FPGA-Based Smart RFID Tag with Robust Authentication Protocol

By

Leili Borghei

A Project

Presented to Ryerson University

In partial fulfillment of the

Requirement for the degree of

Master of Engineering

In the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2009

Author's Declaration

I hereby declare that I am the sole author of this thesis project.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

Signature

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

Instructions on Borrowers

Ryerson University requires the signature of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

FPGA-Based Smart RFID Tag With Robust Authentication Protocol

©Leili Borghei, 2009

Master of Engineering

Electrical and Computer Engineering

Ryerson University

Radio Frequency Identification (RFID) technology is being deployed increasingly in diverse applications and has become pervasive and ubiquitous. While the characteristics of RFID make recognition possible without physical contact, it also has many problems pertaining to privacy and security. This has led to slow adaptation of RFID technology for large number of applications. Moreover, any approach without addressing the crucial factors like, scalability, flexibility, cost, performance, computational resources and ease of use is not acceptable for deploying the RFID technology. This project provides an introduction to RFID technology and the privacy and security threats it faces. It reviews recently proposed RFID authentication techniques, and presents an FPGA-based RFID tag with a secure authentication protocol between the tag and reader, addressing all RFID security issues and threats including forward secrecy, eavesdropping, tracking, cloning, replay attack and denial of service attack. The project explores RFID authentication protocol using the Altera's Nios II embedded processor that provides a flexible exploration environment.

Acknowledgments

I would like to greatly thank the supervision and guidance of my supervisor Dr. Gul Khan during this thesis project. I am particularly thankful to him for providing me with supervision in defining and directing this project work. I would like to thank him for his encouragement and inspiration that boosted my enthusiasm in embedded systems development and hardware/software co-design. I am also much grateful to his patience and willingness throughout my work on this project.

I would also like to express my deep appreciation to my supervisor Dr. Kaamran Raahemifar for providing me with the assistance and resources through my entire Master's program. I am much indebted to his kind sharing of his expertise and research insight in searching materials, writing reports, preparing presentations and giving wise advice that have been all invaluable to me. It has been a distinct privilege for me to work with both Dr. Kaamran Raahemifar and Dr. Gul Khan.

Heartly thanks to all of my friends for their diligent help all through my Master's program. I would like to thank one and all in Electrical and Computer Engineering Department of Ryerson University who supported me in pursuing my Master's program.

Finally, I would like to express my greatest gratitude to my lovely daughter Ghazal, my dear husband Alireza, and my dearest parents. I feel proud and consider myself to be fortunate to have their encouragement and support. To them I dedicate this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Radio frequency identification (RFID) is an emerging technology, which brings enormous productivity benefits in applications where objects have to be identified automatically. The main benefits of RFID systems are that they can provide automated and multiple identification capture and analysis. One can read several RFID tags in the field at the same time automatically to track valuable objects [1]. However, while the RFID feature of recognition without physical contact provides convenience to the user; many problems pertaining to privacy and security still exist. This has led to the slow adaptation of RFID systems in a number of applications.

1.1 Structure of RFID Systems

RFID uses radio frequency for information transfer between tags and readers. Generally, an RFID system consists of three components: RFID tags, RFID readers, and back-end servers with databases, as shown in Figure 1.1.

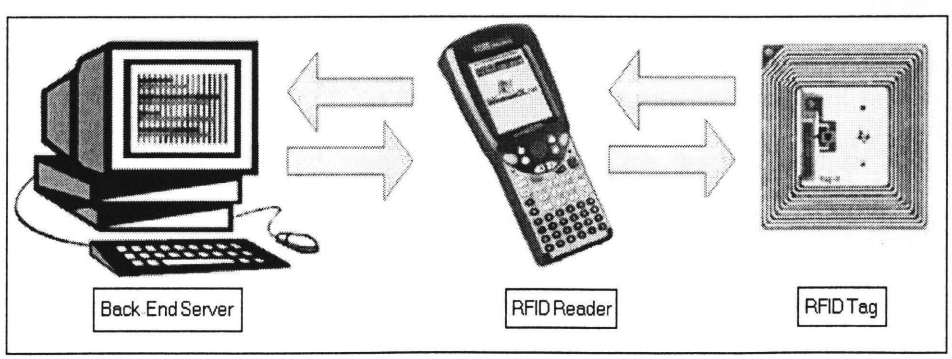

Figure 1.1: RFID Systems

In general, the reader queries tags by broadcasting a radio frequency signal. A tag responds to the reader by transmitting back its identification information. The reader forwards the tag response to a back-end server. The server has a database of tags and can retrieve detailed information regarding the tag (or the item attached to the tag).

1.2 RFID Applications

As of today, RFID systems have been applied to a wide range of problems including supply chain management to replacing barcode, access control in restricted areas such as laboratories and airports, asset tracking, automatic payment and product authentication to detect counterfeits. Some of the widely used applications of RFID are described in this section.

Supply Chain Management

Stores and libraries have used electronic article surveillance (EAS), a 1-bit form of RFID for theft control since the 1960s. EAS tags indicate whether an item has been bought or properly checked out; a clerk will usually deactivate the tag at checkout. By extension, RFID tags are basically EAS tags augmented with additional data storage and processing. Low-cost RFID tags promise to expedite supply chain processes, from moving goods through loading docks to managing the terabytes of data collected from these goods. The US Department of Defence and various retailers (such as Wal-Mart) are already conducting RFID trials at the pallet, case, and item levels [2].

Automatic Payment

Automatic payment is another popular RFID application. Various industry sectors have conducted trials of RFID-enhanced cashless payment technology, ranging from RFID-augmented credit cards and public transportation tickets to RFID-like near field communication in consumer

devices. Electronic toll collection using E-Z Pass is another widespread application. The active E-Z Pass tag attaches to a car's windshield or front license plate; as the car drives on a toll road, the tag sends account information to the toll collection equipment in the lanes or overhead. Although customers consider the E-Z Pass hip and modern, the technology was patented in 1977 (as shown in Figure 1.2) and has been deployed since the 1980s [2].

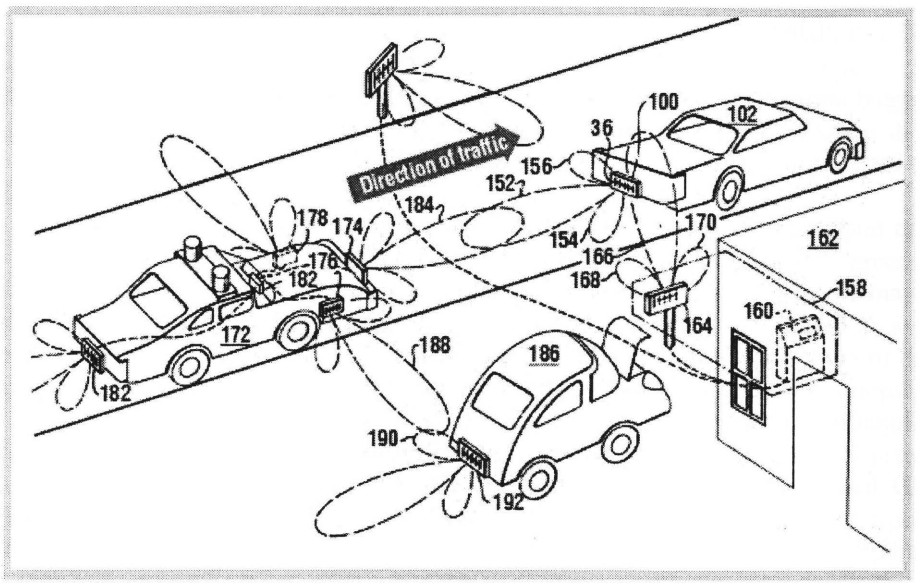

Figure 1.2: Car Tracking with RFID-Tagged License Plates (Courtesy Fred Sterzer, USPatent 4001822)

Access Control

Contactless access control with RFID is popular for securing physical locations, such as office buildings, hospitals and university campuses. Charles Walton first invented an RFID-based access control system in 1973 [2]. It involved an electronic lock that opened with an RFID key card. The passively powered key card, which Schlage sold for US$1.25, was a 36-square-inch circuit board loaded with chips and analog components [2]. Today, RFID-based access cards are the size of a credit card and assist with policing border access. The US Department of Homeland

Security (DHS) and the International Civil Aviation Organization (ICAO) also plan to use passive RFID to police airport access [2]. By 2015, the ICAO wants to replace all passports (approximately 1 billion) with digital passports that store encrypted biometric data on an RFID chip [2]. The DHS also wants to use RFID to record who is entering or leaving the US across land routes [2].

Animal Tracking

RFID-tagged animals are already common. Applications vary from identifying runaway pets to tracking cattle from pastures to the grocer's freezer. Cows and chips first met in the 1970s in American microwave-based systems and European inductively powered systems [2]. Since then, various parties have used RFID-based animal tracking to monitor cows, pigs, cats, dogs, and even fish to control outbreaks of animal diseases such as avian influenza or bovine spongiform encephalopathy ("mad cow disease").

RFID has also been used to track people. Manufacturers have created wearable RFID wristbands, backpacks, and clothing to track prisoners, schoolchildren, and even the elderly. Applied Digital created an injectable RFID tag called the Verichip [2]. This subdermal RFID chip stores personal data that can be read at venues as varied as nightclubs and hospitals.

Other Applications

RFID tagging lets physical objects be represented in cyberspace and entered into databases. Candidates include clothes (to be queried by smart washing machines), packaged foods (to be queried by smart refrigerators), medicine bottles (to be queried by smart medicine cabinets), rental cars, airline baggage, library books, banknotes, driver's licenses, employee badges, and even surgical patients (to avoid mix-ups) [2].

1.3 Motivation, Objective and Project Organization

The objective of this project is to investigate an RFID tag with a secure authentication protocol for communication between the tag and its reader in a way that not only it considers the privacy and security issues but also adds flexibility to the tag. It is achieved in such a way that the cost and time for redesigning and reusing of the tag can be minimized. In most applications, RFID tag and reader hardware and software must be specifically designed for each particular application. RFID system may be physically modified or redesigned every time either the specification for the current application is adjusted or as new applications are introduced. This keeps the overall design time long and the system costs high.

The project explores the RFID system using the Altera's FPGA and Nios II processor that provides a huge advantage in this type of investigation. The tag is implemented as system on programmable chip (SoPC) on the DE2 board [3]. Nios II embedded processor is deployed as a controller. A secure authentication algorithm based on randomized access control is presented in which messages are changed randomly between the tag and its reader, so that the responses are different each time the authentication is in process.

The organization of this thesis report is as the following: Chapter 2 describes the essentials of RFID systems including tag and reader classifications, privacy and security issues, and RFID system performance. Chapter 3 presents a survey on recently proposed RFID systems and existing methods for solving privacy and security issues. Chapter 4 describes the implementation of the proposed FPGA-based RFID tag incorporating a secure randomized access authentication protocol. Chapter 5 presents the tag and reader implementation results. Finally, Chapter 6 provides the conclusion and future work of this study.

Chapter 2

RFID System Essentials

2.1 RFID Tags

An RFID tag is a small microchip with an RF antenna that can be attached to various objects. The microchip is capable of storing elementary information, some processing and radio communication. RFID tags generally come in three types, active, semi-passive, and passive.

Active and Semi-Passive Tags

Active tags require an internal power source (usually a battery) to power the tag for receiving queries and transmitting responses. The power supply also powers the tag's controller, which may be an ASIC or an embedded microprocessor. There are two such types: semi-passive tags, whose batteries power their circuitry when they are interrogated and active tags, whose batteries power their transmissions. Active tags can initiate communication, and have reading ranges of 100 meters or more. [2]

Passive Tags

Passive tags have no on-board power source. As a result, these tags not only receive information from a query, they also receive energy. This energy is used to power the tag to determine and send a response to the query. While passive tags are generally cheaper than active tags, they have two major disadvantages. Firstly, the range of passive tags is significantly lower than active tags, and secondly, the complexity of response is significantly reduced over active tags due to the limited energy budget. However, active tags in addition to being more costly than passive tags

also require maintenance, such as the change of the battery. Table 2.1 summarizes frequency characteristics of RFID systems.

Table 2.1: Frequency Characteristics of RFID Systems

Frequency	Range	Reading Range	Tag Type
Low Frequency (LF)	124 kHz - 135 kHz	Up to half a meter	Passive
High Frequency (HF)	13.56 MHz	Up to a meter or more	Passive
Ultra High Frequency (UHF)	860 MHz – 2.45 GHz	Up to tens of meters	Passive or Active

2.2 RFID Tag Classes

Depending upon the capabilities of the RFID tags, they are divided in different classes [5,6], as described below and summarized in Table 2.2.

Table 2.2: Tag Classes

Class	Known as	Memory	Power Source	Application
0	EAS	None	Passive	Anti Theft
1	EPC	Read-Only	Any	Identification
2	EPC	Read-Write	Any	Data Logging
3	Sensor Tags	Read-Write	Semi-Passive Active	Sensors
4	Smart Dust	Read-Write	Active	Ad-Hoc Networking

Class 0 - Read Only - Factory Programmed

Class 0 tags are the simplest type of tags, where the data is usually a simple ID number, which is written once into the tag during manufacture. The memory is then disabled from any further updates. Class 0 tags are also used to define a category of tags called EAS (Electronic Article Surveillance) or anti-theft devices, which have no ID, and only announce their presence when

passing through an antenna field. They are frequently found in library books or compact discs [5,6].

Class 1 - Write Once Read Many (WORM) - Factory or User Programmed

Class 1 tags contain a unique identifying data stored in a write-once read-many (WORM) memory. Data can either be written by the tag manufacturer or by the user, one time. They are used as barcode replacement. Class 1 tags include EPC (Electronic Product Codes) tags that enable high visibility of products in supply chain [5,6].

Class 2 - Read Write

Class 2 tags are the most flexible type of tags, where users have access to read and write data into the tags memory. Class 2 tags have read-write memory, which allows them to act as logging devices. Class 2 tags may be recycled and used to identify many different items throughout their lifetime [5,6].

Class 3 - Read Write - with on board sensors

Class 3 tags contain on-board environmental sensors and may record parameters like temperature, pressure, and motion; which can be recorded by writing into the tags memory. Since sensor readings must be taken in the absence of a reader, class 3 tags are necessarily semi-passive or active [5,6].

Class 4 - Read Write - with integrated transmitters

Class 4 tags may establish ad-hoc wireless networks with other tags. Since they can initiate communication, class 4 tags are necessarily active [5,6].

2.3 RFID Readers

An RFID reader is a device that can recognise the presence of RFID tags and read or write the information from or to them. RFID readers are becoming more and more advanced with more efficient anti-collision procedures, wider frequency bandwidth and greater data filtering capabilities that allow fast and effective integration of RFID readers into overall information systems. RFID readers consist of three main parts that allow them to function in RF and digital systems. The main three components are control section, high frequency interface, and antenna, as shown in Figure 2.1.

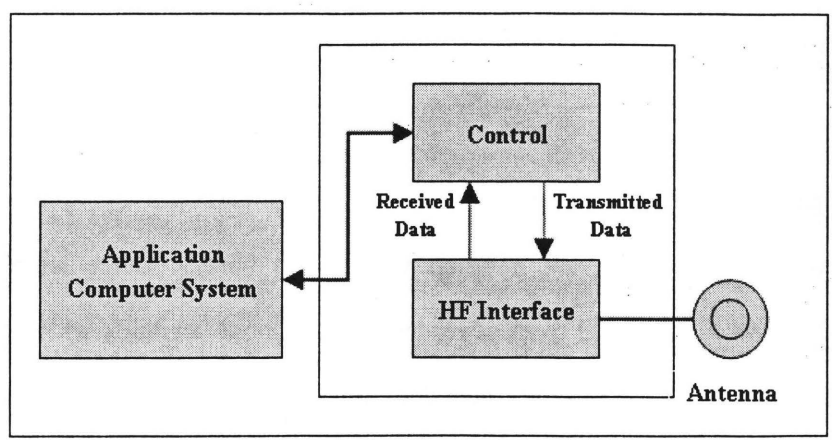

Figure 2.1: Block Diagram of a Typical RFID Reader

RFID readers can be classified on various bases such as power supply, communication interface, mobility, and frequency response [7]. According to these categories, different types of RFID readers are described in this section and summarized in Table 2.3.

