circuit
Gen1	Generation 1 EPC
Gen2	Generation 2 EPC
I/O	Input/Output
ISAAC	Indirection, Shift, Accumulate, Add, and Count
LCG	Linear Congruential Generator
LED	Light-Emitting Diode
PCB	Printed Circuit Board
PRNG	Pseudo Random Number Generator
RFID	Radio Frequency Identification
SQL	Structured Query Language
TaPA	Tag Privacy Agent (in Soft Blocking)
TI	Texas Instruments
TIRIS	Texas Instruments Registration and Identification System
TQFP	Thin Quad Flat Package
USART	Universal Synchronous-Asynchronous Receiver-Transmitter

Bibliography

[1] R. L. Rivest A. Juels and M. Szydlo. Ed. In V. Atluri. *The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy.* 8th ACM Conference on Computer and Communications Security. ACM Press, 2003.

[2] Atmel. *Datasheet: 8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash. ATmega48/V, ATmega88/V, ATmega168/V. Revision 2545F-AVR-06/05.* 2005.

[3] A. Juels. Ed. In C. Blundo. *Minimalist cryptography for low-cost RFID tags.* Security in Communication Networks (SCN). Springer-Verlag, 2004.

[4] MIT Auto-ID Center. *Draft protocol specification for a 900MHz Class 0 radio frequency identification tag.* MIT Auto-ID Center. [online document] Available at: www.epcglobalinc.org [cited Aug 12, 2006], 23 February 2003.

[5] Atmel Corporation. *AVRISP mkII User Guide.*

[6] Atmel Corporation. *AVR ISP In-System Programmer.* [online document] Available at: http://www.atmel.com/dyn/resources/prod_images/AVRISP.jpg [cited Jul. 26, 2006], 2006.

[7] Electro-com. *Radio Frequency Identification: UHF : Supply Chain and Logistics RFID: Inventory Tracking: EPC Global UHF.* [online document] Available at: http://www.electrocom.com.au/rfid_uhf.htm [cited Dec. 13, 2005], 24 August 2005.

[8] EPCglobal. *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID. Protocol for Communications at 860 MHz - 960 MHz. Version 1.0.9.* [online document] Available at: http://www.epcglobalinc.org [cited Jul. 14, 2006], January 2005.

[9] Klaus Finkenzeller. *RFID Handbook.* John Wiley & Sons, Hoboken, NJ, 2 edition.

[10] Simson Garfinkel (Ed.) and Henry Holtzman. Understanding rfid technology. *[Chapter in:] RFID - Applications, Security, and Privacy*, pages 15–36, 2005.

[11] RFID Gazette. *Walmart.* [online document] Available at: http://www.rfidgazette.org/walmart/ [cited Dec. 15, 2005], 13 December 2005.

[12] Toby Dylan Hocking. *Statistical Computing. [Based on lectures by Professor Phil. Spector Statistics 243. UC Berkeley. Fall 2005.].* January 2006.

[13] Henry Holtzman. *Personal Communications.* 2006.

[14] A. Juels, J. Brainard. Ed. In Sabrina De Capitani di Vimercati, and Paul Syverson. *Soft blocking: Flexible blocker tags on the cheap.* Workshop on Privacy in the Electronic Society - WPES. ACM Press, Washington, DC, October 2004.

[15] Sumit Roy Kenneth P. Fishkin. *Enhancing RFID Privacy via Antenna Energy Analysis.* MIT RFID Privacy Workshop, also Intel Research Seattle Technical Memo IRS-TR-03-012, November 2003.

[16] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley, third edition, 1997.

[17] K. Suzuki M. Ohkubo and S. Kinoshita. *A cryptographic approach to 'privacy-friendly' tags.* RFID Privacy Workshop. MIT, Cambridge, MA, November 2003.

[18] Mary Catherine O'Connor. *SkyeTek Adding Security Support into Passive Platforms.* RFID Journal. [online document] Available at: http://www.rfidjournal.com/article/articleview/2480/1/1/ [cited Jul. 15, 2006], 11 July 2006.

[19] D. Piasecki. *RFID Update: The Basics, The Wal-Mart Mandate, EPC, Privacy Concerns, and More.* Inventory Operations Consulting LLC. [online document] Available at: http://www.inventoryops.com/RFIDupdate.htm [cited Dec. 13, 2005], 18 November 2005.

[20] Rich Redemske. *UHF TagModem INSTRUCTIONS V 1.1.* Tagsense, 2005.

[21] Richard Redemske. An electromagnetic measurement tool for uhf rfid diagnostics. Master's thesis, Massachusetts Institute of Technology, Department of Engineering in Computer Science and Electrical Engineering, September 2005.

[22] S.G. Miremadi S. Timarchi and A.R. Ejlali. *Evaluation of Some Exponential Random Number Generators Implemented by FPGA.* From Proceeding (456) Parallel and Distributed Computing and Networks, 2005.

[23] J. Viega. *Practical Random Number Generation in Software.* 19th Annual Computer Security Applications Conference, December 2005.

[24] John Walker. *Pseudorandom Number Sequence Test Program.* [online document] Available at: http://www.fourmilab.ch/random/ [cited Aug. 20, 2006], 2006.