Readers Using the Standard Power Network

Readers using the power network generally have a power cord connected to an appropriate external electrical source. RFID readers need DC voltage supply for operation of their electronic

circuits and components. This means that an appropriate AC/DC adapter is used for the power supply of such readers. Most readers that use this type of power supply are fixed stationary readers and their operating power supply comply from 5V to 12V, but there are examples of readers that operate at voltage levels of 24V [7].

Table 2.3: RFID Readers Classification

Power Supply		Communication Interface	
Readers Supplied from the Power Network	Battery Powered RFID Readers	Serial RFID Readers	Network RFID Readers
Mobility		Frequency Response	
Stationary RFID Readers	Hand-held RFID Readers	Unique Frequency Response Based Readers	Non-Unique Frequency Response Based Readers

Battery Operated RFID Readers

Battery powered (BP) readers are powered by using a battery source attached to their motherboard or packaging. These types of readers provide more flexibility when implementing an RFID system due to the fact that the reader does not need external power supply and thus does not depend on the location of power outlets or the use of power cables. Most BP readers are hand-held but there are stationary readers that are battery assisted as well. BP readers use from 5V to 12V batteries for their power supply [7]. The Alien ALR-2850 is a high-performance BP reader designed for range, sensitivity and sophisticated data handling that works with 12V battery [8].

Serial RFID Readers

Serial readers use a serial communication link for communication with the host computer or software application. The reader is physically connected to a host computer using the RS-232,

RS-485, I2C or USB serial connection [7]. Serial readers have the advantage of being more reliable for data transmission than network readers. The disadvantage of serial readers is that there are a limited number of serial ports at the host side and it might be needed to have a large number of host computers to connect all the serial readers. Moreover, the serial RS-232/485 cable is limited in length and the data transfer rate is lower than the network data transmission rate [7].

Network RFID Readers

Network readers are connected to the host computer via a wired or wireless network. These types of readers behave as a standard network device and do not require particular knowledge of the hardware and system configuration.

Today's RFID readers support multiple network protocols such as Ethernet, TCP/IP, UDP/IP, HTTP, LAN, WLAN and others. This allows easier tracking and maintenance of the readers installed in the system. The number of readers or their placement in the system environment isn't determined or restricted by the wired connection as it is the case with serial connections. The data transmission rate of network readers is far greater than with serial readers (up to 10Mbps for Ethernet) and thus data is collected at a higher rate [7,9].

The disadvantage of network readers is that the communication link is not as reliable as serial communication. When the communication link goes down, the back end cannot be accessed. As a result, the RFID system might come to a complete standstill. RFID readers have internal memory to store the received data from the tags, so that short time network failures may be compensated. The Samsys MP9320 v2.8 reader [9] provides multiple communication options including RS-232, RS-485, and 10/100 Mbps Ethernet connectivity.

Stationary RFID Readers

Stationary RFID readers are also known as fixed readers. This term comes from the reader's ability to be mounted on walls, portals, doors or other objects where they can perform effective tag readings and are not meant to be moved or carried. Fixed RFID readers are mainly used for wireless data capture in supply chain management, asset and product control. Today's fixed RFID readers are used for personnel identification and authentication for restricted access areas installed and mounted on portals and doors as well [7].

Most stationary readers support multiple protocols for tag and reader communication and can operate in standalone and in networking mode. A new trend involving the design of stationary readers places multiple antenna connections for connecting more than one antenna to the reader, allowing the user to achieve greater and diverse radiation patterns of the reader's interrogation area. They use power supply from 12V to 24V, weigh from 1.5 kg to 5kg and can achieve reading ranges up to 300m [7,10]. The 303 MHz Mantis II reader is an advanced stationary RFID reader in RF Code's Mantis product family [10]. Mantis II readers track battery-powered RFID tags to quickly locate and identify assets or people in a defined area.

Hand-held RFID Readers

Hand-held RFID reader is a mobile reader that can be carried and operated by a user as a hand-held unit. Hand-held readers have built in antennas and usually do not have connectors for additional antennas. They are battery powered and light weight (from 82g to 700g) and can achieve shorter reading ranges than fixed readers (up to 100 meters) [7,11].

Hand-held readers are used in tracking live stock, locating items in stores and in stock, etc. They communicate with the host computer using wireless communication protocols and contain memory blocks for saving data and then after the user has finished data capturing it enables data

transfer from the reader to a database via wired communication. Most hand-held readers have the ability to call out to a specific tag and even allow location of a tag for the location of the hand-held RFID reader. Hand-held RFID readers are integrated with bar code scanners so users can perform both tag and bar code identification with one device, enabling flexible and multiple applications [7]. The i-CARD CF is the world's first RFID reader in compact flash format that can attain ranges of up to 100 meters (300 feet) [11].

Unique Frequency Response Based Readers

Unique frequency response based readers operate at a defined frequency range and use this frequency for both data transmission and reception. The reader transmits AC power and the reader's command via its antennas to the tags in its reader zone. This is part of a transceiver unit that is responsible for sending the reader's signal to the surrounding environment and receiving the response back from the tags.

The receiver is the second part of the transceiver module and is responsible for receiving the analog signals from the antenna. It then sends the signals to the reader microprocessor, where it is converted to a digital signal. The microprocessor thus decodes the received analog signal and performs data processing. It also codes and modulates the reader carrier signal when it wants to send out a message to one particular tag or toward all the tags in the interrogation area. The vast majority of RFID readers that can be found on the market today are unique frequency response based readers [7].

Non-Unique Frequency Response Based Readers

Non-unique frequency response based readers operate using one frequency for sending a command or just provide a carrier signal at a certain frequency and listen for an integer multiple

of its carrier frequency, generally in the form of a 2nd harmonic, or a frequency divided signal as the tag's response [12]. Two RF frequencies used for communication by the reader to the RFID system allows fast and reliable full-duplex communication but needs a more complex RF front end for both the reader and tag module. Some RFID systems are designed in such a way that multiple frequency use is enabled by using multiple antennas operating at different but predefined frequencies.

2.4 Privacy and Security in RFID

RFID technology has become an important part of everyday life and incoming ubiquitous environment. However, the communication between tag and reader in RFID system has been conducted by wireless communication at radio frequency and the information on identification could be eavesdropped by a third party maliciously. Such eavesdropped information could be used to impair the privacy of its users.

To provide widespread adoption of RFID, security mechanisms is therefore of utmost important. As a result, many studies have been performed to provide robust wireless communication between tag and reader, addressing privacy and security issues. Generally, the privacy and security concerns and threats in an RFID system can be classified as given below.

2.4.1 Privacy

One of the main concerns for RFID systems is privacy. RFID systems use a shared radio medium, which allows eavesdropping. Unprotected communications between tags and readers over a wireless channel can disclose information about the tags and their positions. In an RFID system, there are two major privacy issues: tag information and location privacies.

Tag Information Privacy

In a typical RFID system, when an RFID reader queries RFID tags, they respond by sending their identifier to the reader; the reader can then request further details by sending the identifiers to a server. If unauthorised readers can also get a tag identifier, then they may be able to determine the additional information related to the tag. For example, if the information associated with a tag attached to a passport, ID-card or medical record can be obtained by any reader, then the damage would be very serious [1]. To protect against such information leakage, RFID based transactions need to be controlled so that only authorised readers are able to access the information associated with a tag.

Tag Location Privacy

If the responses of a tag are linkable to each other or distinguishable from those of other tags, then the location of a tag could be tracked by multiple collaborating tag readers. For example, if the response of a tag to a reader query is a static ID code, then the movements of the tag can be monitored, and the social interactions of an individual carrying a tag may be available to third parties without him or her knowing. If messages from tags are anonymous, then the tag tracking problem can be avoided [1].

2.4.2 Security

The other important concern for RFID systems is security of the RFID protocols. There are many attacks that threat the communication between the RFID reader and tag. Security threats to RFID protocols can be classified as tag impersonation, server impersonation, denial of service attack, replay attack, forward and backward traceability. These threats are described in the following.

Tag Impersonation

An eavesdropper could impersonate a target tag without knowing the tag's internal secrets. It could communicate with readers instead of the tag and be authenticated as the tag [1].

Server Impersonation

In this category of impersonation, an adversary with knowledge of the internal state of a tag is able to impersonate the valid server to the tag. One reason that this is a genuine threat is because of the following attack. If it is possible to impersonate a server to a tag, an adversary could request a target tag to update its shared secrets. The tag and the real server would then be desynchronised, and incapable of successful communications. This threat can also be considered as a reader impersonation [1].

Denial of Service Attack

An adversary disturbs the interactions between readers and tags by intercepting or blocking messages transmitted. Such an attack could cause a server and a tag to lose synchronisation. For example, the server might update the shared data while the tag does not; in such a case they would no longer be able to authenticate each other [1].

Replay Attack

In such an attack, an attacker reuses communications from previous sessions to perform a successful authentication between a tag and a server [1].

Forward Traceability

This can similarly be defined as where knowledge of a tag's internal state at time t can help to identify tag interactions that occur at a time $t' > t$. The only way of maintaining future security

once the current tag secrets have been revealed is to detect key compromise as soon as possible, and to replace the exposed key to protect future transactions [1,13].

Backward Traceability

This occurs if, given all the internal state of a target tag at time t, the attacker is able to identify target tag interactions that occurred at a time $t' < t$. That is, knowledge of a tag's current internal state could help identify the tag's past interactions [13].

2.5 RFID System Performance

Authentication protocols for RFID systems should not only be designed to address the privacy and security threats, but should also take into account the limited capabilities of RFID tags. For example, the use of extensive cryptography-based authentication or high-quality random numbers on the tag-side may not be possible for low cost tags. Extensive cryptographic operations can be shifted to the reader-side [1]. However, this requires the tag to either store large keys or frequently communicate with the reader over a secure out-of-band channel to obtain authorization information. The former option is impractical due to limited tag-side storage; the latter one decreases the utility of an RFID system due to time and cost saving identification technology [14]. The main concerns for limited capability of RFID tags can be classified and summarized as following.

Capacity - The volume of data stored in a tag should be minimised because of the limited size of tag memory [1].

Computation - Tag side computations should also be minimised because of the very limited power available to an RFID tag [1].

Scalability - The server in an RFID system should be able to handle growing amounts of work in a large tag population. Moreover, the reader should be able to identify multiple tags that share the same radio channel [14].

Communication - The volume of data that each tag can transmit per second is limited by the bandwidth available for RFID tags [15].

Chapter 3

RFID Privacy and Security Solutions

Many research groups have proposed solutions to the privacy and security problems in RFID systems. These solutions can be broadly divided into two categories: hardware based and protocol based [16]. Hardware based solutions emphasize on improving RFID tag hardware to provide additional security primitives like elliptic curve cryptography. Protocol based solutions emphasize on designing better protocols using mostly lightweight primitives known to be implementable on RFID tags.

The proposed RFID authentication solution of this project falls under the protocol based category, which will be completely described in chapter 4. In this chapter, a survey on prior work done in both categories with different technical approaches is presented. Through the survey, it is assumed that the connection between the RFID reader and the server is a secure one.

3.1 Killing and Sleeping

Electronic Product Code (EPC) tags address consumer privacy with a simple and draconian provision known as tag killing. When an EPC tag receives a "kill" command from a reader, it renders itself permanently inoperative. To kill a tag, a reader must also transmit a tag-specific PIN (32 bits long in the EPC Class-1 Gen-2 standard) in order to prevent unwanted deactivation of tags. As "dead tags tell no tales," killing is a highly effective privacy measure [4]. Removable RFID tags of Marks and Spencer support a similar approach [17].

Killing or discarding tags enforces consumer privacy effectively, but it eliminates all of the post-purchase benefits of RFID for the consumer. Moreover, in some cases, such as libraries and

rental shops, RFID tags cannot be killed because they must survive over the lifetime of the objects they track. For these reasons, it is imperative to look beyond killing for more balanced approaches to consumer privacy. Rather than killing tags at the point of sale, they can be put to "sleep" [4]. Therefore, some form of access control would be needed for the waking of tags. This access control might take the form of tag specific PINs, much like those used for tag killing. To wake a sleeping tag, a reader could transmit this PIN.

3.2 Re-Labelling

Even if the identifier emitted by an RFID tag has no intrinsic meaning, it can still enable tracking. For this reason, merely encrypting a tag identifier does not solve the problem of privacy. An encrypted identifier is itself just a meta-identifier. It is static and, therefore, subject to tracking like any other serial number. To prevent RFID-tag tracking, it is necessary that tag identifiers be suppressed, or they can be changed over time [4].

Sarma et al. proposed the idea of effacing unique identifiers in tags at the point of sale to address the tracking problem, but retaining product-type identifiers (traditional barcode data) for later use [18]. Inoue et al. suggested that consumers be equipped to re-label tags with new identifiers, but that old tag identifiers remain subject to reactivation for later public uses, like recycling [19]. As a remedy for clandestine scanning of library books, Good et al. proposed the idea of re-labelling RFID tags with random identifiers on checkout [20].

The limitations of these approaches are clear. Effacement of unique identifiers neither eliminates the threat of clandestine inventorying, nor does it eliminate the threat of tracking. Use of random identifiers in place of product codes addresses the problem of inventorying, but does not address the problem of tracking. To prevent tracking, identifiers must be refreshed on a more frequent basis [4].

3.3 Hardware Cryptography

Another approach to RFID security focuses on changing the physical hardware of the RFID tag itself [16]. Batina et al. investigated the possibility of building RFID hardware that is capable of performing public key based authentication [21]. Their efforts have centered on using a particular flavour of public key cryptography based on elliptic curve cryptography (ECC). ECC has been suggested as a good replacement for RSA based public key cryptosystems since a 160-bit ECC offers the same level of security as a 1024-bit RSA encryption. While a public key cryptosystem for RFID tags greatly improves RFID privacy and security, it is more costly to implement than the cryptographic hash functions [16]. Other cryptographic primitives such as Advanced Encryption Standard (AES) and Cyclic Redundancy Check (CRC) are also better fit for the specific demands of RFID [22].

3.4 Re-Encryption Approach

Juels and Pappu addressed the privacy implications of RFID-tags embedded in banknotes, with a scheme where banknote tag serial numbers are encrypted with a law enforcement public key [23]. The resulting ciphertexts undergo periodic re-encryption to reduce the linkability of different appearances of a given tag. Because of the severely restricted computing resources of RFID tags, it is proposed that the re-encryption should be performed by readers [24].

In order to prevent wanton re-encryption (e.g., malicious passers-by), it is proposed that banknotes carry optical write-access keys to re-encrypt a ciphertext and a reader must scan this key. From several perspectives, like the need for re-encrypting readers, the system is very cumbersome [4]. Moreover, it has introduced the principle that cryptography can enhance RFID-tag privacy even when tags themselves cannot perform cryptographic operations [4].

Golle et al. described a similar scheme that is more suitable for privacy-protection of RFID tags embedded in consumer goods [25]. They use multiple public keys; thanks to a technique called "universal re-encryption". This is an extension of the El Gamal cryptosystem in which it is possible to re-encrypt a ciphertext without knowing the associated public key. The Golle et al. scheme suffers from the same drawback as that of Juels and Pappu, namely the requirement for an infrastructure of re-encryption devices [24].

3.5 Minimalist Cryptography

While high-powered devices like readers can relabel tags for privacy, tags can alternatively relabel themselves. Juels proposed a "minimalist" system in which every tag contains a small collection of pseudonyms [26]. It rotates these pseudonyms, releasing a different one on each reader query. An authorized reader can store the full pseudonym set for a tag in advance and, therefore, identify the tag consistently. An unauthorized reader (the one without knowledge of the full pseudonym set for a tag) is unable to correlate different appearances of the same tag.

To protect against an adversarial reader harvesting all pseudonyms through rapid-fire interrogation, Juels proposed that tags "throttle" their data emissions, or slow their responses when queried too quickly. As an enhancement to the basic system, valid readers can refresh tag pseudonyms. The minimalist scheme can offer some resistance to corporate espionage, like clandestine scanning of product stocks in retail environments [4].

Hardware implementation of Feldhofer presents a novel minimalist approach of a 128-bit AES implementation [27]. Their approach provides a promising choice for strong authentication in RFID systems and their proposed low-cost AES hardware implementation is used in various proposals as an enabler of cost-efficient RFID cryptography [28].

3.6 Hash Lock Scheme

Hash-based Access Control (HAC), as defined by Weis et al., is a scheme which involves locking a tag using a one-way hash function [29,30]. A locked tag uses the hash of a random key as its *metaID*. When locked, a tag responds to all queries with its *metaID*. The reader forwards this *metaID* to the backend server which then retrieves the real tag ID for the reader. Every tag has a unique *metaID* and will always reply with the same *metaID* value when queried. However, this scheme allows a tag to be tracked because the same *metaID* is used repeatedly [1].

Therefore, the authors proposed the Randomized Access Control (RAC), which employs a random number generator to prevent the above tracking attack. Under this scheme, a tag generates a new response as a hash function of the tag ID and a random number. The reader forwards this reply to a secure database which then searches its database for the secret information that matches the tag reply. However, tag impersonation remains possible because an intercepted response can be replayed. Moreover, it does not provide backward untraceability because the tag ID is fixed [1].

Molnar and Wagner pointed out that the randomized hash lock scheme does not defend against an eavesdropper properly [31]. An adversary can eavesdrop on the communication between readers and tags to learn the tag reply, $(r, ID \oplus f_k(r))$. The adversary then uses this information to impersonate the RFID tag to fool a reader. The authors suggested having both the reader and tag each contribute a random number, r_1 and r_2 respectively. Their approach assumes that the reader knows the tag secret k. After the reader and tag exchange random numbers, the tag replies with $ID \oplus f_k(0, r_1, r_2)$. Since the reader knows k, it can derive $f_k(0, r1, r2)$ and obtain ID. The protocol works without a central database. However, it does not consider the case of a compromised reader. An adversary with a compromised reader will know the tag secret of every

tag the reader has access to. The adversary can then use this information to make duplicate tags to fool other readers [16].

Ohkubo et al. also proposed a hash chain based scheme for privacy-preserving tag identification on the face of active attacks [32]. Their scheme also provides forward security. Tag 'i' is initialized with secret x_i, and h_1 and h_2 are independent one-way hash functions. When query is sent to the tag, tag updates its secret key by applying h_1 and sends $ID_{i,t}=h_2(h_1(x_i))$ to the reader. The reader in this scheme shares secrets with the tags. After receiving $ID_{i,t}$, the reader determines x_i by brute-force search. They proposed a fixed upper bound 'm' on the number of time steps over which tags operate. In their scheme, the reader pre-computes a giant table $T=\{[ID_{i,t},(I,t)]\}$ where $1\leq i\leq n$ and $1\leq t\leq m$ [33]. Their scheme uses a low-cost hash chain mechanism to update tag secret information to provide indistinguishability (i.e. a tag output is indistinguishable from a truly random value and unlinkable to the ID of the tag) and backward untraceability. However, it is subject to replay attacks and hence it permits an adversary to impersonate a tag without knowing the tag secrets [1].

The initial idea of this project, which is described in the next sections, is based on the randomized access control scheme provided by Weis et al. [29], Lee et al. [34] and Jeong et. al. [35]. The two schemes are used and have been expanded to offer more security and privacy in an RFID system.

3.7 Challenge and Response

An alternative method for RFID authentication is based on a "challenge and response" between a reader and a tag. Juels and Weis observed that human authentication protocols can be applied to RFID, due to their weak computational capabilities, like humans [36]. They introduced HB protocol based on the work of Hopper and Blum [37]. In HB Protocol, a reader issues a new

challenge to a tag each time it queries an RFID tag. The tag computes the binary inner product based on the reader's challenge, and returns the answer to the reader. The reader authenticates the tag by verifying the tag response. The HB+ protocol is an improvement over the HB protocol and it employs an additional binding factor from the tag to defend against an active adversary. HB++ is the later work that was improved on this idea [38].

Ranasinghe et al. presented ways to implement challenge-response authentication protocol on RFID tags without using costly cryptographic primitives [39]. These proposals are based on a Physical Unclonable Function (PUF) residing on the tag, which allows for calculation of unique responses using only some hundreds of logical gates. A possible candidate for the PUF can be found from Lee et al. work [40], where the manufacturing variations of each integrated circuit are used to implement a secret key on a tag. The back-end server needs to store a list of challenge-response pairs for each PUF (i.e. for each tag) because, without encryption, a PUF challenge-response pair that is once used, can not be used again since it may have been observed by an adversary. The PUF based security is still an area of active research. Also, Tuyls et al. propose the use of PUFs to increase RFID tags resistance against both physical and communication based cloning attacks and defined an offline authentication protocol [41]. The authors estimated that their anti-clone tag can be built with on the order of 5,000 gates.

3.8 Blocking

Juels et al. proposed a privacy-protecting scheme that they called blocking [24]. Their scheme depends on the incorporation of a modifiable bit called a 'privacy bit' into tags. A '0' privacy bit marks a tag as subject to unrestricted public scanning and a '1' bit marks a tag as private. They refer to the space of identifiers with leading '1' bits as a privacy zone. A blocker tag is a special RFID tag that prevents unwanted scanning of tags mapped into the privacy zone.

To illustrate how blocking might work in practice, a supermarket scenario may be considered. When first created, and at all times prior to purchase – in warehouses, on trucks, and on store shelves – tags have their privacy bits set to '0'. In other words, any reader may scan them. When a consumer purchases an RFID-tagged item, a point-of-sale device flips the privacy bit to a '1'. Therefore, it transfers the tag into the privacy zone. This operation is much like the "kill" function in EPC tags, and may be similarly PIN-protected [4]. Once in the privacy zone, the tag enjoys the protection of the blocker. Supermarket bags might carry embedded blocker tags, to protect items from invasive scanning when shoppers leave the supermarket. When a shopper arrives home, she/he removes items from shopping bags and puts them in the refrigerator. With no blocker tag inside, an RFID-enabled smart refrigerator can freely scan RFID-tagged items. The consumer gets privacy protection from the blocker when it is needed, but can still use RFID tags when desired [4].

3.9 Timestamp Approach

Tsudik introduced a novel technique, YA-TRAP, which employs timestamps in RFID authentication [42]. YA-TRAP provides tracking-resistant tag authentication through monotonically increasing the timestamps on a tag. This is a novel approach for those RFID tags that have no self-contained power source to keep track of time. In YA-TRAP, a reader will send a timestamp of the current time to a tag which then decides whether to return a random reply or an encrypted reply based on the received timestamp and its own internal timestamp. The reader sends this reply back to the backend server to obtain the tag data [42].

Chatmon et al. proposed anonymous RFID authentication protocols based on YA-TRAP that provide anonymity for authenticated tags and address some vulnerabilities of the original design, while increasing the server workload [43].

3.10 Summary

Each year quite a large number of RFID authentication solutions are published in scientific literature. Based on the computational cost and the operations supported on tags, the proposed solutions can be broadly classified into two categories: protocol based and hardware based [16].

In the protocol based solutions, the emphasis is on designing better protocols using mostly lightweight primitives known to be implementable on RFID tags. These protocols should support Pseudo-Random Number Generator (PRNG), one-way hashing function, functions like Cyclic Redundancy Check (CRC), and bitwise operations (like XOR, AND, OR) on tags. The Electronic Product Codes (EPC) tags that enable high visibility of products in supply chain are one of the main applications in this category. EPC tags support PRNG and CRC checksum.

In the hardware based solutions, the emphasis is on improving RFID tag hardware to provide additional security primitives like symmetric encryption, elliptic curve cryptography, or even the public key algorithms. E-passport is one of the main applications of the hardware based solutions.

Chapter 4

RFID Tags with Randomized Access Authentication Protocol

The major areas that drive the commercial deployment of RFID technology are logistics, supply chain management, library item tracking, medical implants, road tolling, building access control, aviation security, and homeland security applications. Each of these RFID systems has customized requirements that currently are defined ad hoc. In addition, multiple, often competing, standards exist (ISO/IEC JTC1, ANSI, EPC, etc.) for RFID hardware, software, and data management. [44]. In most applications, RFID tag and reader hardware and software must be specifically designed for each particular application. Hardware and software must also be physically modified or re-designed every time the specification for the current application is adjusted, as new applications are introduced, or the standards are modified. This keeps the overall design time long and the system costs high.

The main idea of the proposed system is to implement a robust authentication protocol between the RFID tag and the reader, which addresses all the security issues including eavesdropping, tracking, cloning, replay attack and denial of service attack. A system on a programmable chip (SoPC) tag based on Nios II embedded processor is implemented and a secure randomized access authentication protocol is proposed. It is supposed that the server and the reader have sufficient resources to use strong symmetric or asymmetric key algorithms so that the communications between them are secure. The proposed RFID system can be extended to a reconfigurable RFID tag system [44] as a future work in order to keep the overall design

time shorter and the system costs lower. This section describes and analyzes the proposed system in detail.

4.1 What is A Hash Function?

A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties [45].

The basic requirements for a cryptographic hash function are as follows.

- The input can be of any length.
- The output has a fixed length.
- $H(x)$ is relatively easy to compute for any given x.
- $H(x)$ is one-way.
- $H(x)$ is collision-free.

A hash function H is said to be one-way if it is hard to invert, where 'hard to invert' means that given a hash value h, it is computationally infeasible to find some input x such that $H(x) = h$. If, given a message x, it is computationally infeasible to find a message y not equal to x such that $H(x) = H(y)$, then H is said to be a weakly collision-free hash function. A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$ [45].

4.2 Related Work

Lee and Verbauwhede proposed an RFID authentication protocol for secure and low-cost RFID systems [34]. Their protocol SRAC (Semi-Randomized Access Control) is designed using only

hash function as security primitives in tags. In spite of very restricted functionality, SRAC resolves not only security properties, such as the tracking problem, the forward secrecy and the denial of service attack, but also operational properties such as the scalability and the uniqueness of MetaIDs. Moreover, their scheme has significantly reduced the amount of tag transmissions which is the most energy consuming task.They supposed that the communications between the server and the reader are secure. Therefore, it is assumed that the messages arrived to the reader are securely passed to the server. Figure 4.1 illustrates SRAC authentication protocol.

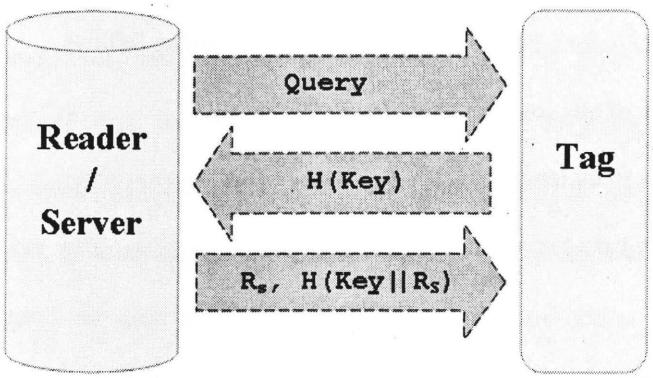

Figure 4.1: SRAC Authentication Protocol

In this scheme, each tag contains its own *key* which is irrelevant to the other tags and *H()* represents one-way cryptographic hash function. SRAC authentication protocol can be described as follows:

Step 1: Reader sends the Query to the tag.

Step 2: Tag sends MetaID=*H(Key)* to the reader/server.

Step 3: Server looks up *Key* using MetaID, generates a random number R_s, and checks whether $H(Key \oplus R_s)$ is unique among the other MetaIDs. If it is not unique, server regenerates R_s until $H(Key \oplus R_s)$ becomes unique.

Server updates *Key* as follows.

If $H(Key_{Curr})$=MetaID

$Key_{Prev} \leftarrow Key_{Curr}$, $Key_{Curr} \leftarrow H(Key_{Curr} \oplus R_s)$

If $H(Key_{Prev})$=MetaID

$Key_{Curr} \leftarrow H(Key_{Prev} \oplus R_s)$

Server sends R_s and $H(Key||R_s)$ to the tag through the reader.

Step 4: Tag checks whether $H(Key||R_s)$ is correct.

If it is correct, tag updates $Key \leftarrow H(Key \oplus R_s)$.

In SRAC protocol, the reason they inserted R_s into a hash function is that the tag needs to check the integrity of R_s. The server authenticates tags by checking whether the received MetaID is on the server's database, and tags authenticate the server by checking $H(Key||R_s)$. In order to be resilient against the denial of service attack, the key update of the server must be more sophisticated than tags. The server keeps two keys, the current key (Key_{Curr}) and the pervious key (Key_{Prev}). SRAC uses only a hash function for security primitives.

In SRAC protocol, the server can search out a tag's Key using MetaID ($H(Key)$). Since the database of the server can be indexed using MetaIDs, the searching is efficient and thus the system is scalable. SRAC resolves the problem of uniqueness of MetaIDs by checking whether an updating MetaID is to be unique in step 3. In this protocol the server only needs to regenerate a random number R_s again until a new MetaID becomes unique. Since the uniqueness is confirmed, they do not need a large size of MetaIDs to evade the conflictions of MetaIDs. Therefore, they can significantly reduce the number of bits used in MetaIDs, which means less energy to transmit and less memory to store a MetaID. On the other hand, for each time of

authentication, the required cryptographic computations in tags are only three hashes, and the amount of transmission of a tag is the size of the hash output. Their scheme can be implemented very efficiently in computation and transmission [34].

SRAC protocol, also, resolved the cloning problem, the forward secrecy and the denial of service attack. The secret information stored on the tag is pertinent to each tag. Even if some tags are compromised, the other tags are irrelevant to the compromised information. Therefore, attacker cannot make any other fake tag except for the compromised tags. The proposed scheme resolves the tracking problem by changing tags' secret information whenever the authentication is successful [34].

In this protocol, the revealed secret information of tags cannot affect the past secrecy. Even if all the communications between a reader and a tag were eavesdropped and recorded, using the current secret information, i.e. *Key*, attackers cannot infer the past secret information. This is because a reader and a tag update their secret information using a hash function each time of the protocols. Therefore, as long as a hash function is not invertible, the past secret information is secure [34].

In SRAC protocol, if the server fails in searching for a MetaID, the server can search out through the previous MetaIDs. Since only one more MetaID for each tag is stored in the server, they can effectively prevent the denial of service attack [34]. Therefore, SRAC can be a good solution for low-cost RFID systems that require good operational and security properties [34].

The authentication protocol presented in this project is based on SRAC protocol, which has been expanded to deploy more than one cryptographic hash function. For each authentication, one of these hash functions is selected randomly. By deploying a randomly changed hash function, the proposed protocol offers more security and privacy for an RFID system.

4.3 Authentication Protocol Proposal

The main idea of the proposed scheme originated from the fact that deploying standard cryptography functions like RSA encryption or AES are not enough for providing security to the RFID system. In fact, applying a robust authentication protocol between the tag and reader can be considered more important than deploying the complex and computationally intensive cryptography functions to encrypt messages transferred between the tag and reader.

Generally, the authentication in an RFID system is done first by authenticating the reader to a tag. Then, a tag is ready to open its information to a reader, and secondly, by authenticating a tag to a reader so that the system prohibits the usage of fake tags. Existing authentication protocols can be divided into two categories: fixed access control in which a tag replies a reader with a fixed message, and randomized access control in which a tag replies to a reader with a pseudo-random message which varies each time of the responses.

The fixed access control is the simplest type so that tags can be implemented in a cheap price. However, this kind of protocols leads to the tracking problem. In such a system, even though attackers cannot figure out the real ID, the constant responses of tags cause the tracking problem. A solution to prevent the tracking problem is the randomized access control in which messages are changed randomly so that the responses are different each time the authentication is in progress. The proposed authentication protocol is based on this solution.

It is supposed that the server and the reader have secure communication in between for this randomized access protocol. Therefore, only communications between the reader and tags are considered and it is assumed that the messages arrived to the reader are securely passed to the server. In the following protocol, which is illustrated in Figure 4.2, the reader and the server are

not separated. The definition of variables and operations used through the protocol description are briefly introduced in Table 4.1.

Table 4.1: Notations

Notation	Definition
ID_k	The Identifier of Tag k
Hnum	Hash Function Identifier Assigned to a Tag
Hash()	One-way Hash Function
R_n	Random Number
RH_{num}	Random Hash Function Identifier Assigned to a Tag
$\oplus$	Exclusive-Or Operation
$M_1 \| \| M_2$	Concatenation of M1 and M2

In the proposed system, tag *k* contains its own identifier *(ID_k)* which is irrelevant to other tags. For encryption of the tag identifier, the system deploys a list of one-way hash functions. A one-way hash function is a cryptography function, besides the function being difficult to invert, whose output should not reveal any substantial information on its input. In fact, use of secure one-way hash functions should be considered for maximum security.

In this system, for applying more security, more than one-way hash function is deployed for encryption and each tag contains its own hash function identifier *(Hnum)*. This identifier determines which hash function should be used to encrypt the tag identifier while this identifier is also irrelevant to other tags. These two identifiers are tag's secret information, which will be randomly changed whenever the authentication is successful.

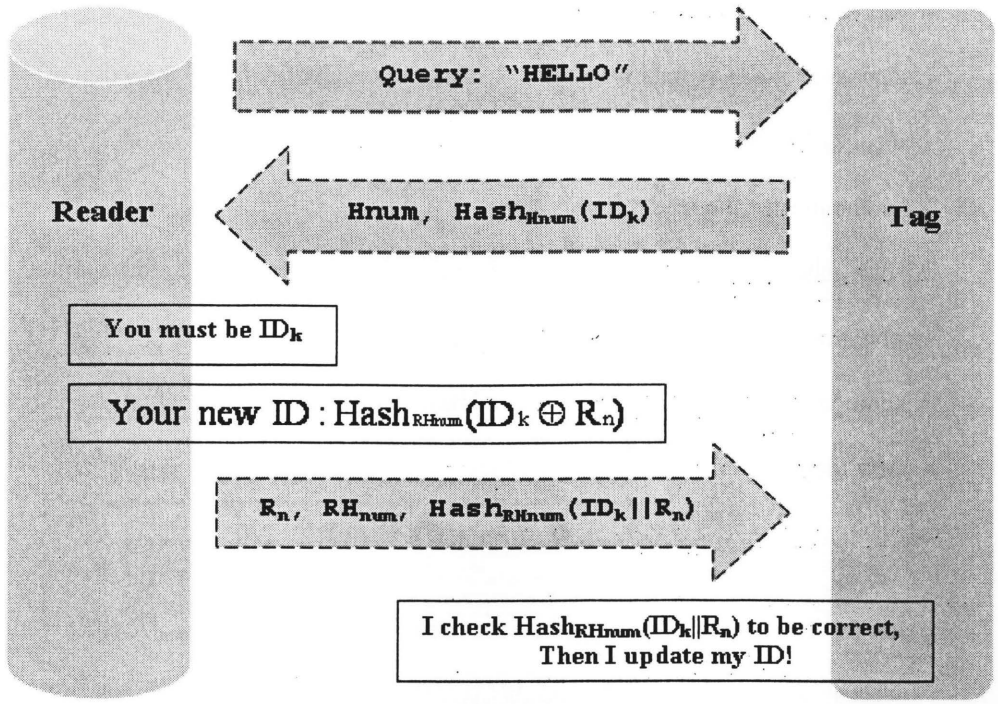

Figure 4.2: Proposed Randomized Access RFID Authentication Protocol

The proposed authentication protocol can be described as following:

Step 1: Reader sends the Query to the tag.

Step 2: After receiving the Query, tag calculates the hashed value of tag identifier $Hash_{Hnum}(ID_k)$ according to its assigned hash function identifier *Hnum*.

Tag sends the result of $Hash_{Hnum}(ID_k)$ to the reader.

Step 3: Reader authenticates the tag in this step:

Reader looks up tag identifier in its database using the received hashed value. If such an identifier does exist, reader generates a random number R_n and a random hash function identifier RH_{num}, and updates tag identifier with the new value:

$$Hash_{RHnum}(ID_k \oplus Rn)$$

Reader saves the current tag identifier as previous identifier.

Reader sends random number, random hash function identifier, and the value of $Hash_{RHnum}(ID_k||R_n)$ to the tag.

Step 4: Tag authenticates the reader in this step:

Tag calculates the value of $Hash_{RHnum}(ID_k||R_n)$ using the received random number, received random hash function identifier and its own identifier ID_k.

Tag compares the result with the received hashed value; if they are equal; tag calculates its new identifier and updates it:

$$Hash_{RHnum}(ID_k \oplus Rn)$$

The reason for inserting R_n and RH_{num} into the hash function in $Hash_{RHnum}(ID_k||R_n)$ is that the tag needs to check the integrity of R_n and RH_{num}. The reader authenticates tags by checking whether the received $Hash_{Hnum}(ID_k)$ is on the server's database, and tag authenticates the reader by checking $Hash_{RHnum}(ID_k||R_n)$. In order to be resilient against the denial of service attack, the secret information update of the reader must be more sophisticated than tags. The reader keeps current and pervious secret information of each tag. The reason will be discussed in detail on the following sub-section of security properties.

The drawback of this solution is that it is under the replay attack. In this system, the server stores two identifiers for each tag, the current one and the previous one, and if it is matched with either of two, the server will authenticate a tag. Attackers may eavesdrop and reuse the recently used secret information and will succeed to be authenticated. To prevent the replay attack, the proposed protocol is modified to use the challenge and response method for both directions, as illustrated in Figure 4.3.

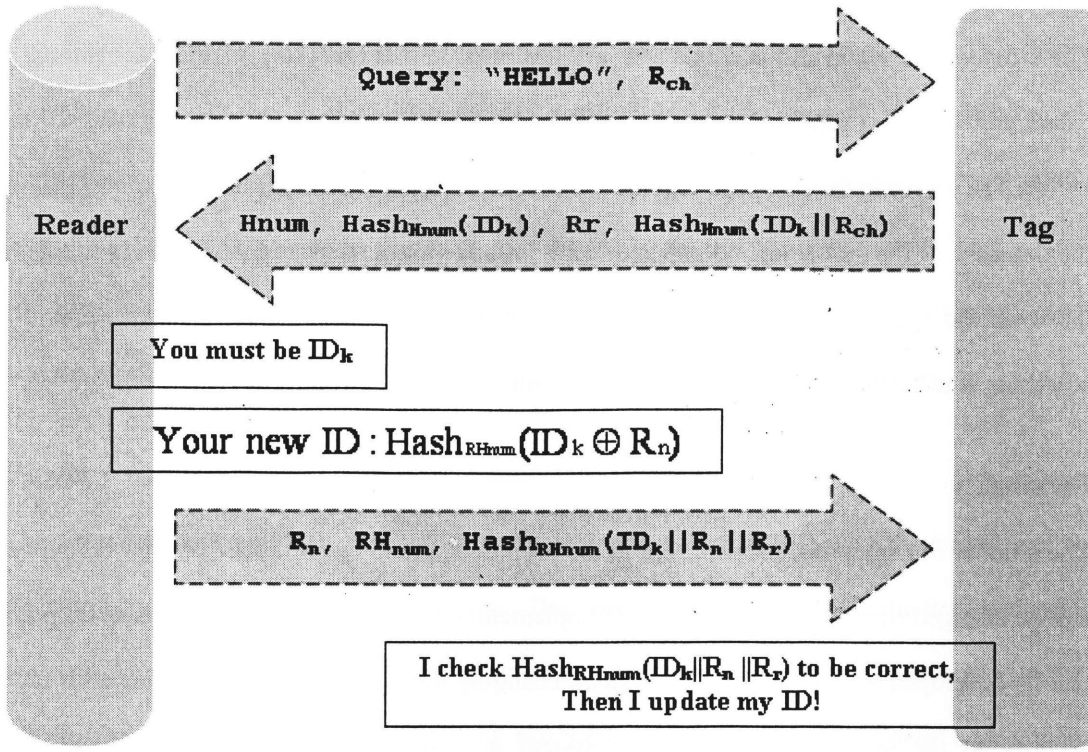

Figure 4.3: Proposed Robust Authentication Protocol

According to the protocol shown in Figure 4.3, the reader authenticates tags by checking whether the received $Hash_{Hnum}(ID_k)$ is on the server's database and checking $Hash_{Hnum}(ID_k||R_{ch})$, and tags authenticate the reader by checking $Hash_{RHnum}(ID_k||R_n||R_r)$.

4.4 The Proposed Protocol - Operational Properties

In an RFID system, in addition to total life cycle cost (one-time and recurring costs), various architectural and operational issues including scalability, computational resources, power consumption, functionality, flexibility, system management and ease of use will drive the competing system designs. In the following, operational properties of the proposed system are described and analyzed.

Scalability

The efficiency of tag identification is an important issue in protocol design. Unlike several hash-based protocols that require exhaustive brute-force searches incur the heavy overhead for the server. The server in the proposed scheme can search out a tag's identifier using *Hash(ID)*. Since the database of the server can be indexed using hashed values of tags identifiers, this mitigates the overhead of tag identification and helps the RFID system scale well. In other words, the searching is efficient and thus the system is scalable.

Flexibility

The number of hash functions deployed in the proposed system can be added or their definitions can be changed over time. Since the tag is implemented as a system on programmable chip and can be developed as a reconfigurable system, changing the hash functions over time adds more security to the system. Meanwhile, the system is also flexible and reusable for different applications. It does not need to physically modify or re-design the proposed tag every time the specification for the current application is adjusted, new applications are introduced, the standards are modified or new standards are developed. This keeps the overall design time short and the system cost low.

Computational Resources

For each authentication, the required cryptographic computations in tags are four hashes, two in step 2 and two in step 4. In order to provide more security to the system, after any successful authentication, the hash function will change. Number of different hash functions that can be added to the system depends on the memory space available. In addition to hash function operations, exclusive-or and concatenation operations are also involved.

Maximum amount of transmission of a tag is as large as the size of the hash output and for the proposed authentication protocol; one may use the hash function producing small output. Therefore, this scheme can be implemented very efficiently in terms of computation and transmission. In the proposed system, lightweight hash functions are deployed for encryption. The minimum number of hash functions is assumed to be three, the minimum length of tag identifier is assumed to be 20 bytes or 160 bits that can include numbers and characters together, and the minimum length of hash function identifier is considered to be one byte.

Power Consumption

The proposed active RFID tag requires an internal power source to power the transceiver for receiving queries and transmitting responses. The power supply also powers the tag's controller, which is Nios II/e embedded microprocessor. Nios II/e provides lower power consumption than off-the-shelf microprocessors by combining many functions onto one chip. Power consumption in a Nios II design is as low as 100 mW. Since the proposed system is implemented on an evaluation FPGA board and transaction between the tag and the reader is simulated using wired serial RS232 standard protocol, measurement of the exact amount of power consumption is not feasible.

4.5 The Proposed Protocol - Security Properties

There are several common attacks on RFID systems. In this section, the resistance of the proposed protocol against these attacks is discussed.

Forward Secrecy

In this protocol, the revealed secret information of tags cannot affect the past secrecy. Even if all the communications between the reader and tag are eavesdropped and recorded, using the current

secret information (i.e. tag identifier) attackers cannot infer the past secret information. This is because a reader and a tag update their secret information using a hash function and a random number each time of the protocol process while the hash function is also being changed for each update. Therefore, the past secret information is secure.

Replay Attack Resistance

Another security property of the proposed RFID tag system is the resistance against the replay attack. Since a reader and a tag both confirm the received message using hash outputs which contain internally generated random numbers, attackers cannot reuse the past messages.

Cloning and Forgery Resistance

After selling a product to the consumer, the retailer inputs secret information into the tag attached to this product. The database maintains the pair of tag identifier and hash function identifier. This pair can be used to examine the forgery. It is extremely difficult for a forger to make a fake tag of the proposed system unless he tampers a tag physically to harvest such a pair. The secrete information stored on each tag including tag identifier and the assigned hash function identifier is pertinent to each tag. Even if some tags are compromised, the other tags are irrelevant to the compromised information. Therefore, attacker cannot make any other fake tag except for the compromised tags.

Eavesdropping Resistance

The secret information of a tag is changed after successful authentication. Moreover, the transmitted values are always hashed, XOR-ed, or concatenates with a different random value every time. Therefore, the messages are indistinguishable from other random values and

meaningless for the adversary. Hence, the probability of breaking secret information is very low and it is difficult for the adversary to extract any useful information.

Tracking Protection

Tracking problem can occur when responses of a tag are constant. The proposed scheme resolves this problem by changing the tag's secret information, which is used as identification data, whenever the authentication is successful.

Denial of Service Attack Resistance

If the server updates secret information in the same way as tags, the protocol is under the denial of service attack. Suppose the server keeps only the current secret information per tag. Then, the attack is possible as following. An attacker generates a jamming signal at step 3 so that the tag cannot receive the message from the reader and does not update its secret information while the server updates the tag's secret information. After this attack, the secret information will be inconsistent between the reader and the tag. Therefore, the later protocols will fail due to lack of synchronization. To resolve this problem, the server also needs to store the previous secret information for each tag. If the server fails in searching for a $Hash(ID_k)$, the server can search out through the previous values. Since only one more hash value for each tag is stored in the server, it can effectively prevent the denial of service attack.

4.6 Comparison

Table 4.2 represents the comparison of some important operational and security properties among the proposed RFID system and some existing systems [15,29,34,45]. From the analysis in section 4.3 and 4.4, it can be concluded that the proposed protocol has a considerably high operational properties and safety capabilities to withstand various attacks on an RFID system.

Table 4.2: Comparison of Operational and Security Features

Properties	Schemes				
	Weis[29]	Ohkubo[15]	Lee[34]	Henrici[46]	Proposed System
Scalability	Scalable	Un-Scalable	Scalable	Scalable	Scalable
Flexibility	Un-Flexible	Un-Flexible	Un-Flexible	Un-Flexible	Flexible
Eavesdropping Resistance	Vulnerable	Strong	Strong	Strong	Strong
Cloning & Forgery Resistance	Strong	Strong	Strong	Strong	Strong
Tracking Protection	Vulnerable	Strong	Strong	Vulnerable	Strong
Replay Attack Resistance	Vulnerable	Vulnerable	Strong	Vulnerable	Strong
Forward Secrecy	Vulnerable	Strong	Strong	Vulnerable	Strong
Denial of Service Attack Resistance	Strong	Vulnerable	Strong	Vulnerable	Strong

Chapter 5

Implementation and Results

An RFID tag is designed and implemented in this project as a system on programmable chip using Quartus II CAD software. The behavior of the RFID tag is developed in an embedded C program which will be compiled and run on Nios II embedded processor. The RFID reader is simulated and implemented as a C++ program which is executed on a PC. The communication between the RFID tag and the reader is simulated and implemented by using RS-232 serial communication. The following sections provide the design and implementation process and the results in detail.

5.1 Software and Tools

This thesis project explores using the Altera's FPGA and embedded Nios II embedded processor integrated in Altera Development and Education (DE2) board. All the tools and programs that are used for completing the project are described here briefly.

DE2 Development and Education Board

The Altera DE2 board, as shown in Figure 5.1, has many features that allow the user to implement a wide range of digital circuits, from simple circuits to various complex multimedia applications. The DE2 board features a state-of-the-art Cyclone II FPGA. All important components on the board are connected to the I/O pins of this chip, allowing the user to control all aspects of the board's operation. The DE2 board includes a sufficient number of switches, LEDs, and 7-segment displays, SRAM, SDRAM, and flash memory chips, as well as a 16 x 2

character LCD display. For experiments like this project requiring a processor and I/O interfaces, it is possible to instantiate Altera's Nios II processor and use interface standards such as RS-232 and JTAG UART.

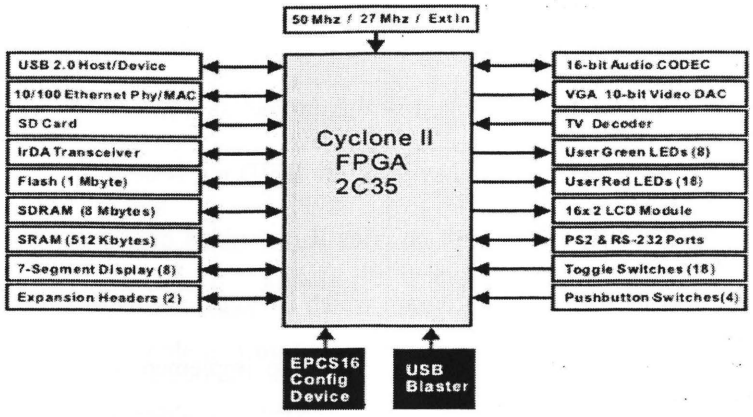

Figure 5.1: Block diagram of the DE2 board

Quartus II

Quartus II is the main CAD software to be used for building a system on chip. It has many tools to prepare VHDL or Verilog structures, compile and download them to the FPGA board. Quartus II is used to build HDL codes while it is also possible to use the **SOPC Builder** tool that specifies the Nios II processor core, memory and other peripherals. The assignment of the pins can be done with the **Assignment Editor**, and downloading the compiled HDL files is done with **Programmer** tool. JTAG programming is used so that the configuration bit stream is downloaded directly to the FPGA.

SOPC Builder

SOPC Builder tool generates the Nios II processor system and adds the desire peripherals. The Nios II processors implement instruction set based on a 32-bit RISC architecture. It has an intuitive user interface that allows developer to select and parameterize components, select

connections between components, and generate a complete system. A major challenge facing embedded developers is selecting a processor that best suits their applications without overspending for performance or sacrificing features. Nios II processors, the ideal embedded solution, allow designers to:

- Choose the exact set of CPUs, peripherals, and interfaces needed for the application.
- Remotely upgrade in the field to stay competitive and address changing requirements.
- Increase performance without changing the board design and by accelerating only the required functions.
- Eliminate the risk of processor and application specific standard product obsolescence.
- Lower overall cost, complexity, and power consumption by combining many functions onto one chip.

With the perfect fit of CPUs, peripherals, memory interfaces, and custom hardware accelerators, Nios II processors offer designers tremendous flexibility, where and when they need it, to meet the unique demands of every new design cycle. Using SOPC Builder, it is possible to choose in which language (Verilog or VHDL) is the code generated. The JTAG UART provides a way to communicate with the processor through the USB-Blaster. Many other components as timers, input/output ports, can be added and personal components and interfaces can be created. SOPC Builder connects multiple components together to create a top-level HDL file called the system module. SOPC Builder also generates **Avalon Switch Fabric** automatically that contains logic to manage the connectivity of all the components of the system.

Nios II IDE

Nios II integrated development environment (IDE) provides a graphical user interface (GUI) to facilitate software development for Altera's Nios II processors. It compiles C language programs

and downloads them into the program memory of Nios II CPU. It is needed to select the SOPC system used, the memory, timer and many different options of compiling, debugging and running the C program. JTAG UART is used to download the object code file to the Nios II program memory.

The **Hardware Abstraction Layer** (HAL) system library provides a hosted C runtime environment based on the ANSI C standard libraries. The HAL provides generic I/O devices, allowing you to write programs that access hardware using the C standard library routines. The HAL minimizes or eliminates the need to access hardware registers directly to control and communicate with peripherals. The Nios II IDE contains a robust software debugger based on the GNU debugger. The Nios II IDE allows you to run or debug the application software either on a target board or the Nios II instruction set simulator. Figure 5.2 illustrates the entire development flow of a system on programmable chip integrating Nios II processor.

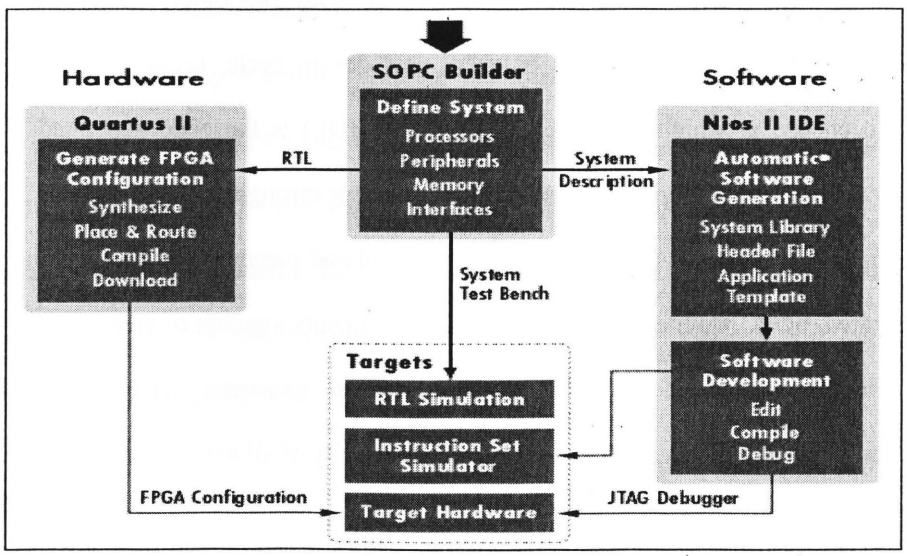

Figure 5.2: Nios II Embedded Processor Development Flow

5.2 RFID System Experimental Setup

The RFID tag system presented in this thesis is illustrated in Figure 5.3. The RFID tag consists of a Nios II/e embedded microprocessor core, 20K on-chip memory and 8M bytes of SDRAM memory. To program the embedded processor, JTAG interface has been added. In order to minimize the power consumption of the tag, the power-aware smart buffer can be added to the tag to manage the activation of the controller.

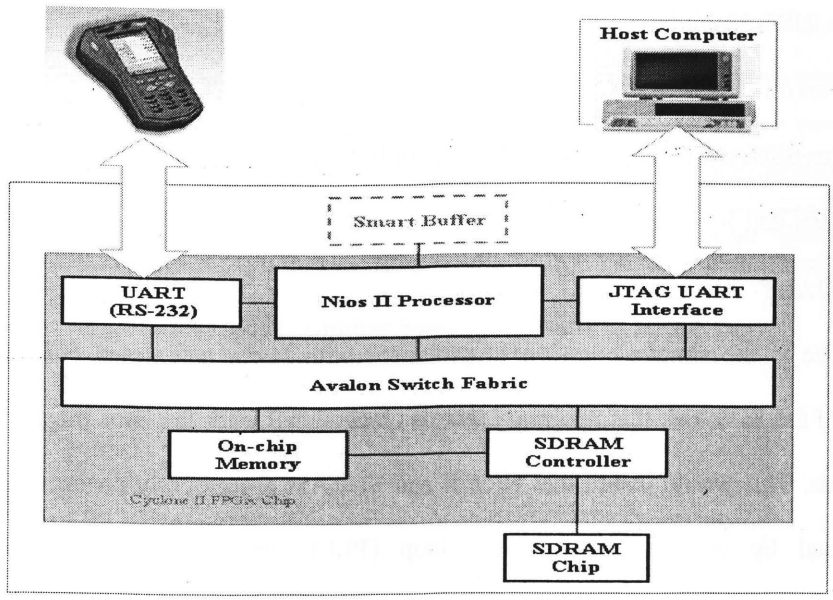

Figure 5.3: The Proposed RFID Tag

Nios II/e processor is the economic type of Nios II CPU which contains about 700 logic elements and works at 50MHz. The Nios II/e core has higher performance but is in the same cost class as a typical 8051 architecture, achieving over 30 DMIPS at 200 MHz of clock. This processor is optimal for cost-sensitive applications, such as those found in the automotive, industrial, and consumer markets. For validation of the proposed RFID system, the reader is

simulated on PC and communication between the tag and reader is done wired through RS-232 serial port. The complete description of the tag implementation is presented later in this report.

Using Quartus II, a new project is created for the RFID tag named 'RFID_Tag.qpf'. The SOPC Builder tool is used to implement the design with Nios II processor core. For this project, the result of generation is a block symbol named 'Nios_System' which is used to integrate the system in the Quartus II project and a file named 'Nios_System.ptf'. The Nios II system in this project, as shown in Figure 5.4, contains the following components:

- 50MHz RISC 32-bit Nios II/e processor core
- 20 Kbytes on-chip memory with 32 bits data width
- 8 Mbytes SDRAM memory controller with 16 bits data width
- UART RS-232 Serial Port
- JTAG UART

Nios II/e is the simplest version of processor with lower gate count (CLB). For proper operation of the SDRAM, it is necessary that its clock signal leads the Nios II system clock by 3 nanoseconds. This would insure that Nios II and SDRAM are properly connected. This can be accomplished by using a phase-locked loop (PLL) circuit. There exists a Quartus II Megafunction, called ALTPLL, which has been used to generate the desired circuit for this project. JTAG UART is added to provide a convenient way to communicate data/code between the Nios II system and the host computer. RS-232 UART is added for serial communication between the RFID tag system and the reader. Serial communication is considered with the baud rate of 19200 bit/second, 8 bits data, 1 stop bit, and no parity. Figure 5.5 illustrates the 'Nios_System' block symbol which is generated by the SOPC Builder system.

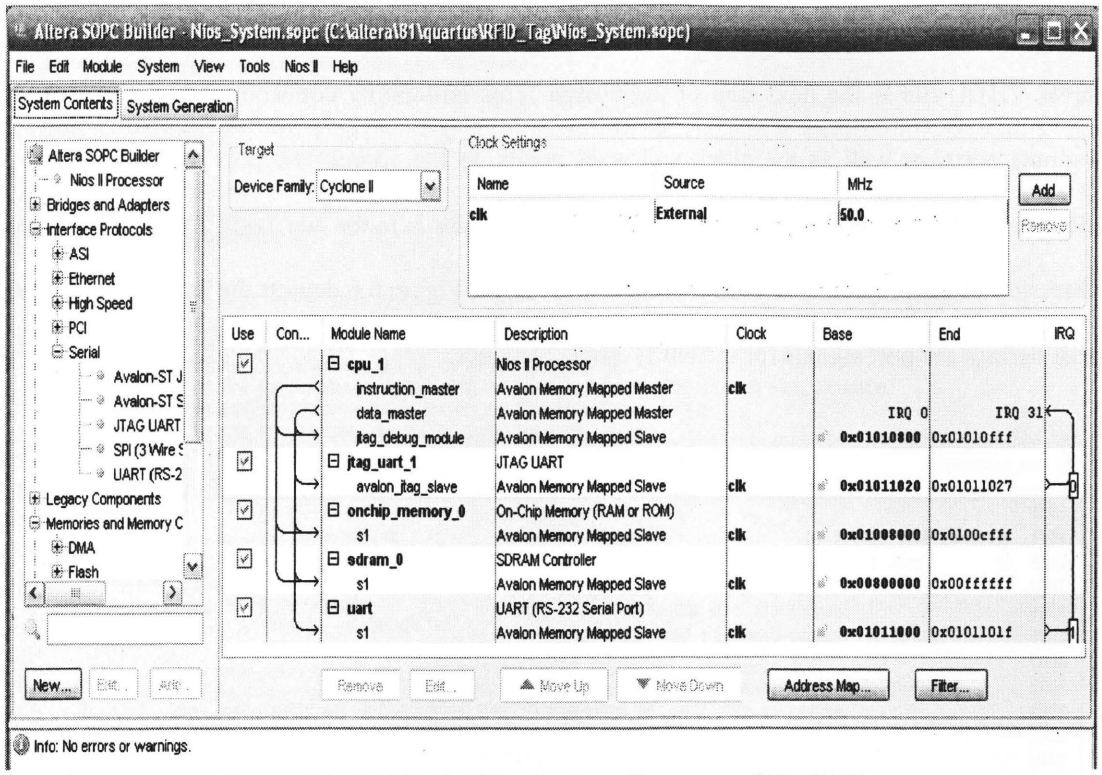

Figure 5.4: SOPC Builder System Contents of RFID Tag

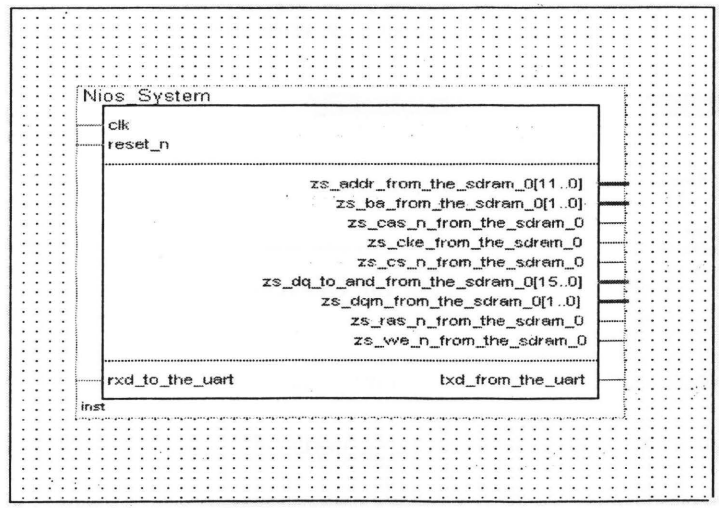

Figure 5.5: Block Symbol of Nios II System for RFID Tag

The instantiation of the generated Nios II module into a Quartus II project and in the top-level VHDL file is the next step of the design. This is done by connecting all the inputs and outputs ports, as well as the clock and reset inputs, to the appropriate pins on the Cyclone II device. The VHDL entity generated by the SOPC Builder is in the file 'Nios_System.vhd' in the directory of the project. The VHDL code is quite large. Figure 5.6 depicts the portion of the code that defines the port signals for the entity 'Nios_System'.

Nios_System.vhd

```
3051
3052  entity Nios_System is
3053          port (
3054                -- 1) global signals:
3055                   signal clk : IN STD_LOGIC;
3056                   signal reset_n : IN STD_LOGIC;
3057
3058                -- the_sdram_0
3059                   signal zs_addr_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
3060                   signal zs_ba_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
3061                   signal zs_cas_n_from_the_sdram_0 : OUT STD_LOGIC;
3062                   signal zs_cke_from_the_sdram_0 : OUT STD_LOGIC;
3063                   signal zs_cs_n_from_the_sdram_0 : OUT STD_LOGIC;
3064                   signal zs_dq_to_and_from_the_sdram_0 : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
3065                   signal zs_dqm_from_the_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
3066                   signal zs_ras_n_from_the_sdram_0 : OUT STD_LOGIC;
3067                   signal zs_we_n_from_the_sdram_0 : OUT STD_LOGIC;
3068
3069                -- the_uart
3070                   signal rxd_to_the_uart : IN STD_LOGIC;
3071                   signal txd_from_the_uart : OUT STD_LOGIC
3072                );
3073  end entity Nios_System;
3074
```

Figure 5.6: A Part of the Generated VHDL Entity

'RFID_Tag.vhd' is a top-level VHDL entity that instantiates the Nios II system and is presented in Figure 5.7. The input and output ports of the entity use the pin names that are specified in the DE2 user manual and allows making the pin assignments by importing them from the file called 'DE2_pin_assignments.csv' which is available at Altera's DE2 web pages.

```
-- Filename:    RFID_Tag.vhd
-- Description: A Nios II system for the RFID tag on DE2 board
-- Department:  Computer and Electrical Engineering of Ryerson University
-- Author:      Leili Borghei
-- Date:        November 2008

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY RFID_Tag  IS
        PORT (  CLOCK_50 : IN STD_LOGIC;
                DRAM_CLK, DRAM_CKE : OUT STD_LOGIC;
                DRAM_ADDR : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
                DRAM_BA_1, DRAM_BA_0 : BUFFER STD_LOGIC;
                DRAM_CS_N, DRAM_CAS_N, DRAM_RAS_N, DRAM_WE_N : OUT STD_LOGIC;
                DRAM_DQ : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
                DRAM_UDQM, DRAM_LDQM : BUFFER STD_LOGIC;
                UART_RXD : IN STD_LOGIC;
                UART_TXD : OUT STD_LOGIC );
END RFID_Tag ;

ARCHITECTURE Structure OF RFID_Tag IS

COMPONENT Nios_System
        PORT (  clk : IN STD_LOGIC;
                reset_n : IN STD_LOGIC;

                zs_addr_from_the_sdram_0: OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
                zs_ba_from_the_sdram_0: BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
                zs_cas_n_from_the_sdram_0: OUT STD_LOGIC;
                zs_cke_from_the_sdram_0: OUT STD_LOGIC;
                zs_cs_n_from_the_sdram_0: OUT STD_LOGIC;
                zs_dq_to_and_from_the_sdram_0: INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
                zs_dqm_from_the_sdram_0: BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
                zs_ras_n_from_the_sdram_0: OUT STD_LOGIC;
                zs_we_n_from_the_sdram_0: OUT STD_LOGIC;

                rxd_to_the_uart: IN STD_LOGIC;
                txd_from_the_uart: OUT STD_LOGIC  );
END COMPONENT;

COMPONENT sdram_pll
        PORT (  inclk0 : IN STD_LOGIC;
                c0 : OUT STD_LOGIC );
END COMPONENT;

SIGNAL BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL DQM : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN
        DRAM_BA_1 <= BA(1);
        DRAM_BA_0 <= BA(0);

        DRAM_UDQM <= DQM(1);
        DRAM_LDQM <= DQM(0);

        -- Instantiate the Nios II system entity generated by the SOPC Builder.
        NiosII: Nios_System PORT MAP (CLOCK_50, '1',
                        DRAM_ADDR, BA, DRAM_CAS_N, DRAM_CKE,
                        DRAM_CS_N, DRAM_DQ, DQM, DRAM_RAS_N, DRAM_WE_N,
                                UART_RXD, UART_TXD);
        -- Instantiate the entity sdram_pll (inclk0, c0).
        neg_3ns: sdram_pll PORT MAP (CLOCK_50, DRAM_CLK);
END Structure;
```

Figure 5.7: Instantiating the Nios II System

5.3 Software Design

As mentioned earlier in this report, Nios II IDE is used to develop and compile the 'RFID_Tag.c' program to be executed on Nios II processor. Target hardware of the system is selected as SOPC Builder system of 'Nios_System.ptf'. After building the C/C++ project, the program is ready to be run on the target hardware by JTAG and USB devices. 'Reader.c++' is the program for simulating the RFID reader in the proposed system. This program has been developed using Microsoft Visual Studio 2005. Complete source codes of 'RFID_Tag.c' and 'Reader.c++' programs are attached to this report in Appendix A.

For the proposed authentication algorithm three general purpose hashing algorithms are being used. The number of the deployed hash functions is flexible and it can be extended to even more hash functions, which depends to the size of the tag memory. The complexity of the hash functions also depends on the application and more complex functions can be used for even more security. Upon starting the authentication process, according to the hash function identifier which is assigned to each tag, one of these hash functions is selected. Since this identifier is randomly changed after a successful authentication, next time another hash function will be used to encrypt the tag's secret information.

The sample hashing algorithms used in the proposed system, as shown in Figure 5.8, are additive, multiplicative, and rotative hashing functions [47]. These functions are not computationally intensive as the objective of this project is to design a secure and robust authentication protocol between the RFID tag and reader. Moreover, deploying a robust authentication protocol is more important than using a complex and computationally intensive cryptography functions that increases the authentication time. As mentioned before for data transaction between the tag and reader, RS-232 serial communication is used in this project.

Opening the serial port, setting serial port properties and timeouts, reading and writing data, cleaning up, and managing serial port communications for PC serial port is done by using Windows Application Programming Interfaces [48].

```
//=============================================================================
/* Hash(char h, char* str, unsigned int len) includes 3 hash functions to generate
   the encrypted form of Tag ID. Any of the 3 hash functions may be selected according
   to the assigned hash ID of the Tag which is randomly change */
//=============================================================================
unsigned int Hash(char h, char* str, unsigned int len)
{
    unsigned int hash = 0;
    unsigned int i = 0;

    // RSHash parameters
    unsigned int b = 378551;
    unsigned int a = 63689;

    // BKDRHash parameters
    unsigned int seed = 131;

    switch (h){
        // ------------ RSHash function -------------
        // ------- Robert Sedgwicks Algorithm -------
        case '1':
            for(i = 0; i < len; str++, i++)
            {
                hash = hash * a + (*str);
                a = a * b;
            }
            break;
        // -----------------------------------------
        // ------------ JSHash function -------------
        // --------- Justin Sobel Algorithm ---------
        case '2':
            hash = 1315423911;
            for(i = 0; i < len; str++, i++)
            {
                hash ^= ((hash << 5) + (*str) + (hash >> 2));
            }
            break;
        // -----------------------------------------
        // ----------- BKDRHash function ------------
        // Brian Kernighan & Dennis Ritchie Algorithm
        case '3':
            for(i = 0; i < len; str++, i++)
            {
                hash = (hash * seed) + (*str);
            }
            break;
        // -----------------------------------------
    }
        return hash;
}
```

Figure 5.8: General Purpose Hashing Algorithms

5.4 Results

The RFID tag is implemented on the Altera's Cyclone II FPGA and other components of DE2 board. The flow summary of compilation is shown in Figure 5.9. As shown, the total logic elements required for this implementation is 2,349 and the total memory bits are 174,080.

In order to evaluate the authentication protocol, secret information (ID) of the tag is assigned within the tag C program which runs on Nios II processor over the DE2 board. A data file is prepared for the reader as its database (ID_file.dat), which includes current and previous secret information and the appropriated hashed values for each tag. The RFID tag system and the authentication protocol are evaluated in various situations and the results are shown in the following snapshots of the RFID reader screen.

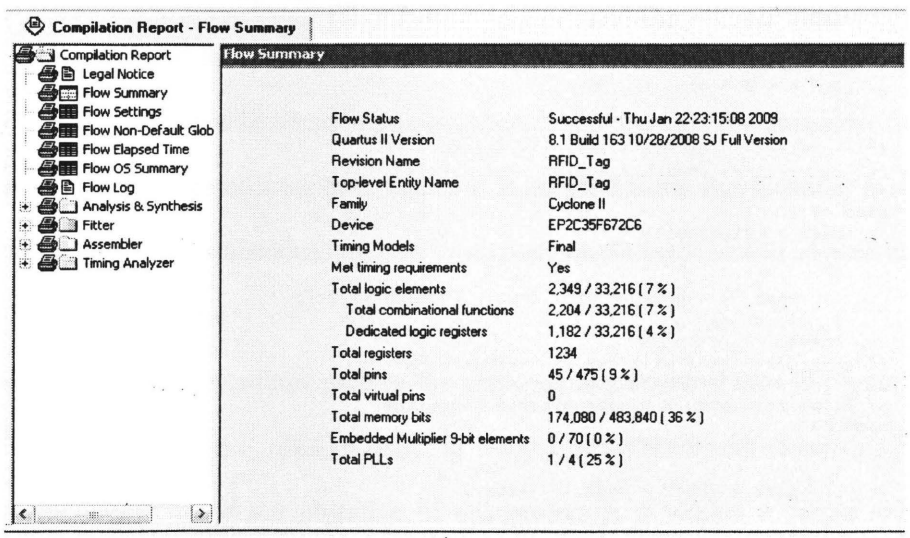

Figure 5.9: Compilation Report

Figure 5.10 shows the reader activities for a successful authentication process. As shown in Figure 5.10, after sending the query, the reader receives two values: hashed value of the tag identifier *($Hash_{Hnum}(ID_k)$=3157342001)*, and its assigned hash function identifier (*Hnum=1)*.

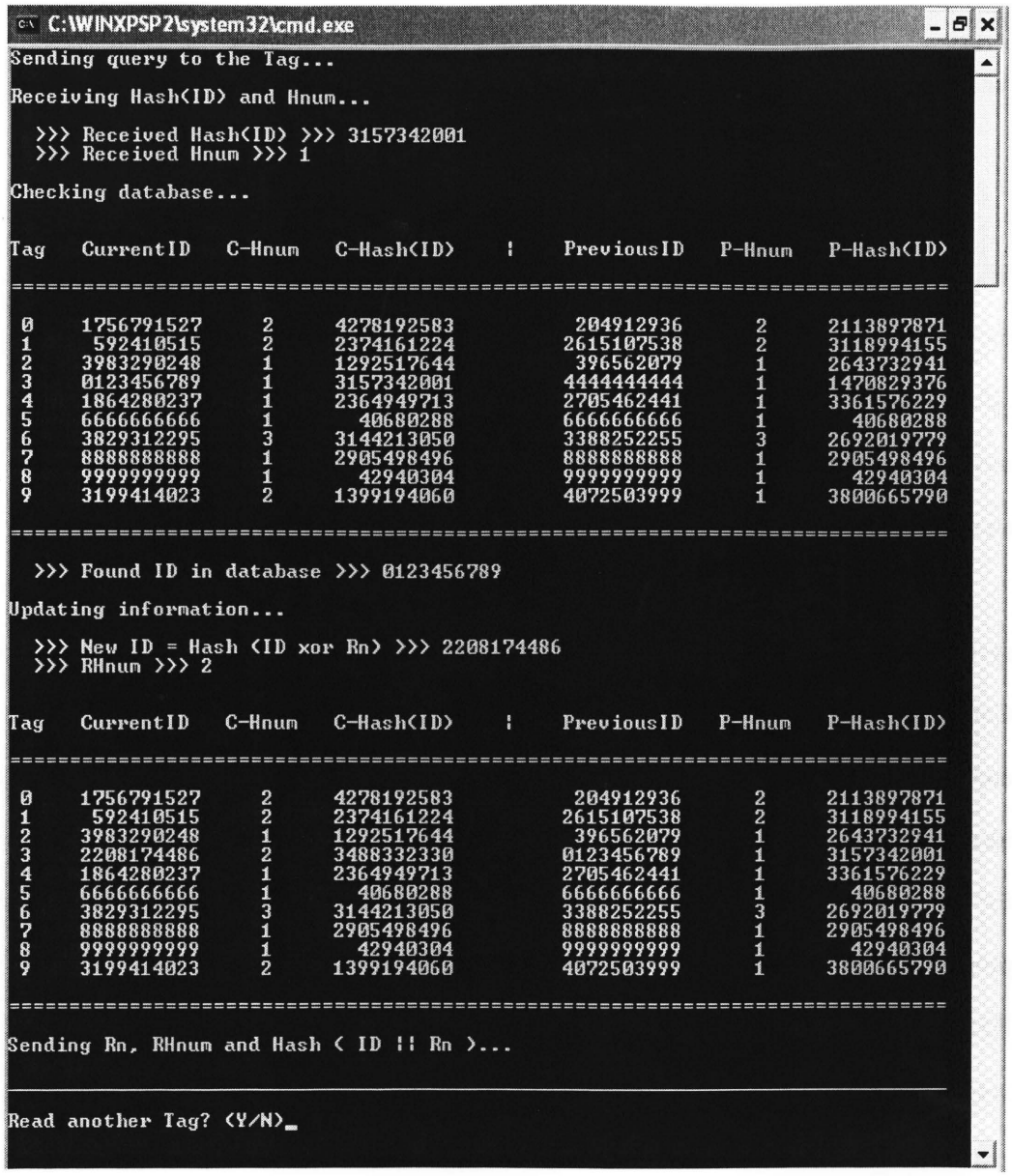

Figure 5.10: A Successful Authentication

The reader looks up the tag identifier in its database using the two received values. It finds the tag identifier *(ID_k= 0123456789)*. The RFID reader has authenticated the tag at this step.

Then, the reader calculates the new tag identifier by generating a random number (R_n) and a random hash function identifier $(R_{Hnum}=2)$:

$$new\ ID_k=Hash_{RHnum}(ID_k \oplus R_n)=2208174486$$

The reader, then, updates its database with the new information and sends the value of R_n, R_{Hnum}, and $Hash_{RHnum}(ID_k||R_n)$ to the tag. The RFID tag uses these three values to authenticate the reader. After successful authentication, the tag calculates its new identifier and updates the old one. As shown in Figure 5.10, the value of the H_{num} has been changed randomly, and for the next authentication between the RFID reader and tag, according to the new value of H_{num}, another hash function will be used to encrypt the secret information.

By using different hash functions through different authentication processes, the proposed system makes a crucial improvement to the security and privacy issues on an RFID system. Comparing with the SRAC protocol [34], the proposed system has doubled the levels of security and privacy of an RFID system. An eavesdropper needs more resources to reveal both the tag identifier and the hash function identifier. On the other hand, it will be much more complicated for the eavesdropper or attacker to find out and reveal all the hash functions deployed in the proposed protocol comparing with one hash function of the SRAC protocol. As a result, the RFID system will be considerably more secure against the usual threats including forward secrecy, cloning and forgery, replay attack, eavesdropping, and tracking.

Figure 5.11 demonstrates another successful authentication processes through the reader screen. As shown, the values of tag identifier (ID_k) and hash function identifier (H_{num}) have been changed after authentication. It is also shown that the old values will be stored in the reader's database to prevent the RFID system from denial of service attack.

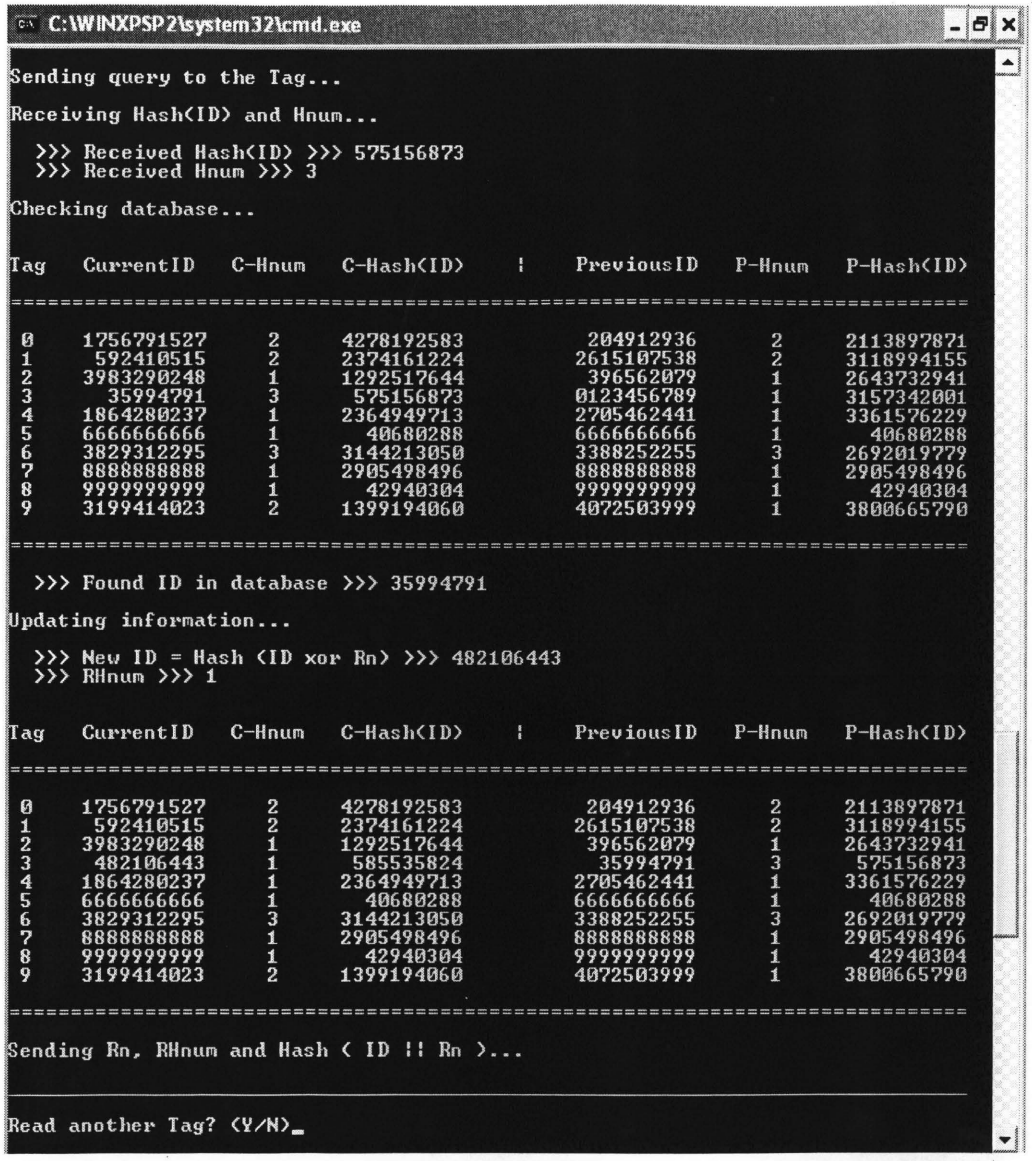

Figure 5.11: Randomly Updated Secret Information after Authentication

During the 3rd step of the proposed authentication protocol, when the RFID reader sends updating information to the tag, an attacker may generate a jamming signal causing a desynchronization between the tag and reader. In the proposed system, the reader keeps current

and pervious secret information of each tag to prevent denial of service attack. This is demonstrated and shown in Figure 5.12.

Figure 5.12: Resistant Against Denial of Service Attack

According to the data shown in Figure 5.12, after sending the query, the reader receives two values: hashed value of the tag identifier $(Hash_{Hnum}(ID_k)=3157342001)$, and its assigned hash function identifier $(Hnum=1)$. The reader looks up the tag identifier in its database using the two

received values. It does not find the tag identifier in its current list of information, instead, it finds the received values matching with a previous ID (*previous* ID_k= *0123456789)*. The RFID reader concludes that a denial of service attack has been occurred during the last authentication process of the tag. The reader, then, calculates the new tag identifier by generating a random number (R_n) and a random hash function identifier $(R_{Hnum}=3)$:

$$new\ ID_k=Hash_{RHnum}(ID_k \oplus R_n)=35994791$$

The reader updates its database with the new information and sends the value of R_n, R_{Hnum}, and $Hash_{RHnum}(ID_k||R_n)$ to the tag. The previous information of the tag will be kept as before in the reader's database.

Chapter 6

Conclusion

6.1 Concluding Remarks

The objective of this project is to present an FPGA-based RFID tag with a secure authentication protocol between the tag and reader. Privacy, security, scalability and flexibility are the main contributions of this project to the RFID systems while minimizing the overall design time and the system costs. This project report presented the design and implementation of an extensible flexible RFID tag that deploys a robust authentication protocol based on randomized access control. The system is designed so that to be highly resistant against known threats to RFID systems including forward secrecy, cloning and forgery, replay attack, eavesdropping, tracking, and denial of service attack. The RFID tag was implemented as a system on programmable chip using Nios II embedded processor.

The report started with an introduction to RFID system structure, motivation and objective. Then essentials of an RFID system are presented including tag and reader classifications, privacy and security issues, and RFID system performance. A survey on recently proposed and existing methods for solving RFID privacy and security issues is provided. The design and implementation steps of the proposed FPGA-based RFID tag with secure randomized access authentication protocol is presented in detail followed by the comparison of operational and security features of the proposed scheme with some existing technologies. Finally, the implementation details and verification results are provided.

6.2 Future Work

C-to-Hardware (C2H) acceleration

In order to increase security of the proposed RFID system, complex one-way hashing functions or standard encryption functions can be deployed within the authentication protocol; while the award winning Nios II embedded processor C-to-Hardware (C2H) acceleration compiler is being added to boost the performance and lower power consumption. This tool, provided by Altera, boosts the performance of time-critical ANSI C functions of the Nios II system by converting them into hardware accelerators in the FPGA.

The rule of thumb in system design has been that adding hardware increases power demands. The careful use of hardware accelerators, however, inverts the rule: adding hardware can reduce power. By analyzing algorithms and implementing appropriate accelerators in the programmable logic, developers can increase their design's performance while reducing power consumption in an embedded computing system.

Reconfigurable RFID tag

RFID tag development requires lengthy design, fabrication, and testing cycles which can take many months with intellectual property (IP) reuse to many years if developing new IP. A programmable customizable RFID tag can handle variations in standards and requirements as they are developed with a significantly shorter time to market. Such a tag could be mass produced and tailored to a particular RFID application after fabrication. With the use of automation to program the device, the design time could be reduced. Developing a reconfigurable tag can be the second suggested work for continuing this project.

Appendix A

RFID_Tag.vhd

```
1 -- Filename:    RFID_Tag.vhd
2 -- Description: A Nios II system for the RFID tag on DE2 board
3 -- Department:  Computer and Electrical Engineering of Ryerson University
4 -- Author:      Leili Borghei
5 -- Date:        November 2008
6
7 LIBRARY ieee;
8 USE ieee.std_logic_1164.all;
9 USE ieee.std_logic_arith.all;
10 USE ieee.std_logic_unsigned.all;
11
12 ENTITY RFID_Tag  IS
13        PORT (  CLOCK_50 : IN STD_LOGIC;
14                  DRAM_CLK, DRAM_CKE : OUT STD_LOGIC;
15                  DRAM_ADDR : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
16                  DRAM_BA_1, DRAM_BA_0 : BUFFER STD_LOGIC;
17                  DRAM_CS_N, DRAM_CAS_N, DRAM_RAS_N, DRAM_WE_N : OUT STD_LOGIC;
18                  DRAM_DQ : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
19                  DRAM_UDQM, DRAM_LDQM : BUFFER STD_LOGIC;
20                  UART_RXD : IN STD_LOGIC;
21                  UART_TXD : OUT STD_LOGIC );
22 END RFID_Tag ;
23
24 ARCHITECTURE Structure OF RFID_Tag IS
25
26 COMPONENT Nios_System
27        PORT (  clk : IN STD_LOGIC;
28                  reset_n : IN STD_LOGIC;
29
30                  zs_addr_from_the_sdram_0: OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
31                  zs_ba_from_the_sdram_0: BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
32                  zs_cas_n_from_the_sdram_0: OUT STD_LOGIC;
33                  zs_cke_from_the_sdram_0: OUT STD_LOGIC;
34                  zs_cs_n_from_the_sdram_0: OUT STD_LOGIC;
35                  zs_dq_to_and_from_the_sdram_0: INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
36                  zs_dqm_from_the_sdram_0: BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
37                  zs_ras_n_from_the_sdram_0: OUT STD_LOGIC;
38                  zs_we_n_from_the_sdram_0: OUT STD_LOGIC;
39
40                  rxd_to_the_uart: IN STD_LOGIC;
41                  txd_from_the_uart: OUT STD_LOGIC  );
42 END COMPONENT;
43
44 COMPONENT sdram_pll
45        PORT (  inclk0 : IN STD_LOGIC;
46                  c0 : OUT STD_LOGIC );
47 END COMPONENT;
48
49 SIGNAL BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
50 SIGNAL DQM : STD_LOGIC_VECTOR(1 DOWNTO 0);
51
52 BEGIN
53      --BA <= (DRAM_BA_1 & DRAM_BA_0);
54      --DQM <= (DRAM_UDQM & DRAM_LDQM);
55      DRAM_BA_1 <= BA(1);
56      DRAM_BA_0 <= BA(0);
57
58      DRAM_UDQM <= DQM(1);
59      DRAM_LDQM <= DQM(0);
60
61      -- Instantiate the Nios II system entity generated by the SOPC Builder.
62      NiosII: Nios_System PORT MAP (CLOCK_50, '1',
63                              DRAM_ADDR, BA, DRAM_CAS_N, DRAM_CKE,
64                              DRAM_CS_N, DRAM_DQ, DQM, DRAM_RAS_N, DRAM_WE_N,
65                                      UART_RXD, UART_TXD);
66
67      -- Instantiate the entity sdram_pll (inclk0, c0).
68      neg_3ns: sdram_pll PORT MAP (CLOCK_50, DRAM_CLK);
69 END Structure;
```

RFID_Tag.c

```
1 /* Filename:    RFID_Tag.c
2  * Description: RFID Tag in C running on NIOS-II soft processor
3  * Department:  Computer and Electrical Engineering of Ryerson University
4  * Author:      Leili Borghei
5  * Date:        November 2008
6  ********************************************************************************/
7
8 #include <stdio.h>
9 #include <string.h>
10 #include <stdlib.h>
11
12
//=============================================================================================
13 /*
14   Hash(char h, char* str, unsigned int len) includes 3 hash functions to generate
15   the encrypted form of Tag ID. Any of the 3 hash functions may be selected according
16   to the assigned hash ID of the Tag which is randomly change
17 */
18
//=============================================================================================
19 unsigned int Hash(char h, char* str, unsigned int len)
20 {
21    unsigned int hash = 0;
22    unsigned int i = 0;
23
24    // RSHash parameters
25    unsigned int b = 378551;
26    unsigned int a = 63689;
27
28    // BKDRHash parameters
29    unsigned int seed = 131;
30
31    switch (h){
32       // ------------ RSHash function -------------
33       // ------- Robert Sedgwicks Algorithm -------
34       case '1':
35          for(i = 0; i < len; str++, i++)
36          {
37             hash = hash * a + (*str);
38             a = a * b;
39          }
40          break;
41       // ------------------------------------------
42
43       // ------------ JSHash function -------------
44       // --------- Justin Sobel Algorithm ---------
45       case '2':
46          hash = 1315423911;
47          for(i = 0; i < len; str++, i++)
48          {
49             hash ^= ((hash << 5) + (*str) + (hash >> 2));
50          }
51          break;
```

```
52      // ----------------------------------------
53
54      // ---------- BKDRHash function ----------
55      // Brian Kernighan & Dennis Ritchie Algorithm
56      case '3':
57         for(i = 0; i < len; str++, i++)
58         {
59            hash = (hash * seed) + (*str);
60         }
61         break;
62      // ----------------------------------------
63   }
64      return hash;
65 }    .
66
67
//=============================================================================
68 /*
69   main () function is defined for sending Tag secret information to the Reader,
70   authenticating Reader and updating Tag information, using a secure
71   authentication protocol
72 */
73
//=============================================================================
74 int main()
75 {
76   int step;
77   int i;
78
79   int Rn;                    // Random number for choosing hash function
81                              //  (received from the Reader)
82   char RecIDcatRn[257];  // Hash(ID||Rn) from the Reader
83   char IDcatRn[257];     // Hash(ID||Rn)
84
85   char buffer[257];
86
87   char query[257];
88
89   char temp1[256+1];
90   char temp2[256+1];
91
92   char tempID[257];
93   char MetaID[257];
94
95   // -------------------- This Tag's Progile -------------------------
96
97   char ID[257]= "0123456789\0";
98   char Hnum[257] = "1\0";
99
100  // ------------------------------------------------------------------
101
102  FILE *fs;
103
104  strcpy(query,"HELLO\0");
105
106  printf("Beginning... \n\n");
```

```
107
108 fs=fopen ("/dev/uart", "rw+"); //open file for reading and writing
109
110 step = 2;                          // Waiting for query from the Reader
111
112 do{
113 switch (step){
114
115  /* Step 2: Tag receives query from the Reader,
116          and sends MetaID=Hash(ID) to the Reader */
117  case 2:
118     if (fs){
119        do{
120           printf("Waiting for query... \n");
121           fscanf(fs,"%s",buffer);
122        }. while (strcmp (buffer,query) != 0);
123        printf("  >>> Query received. \n\n");
124     }
125
126     printf ("Sending Hash(ID) and Hnum to the Reader... : %u , %s \n\n", Hash(Hnum[0],ID,strlen(ID)),Hnum
);
127     if (fs){
128        sprintf(MetaID,"%u",Hash(Hnum[0],ID,strlen(ID)));
129        fwrite(MetaID, 256, 1, fs);          //serial port buffer length is 256
130     }
131
132     if (fs){
133        fwrite(Hnum, 256, 1, fs);            //serial port buffer length is 256
134     }
135
136     step = 4;                            // waiting for update information from the Reader
137     break;
138
139  /* Step 4: Tag receives update information from the Reader,
140          and updates ID if Hash(ID||Rn) is equal to the received information */
141  case 4:
142     if (fs){
143        printf("Waiting for update information from the Reader... \n");
144        //do{
145        fscanf(fs,"%s",buffer);
146        //} while (strcmp (buffer,NULL) == 0);
147        if(!strcmp(buffer,"Failed\0"))
148        {
149           printf("  >>> Authentication FAILED!\n");
150           printf("  ==========================\n\n");
151           step=2;
152           break;
153        }
154        else
155        {
156           Rn = atoi(buffer);
157           printf("  >>> Received Rn: %d \n", Rn);
158
159           //do{
160           fscanf(fs,"%s",buffer);
161           //} while (strcmp (buffer,NULL) == 0);
```

```
162             strncpy(Rh,buffer,1);
163             Rh[1]='\0';
164             printf(" >>> Received RHnum >>> %s \n", Rh);
165
166             //do{
167             fscanf(fs,"%s",buffer);
168             //} while (strcmp (buffer,NULL) == 0);
169             strcpy(RecIDcatRn,buffer);
170             printf(" >>> Received Hash(ID||Rn) >>> %s \n\n", RecIDcatRn);
171
172             strcpy(temp1,ID);               // temp1: ID
173             sprintf(temp2,"%d",Rn);         // temp2: Rn
174             strcat(temp1,temp2);            // temp1: ID||Rn
175             sprintf(IDcatRn,"%u",Hash(Rh[0],temp1,strlen(temp1)));
176
177             printf("Verifying received information...\n");
178             //printf("%s\n",RecIDcatRn);
179             //printf("%s\n",IDcatRn);
180             if(!strcmp(RecIDcatRn,IDcatRn)) //checks whether Hash(ID||Rn) is correct
181             {
182                i=0;
183                for(i=0; i<strlen(ID); i++)
184                {
185                   tempID[i]= ID[i]^Rn;
186                }
187                tempID[i]='\0';
188                sprintf(ID,"%u",Hash(Rh[0],tempID,strlen(tempID)));
189                strcpy(Hnum,Rh);
190                printf(" >>> Verified >>> ID updated >>> New ID = Hash (ID xor Rn): %s \n",ID);
191                printf(" >>> Hnum >>> %s \n",Hnum);
192                printf("----------------------------------------\n\n");
193             }
194             else
195             {
196                printf(" >>> NOT verified! >>> ID is not updated!\n");
197                printf("----------------------------------------\n\n");
198             }
199             step=2;
200          }
201       }
202       break;
203    }
204    }while(1);
205
206    fclose(fs);
207
208    return 0;
209 }
210
211
212
```

Reader.cpp

```
1 /* Filename:    Reader.cpp
2  * Description: RFID Reader in C++
3  * Department:  Computer and Electrical Engineering of Ryerson University
4  * Author:      Leili Borghei
5  * Date:        November 2008
6  ********************************************************************************/
7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <stdafx.h>
11 #include <iostream>
12 #include <string.h>
13 #include <windows.h>
14 #include <time.h>
15
16 using namespace std;
17
18 HANDLE hSerial;
19
20
//================================================================================
21 /*
22   SerialComm () is used to set up serial communication through serial port on PC.
23   Parameters are defined as following:
24         Baud Rate = 19200
25         Byte Size = 8
26         Stop bits = 1
27         Parity = None
28 */
29
//================================================================================
30 void SerialComm ()
31 {
32       hSerial = CreateFile("COM1",
33                               GENERIC_READ | GENERIC_WRITE,
34                                                       0,
35                                                       0,
36                                                       OPEN_EXISTING,
37                                                       FILE_ATTRIBUTE_NORMAL,
38                                                       0);
39
40       if(hSerial==INVALID_HANDLE_VALUE){
41               printf("Error: Serial port does not exist...\n");
42               }
43
44       DCB dcbSerialParams = {0};
45
46       dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
47
48       if (!GetCommState(hSerial, &dcbSerialParams)) {
49               printf("Error in getting serial port state...\n");
50       }
51
```

```
52      dcbSerialParams.BaudRate =CBR_19200;
53      dcbSerialParams.ByteSize =8;
54      dcbSerialParams.StopBits =ONESTOPBIT;
55      dcbSerialParams.Parity =NOPARITY;
56      dcbSerialParams.fRtsControl =RTS_CONTROL_DISABLE;
57      dcbSerialParams.fDtrControl =DTR_CONTROL_DISABLE;
58
59      if(!SetCommState(hSerial, &dcbSerialParams)){
60              printf("Error in setting serial port state...\n");
61      }
62
63      COMMTIMEOUTS timeouts={0};
64      timeouts.ReadIntervalTimeout=50;
65      timeouts.ReadTotalTimeoutConstant=50;
66      timeouts.ReadTotalTimeoutMultiplier=10;
67      timeouts.WriteTotalTimeoutConstant=50;
68      timeouts.WriteTotalTimeoutMultiplier=10;
69
70      if(!SetCommTimeouts(hSerial, &timeouts)){
71              printf("Error in setting timeout...\n");
72      }
73 }
74
75
//=============================================================================
76 /*
77   Hash(char h, char* str, unsigned int len) includes 3 hash functions to generate
78   the encrypted form of Tag ID. Any of the 3 hash functions may be selected according
79   to the assigned hash ID of the Tag which is randomly change
80 */
81
//=============================================================================
82 unsigned int Hash(char h, char* str, unsigned int len)
83 {
84   unsigned int hash = 0;
85   unsigned int i = 0;
86
87   // RSHash parameters
88   unsigned int b = 378551;
89   unsigned int a = 63689;
90
91   // BKDRHash parameters
92   unsigned int seed = 131;
93
94   switch (h){
95     // ------------ RSHash function -------------
96     // ------- Robert Sedgwicks Algorithm -------
97     case '1':
98       for(i = 0; i < len; str++, i++)
99       {
100          hash = hash * a + (*str);
101          a = a * b;
102       }
103       break;
104    // ----------------------------------------------
105
```

```
106       // ------------ JSHash function ------------
107       // --------- Justin Sobel Algorithm ---------
108       case '2':
109          hash = 1315423911;
110          for(i = 0; i < len; str++, i++)
111          {
112             hash ^= ((hash << 5) + (*str) + (hash >> 2));
113          }
114          break;
115       // -----------------------------------------
116
117       // ----------- BKDRHash function ------------
118       // Brian Kernighan & Dennis Ritchie Algorithm
119       case '3':
120          for(i = 0; i < len; str++, i++)
121          {
122             hash = (hash * seed) + (*str);
123          }
124          break;
125       // -----------------------------------------
126    }
127    return hash;
128 }
129
130
//=============================================================================
131 /*
132    main () function is defined for reading, authenticating and updating Tag
133    secret information using a secure authentication protocol
134 */
135
//=============================================================================
136 int main(){
137
138    DWORD dwBytesWrite= 0;
139    DWORD dwBytesRead = 0;
140
141    char buffer[256+1]={0};
142    char tempBuffer1[256+1];
143    char tempBuffer2[256+1];
144
145    int i;
146    int step;
147    int finished=0;
148
149    char rec_MetaID[257];
150    char rec_Hnum[257];
151
152    bool found_ID;
153    bool DOS;                    //Denial OF Service attack
154
155    int Rn;                      //Random number
156    char Rh[257];                //Random Hash number
157    char tempID[257];            //temporary ID
158
159    int tagIndex;
```

```
160
161     char YesNo[5];
162
163     class tag {
164             public:
165                     char currID[257];           // Current ID
166                     char CmetaID[257];          // Hash of Current Id
167                     char Hc[257];               // Current Hash function
168                     char prevID[257];           // Previous ID
169                     char PmetaID[257];          // Hash of Previous Id
170                     char Hp[257];               // Previous Hash function
171     };
172
173     FILE *q;
174     tag tagArray[10];
175
176     SerialComm ();
177     step = 1;
178
179     do{
180     switch (step)
181     {
182             /* Step 1: Reader sends query to Tag */
183             case 1:
184                     printf("Sending query to the Tag... \n\n");
185
186                     lstrcpy((LPTSTR)buffer,"HELLO \0");
187                     WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
188                     lstrcpy((LPTSTR)buffer," \n");
189                     WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
190
191                     step=3;
192                     break;
193
194             /* Step 3: Reader receives MetaID=hash(ID) from the Tag,
195                       checks database for valid ID,
196                       updates ID, and
197                       sends update information to the Tag */
198             case 3:
199                     // Receive Hash(ID) and Hnum from the Tag
200                     printf("Receiving Hash(ID) and Hnum... \n\n");
201
202                     if(!ReadFile(hSerial, buffer, 256, &dwBytesRead, NULL))
203                     {
204                             printf("//Communication error occurred. 1\n");
205                             finished=1;
206                             break;
207                     }
208                     else
209                     {
210                             strcpy(rec_MetaID, buffer);
211                             rec_MetaID[dwBytesRead]='\0';
212                             printf("  >>> Received Hash(ID) >>> %s \n", rec_MetaID);
213                     }
214
215                     if(!ReadFile(hSerial, buffer, 256, &dwBytesRead, NULL))
```

```
216                     {
217                             printf("//Communiation error occurred. 1\n");
218                             finished=1;
219                             break;
220                     }
221                     else
222                     {
223                             strncpy(rec_Hnum, buffer,1);
224                             rec_Hnum[1]='\0';
225                             printf(" >>> Received Hnum >>> %s \n\n", rec_Hnum);
226                     }
227
228                     // Copy database to Tag array
229                     printf("Checking database... \n\n\n");
230
231                     q=fopen("C:\\Documents and Settings\\Dear-User\\My Documents\\Visual Studio
2005\\Projects\\Reader\\ID_file.dat","r");
232
233                     printf("Tag  CurrentID  C-Hnum  C-Hash(ID)  |  PreviousID  P-Hnum  P-Hash(ID)
\n");
234
        printf("=================================================================================
=======\n\n");
235                     for(i=0; i<10; i++)
236                     {
237                             printf("%2d",i);
238                             fscanf(q,"%s",tagArray[i].currID);          // read ID from database
239                             printf("%14s  ",tagArray[i].currID);
240
241                             fscanf(q,"%s",tagArray[i].Hc);
242                             printf("   %s",tagArray[i].Hc);
243
244                             fscanf(q,"%s",tagArray[i].CmetaID);         // read MetaId from database
245                             printf(" %14s  ",tagArray[i].CmetaID);
246
247                             printf("  ");
248
249                             fscanf(q,"%s",tagArray[i].prevID);          // read ID from database
250                             printf("%15s  ",tagArray[i].prevID);
251
252                             fscanf(q,"%s",tagArray[i].Hp);
253                             printf("    %s",tagArray[i].Hp);
254
255                             fscanf(q,"%s",tagArray[i].PmetaID);         // read MetaId from database
256                             printf("%15s\n",tagArray[i].PmetaID);
257         }
258
259                     fclose(q);
260
        printf("\n================================================================================
========\n\n");
261
262                     // Check database for valid ID
263                     i=0;
264                     found_ID=false;
265                     do{
```

```
266                     if (!strcmp(rec_MetaID, tagArray[i].CmetaID) && !strcmp(rec_Hnum,
tagArray[i].Hc))
267                     {
268                             printf(" >>> Found ID in database >>> %s \n\n", tagArray[i].currID);
269                             tagIndex=i;
270                             found_ID=true;
271                     }
272                     i=i+1;
273             } while ( i<10 && !(found_ID));
274
275             // ID not found >>> Check database for previous ID in case DOS has happened!
276             i=0;
277             DOS=false;
278             if (!(found_ID))
279             {
280                     do{
281                             if (!strcmp(rec_MetaID, tagArray[i].PmetaID) && !strcmp(rec_Hnum,
tagArray[i].Hp))
282                             {
283                                     printf(" >>> Found ID in Previous database >>> %s \n",
tagArray[i].prevID);
284                                     printf(" >>> DOS has happened in last
authentication!!!\n\n");
285                                     tagIndex=i;
286                                     found_ID=true;
287                                     DOS=true;
288                             }
289                             i=i+1;
290                     } while ( i<10 && !(found_ID));
291             }
292
293     // Updates ID
294             if (found_ID)
295             {
296                     srand ( time(NULL) );           // initialize random seed
297                     Rn = rand() % 100 + 1;          // generate random number for XORing with ID
298
299                     srand ( time(NULL) );           // initialize random seed
300                     sprintf(Rh, "%d", (rand() % 3 + 1)); // generate random number for choosing
hash function
301                     Rh[1]='\0';
302
303                     printf("Updating information... \n\n");
304                     switch(DOS)
305                     {
306                             case false:
307                                     i=0;
308                                     for(i=0; i<lstrlen(tagArray[tagIndex].currID); i++)
309                                     {
310                                             tempID[i]= tagArray[tagIndex].currID[i]^Rn;
311                                     }
312                                     tempID[i]='\0';
313                                     sprintf(tempID,"%u",Hash(Rh[0],tempID,strlen(tempID)));
314
315                                     // Updates current and previous ID
316                                     strcpy(tagArray[tagIndex].prevID,tagArray[tagIndex].currID);
```

```
317                                             strcpy(tagArray[tagIndex].currID,tempID);
318
319                                             strcpy(tagArray[tagIndex].Hp,tagArray[tagIndex].Hc);
320                                             strcpy(tagArray[tagIndex].Hc,Rh);
321
322
        strcpy(tagArray[tagIndex].PmetaID,tagArray[tagIndex].CmetaID);
323
        sprintf(tagArray[tagIndex].CmetaID,"%u",Hash(Rh[0],tempID,strlen(tempID)));
324                                             break;
325
326                                     case true:
327                                             // Denial of Service attack has happened in last authentication
328                                             // that Tag could not update its ID
329                                             i=0;
330                                             for(i=0; i<lstrlen(tagArray[tagIndex].prevID); i++)
331                                             {
332                                                     tempID[i]= tagArray[tagIndex].prevID[i]^Rn;
333                                             }
334                                             tempID[i]='\0';
335                                             sprintf(tempID,"%u",Hash(Rh[0],tempID,strlen(tempID)));
336
337                                             // Updates only current ID
338                                             strcpy(tagArray[tagIndex].currID,tempID);
339
340                                             strcpy(tagArray[tagIndex].Hc,Rh);
341
342
        sprintf(tagArray[tagIndex].CmetaID,"%u",Hash(Rh[0],tempID,strlen(tempID)));
343                                             break;
344                                     }
345
346                                     printf("  >>> New ID = Hash (ID xor Rn) >>> %s \n", tempID);
347                                     printf("  >>> RHnum >>> %s \n\n\n", Rh);
348
349                                     q=fopen("C:\\Documents and Settings\\Dear-User\\My Documents\\Visual
Studio 2005\\Projects\\Reader\\ID_file.dat","w+");
350
351                                     // Save Tag array in database and show updated database
352                                     printf("Tag  CurrentID  C-Hnum  C-Hash(ID)  |  PreviousID  P-Hnum  P-
Hash(ID)  \n");
353
        printf("=================================================================================
=======\n\n");
354                                     for(i=0; i<10; i++)
355                                     {
356                                             printf("%2d",i);
357                                             fprintf(q,"%s\n",tagArray[i].currID);           // write ID into database
358                                             printf("%14s  ",tagArray[i].currID);
359
360                                             fprintf(q,"%s\n",tagArray[i].Hc);
361                                             printf("  %s",tagArray[i].Hc);
362
363                                             fprintf(q,"%s\n",tagArray[i].CmetaID);          // write MetaId into
database
364                                             printf(" %14s  ",tagArray[i].CmetaID);
```

```
365
366                                 printf("  ");
367
368                                 fprintf(q,"%s\n",tagArray[i].prevID);          // write ID from database
369                                 printf("%15s  ",tagArray[i].prevID);
370
371                                 fprintf(q,"%s\n",tagArray[i].Hp);
372                                 printf("   %s",tagArray[i].Hp);
373
374                                 fprintf(q,"%s\n\n\n",tagArray[i].PmetaID);     // write MetaId into
database
375                                 printf("%15s\n",tagArray[i].PmetaID);
376                         }
377
378
        printf("\n==========================================================================
==========\n\n");
379
380                         fclose(q);
381
382                         // Send update information to the Tag: Rn and hash(ID||Rn)
383                         printf("Sending Rn, RHnum and Hash ( ID || Rn )... \n\n");
384
385                         sprintf(tempBuffer1,"%d ",Rn);              //int to char
386                         lstrcpy((LPTSTR)buffer,tempBuffer1);
387                         WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
388
389                         //printf("Rn: %d.\n",Rn);
390
391                         sprintf(tempBuffer1,"%s ",Rh);
392                         lstrcpy((LPTSTR)buffer,tempBuffer1);
393                         WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
394
395                         //printf("RHnum: %s.\n",Rh);
396
397                         strcpy(tempBuffer1,tagArray[tagIndex].prevID); // buffer1: ID, of course
Previous one!
398                         sprintf(tempBuffer2,"%d",Rn);                  // buffer2: Rn
399                         strcat(tempBuffer1,tempBuffer2);               // buffer1: ID||Rn
400                         sprintf(tempBuffer2,"%u ",Hash(Rh[0],tempBuffer1,strlen(tempBuffer1))); //
buffer2: Hash(RN||ID)
401
402                         lstrcpy((LPTSTR)buffer,tempBuffer2);
403                         WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
404
405                         //printf("Hash(ID||Rn): %u.\n",Hash(Rh[0],tempBuffer1,strlen(tempBuffer1)));
406                 }
407                 else
408                 {
409                         printf("  >>> Could NOT find ID in database! \n\n");
410                         printf("  >>> Authentication FAILED! \n\n");
411                         lstrcpy((LPTSTR)buffer,"Failed ");
412                         WriteFile(hSerial,buffer,lstrlen(buffer),&dwBytesWrite,NULL);
413                 }
414
```

```
415
        printf("_____\n\
n");
416                     printf("Read another Tag? (Y/N)");
417                     scanf("%s",YesNo);
418
        printf("_____\n\
n");
419                     if (!strcmp(YesNo,"Y")||!strcmp(YesNo,"y"))
420                     {
421                             printf("\n\n\n");
422                             step=1;
423                     }
424                     else
425                     {
426                             finished=1;
427                     }
428                     break;
429     }
430     } while(!finished);
431
432     CloseHandle(hSerial);
433
434     return 0;
435 }
436
437
438
439
```

References

[1] B. Song and C. J. Mitchell, "*RFID Authentication Protocol for Low-cost Tags*", Proceedings of the First ACM Conference on Wireless Network Security, Alexandria, VA, USA, Page(s): 140-147, March - April 2008.

[2] M. R. Rieback, B. Crispo, and A. S. Tanenbaum, "*The Evolution of RFID Security*", IEEE Pervasive Computing, Volume 5, Issue 1, Page(s): 62-69, January - March 2006.

[3] Altera® Development and Education (DE2) Board Webpage: <http://www.altera.com/education/univ/materials/boards/unv-de2-board.html>

[4] A. Juels, "*RFID Security and Privacy: A Research Survey*", IEEE Journal on Selected Areas in Communications, Volume 24, Issue 2, Page(s): 381-394, February 2006.

[5] S. Boumerdassi, P. K. Diop, E. Renault, and A. Wei, "*T2MAP: A Two-Message Mutual Authentication Protocol for Low-Cost RFID Sensor Networks*", IEEE 64th Vehicular Technology Conference, Montreal, Canada, Page(s): 1-5, September 2006.

[6] S. Lewis, "*A Basic Introduction to RFID Technology and its Use in the Supply Chain*", Laran RFID, White paper, 2004.

[7] S. Preradovic and N. C. Karmakar, "*RFID Readers - A Review*", International Conference on Electrical and Computer Engineering, Dhaka, Page(s): 100-103, December 2006.

[8] Alien Technology Corporation Webpage: <http://www.alientechnology.com/readers/index.php>

[9] Samsys Technologies Inc. Webpage: <http://old.samsys.com/default.php>

[10] RF Code Inc. Webpage: <http://www.rfcode.com>

[11] IDENTEC SOLUTIONS Inc. Webpage: <http://www.identecsolutions.com>

[12] ActiveWave Inc. Webpage: <http://www.activewaveinc.com>

[13] C. H. Lim and T. Kwon, "*Strong and Robust RFID Authentication Enabling Perfect Ownership Transfer*", Conference on Information and Communications Security, Raleigh, North Carolina, USA, Page(s): 1-20, December 2006.

[14] S. Karthikeyan and M. Nesterenko, "*RFID Security without Extensive Cryptography*", Proceedings of the 3rd ACM Workshop on Security of Ad hoc and Sensor Networks, Alexandria, VA, USA, Page(s): 63-67, 2005.

[15] M. Ohkubo, K. Suzuki and S. Kinoshita, "*Cryptographic Approach to Privacy-Friendly Tags*", RFID Privacy Workshop, MIT, November 2003.

[16] C. C. Tan, B. Shengm, and Q. Li, "*Secure and Serverless RFID Authentication and Search Protocols*", IEEE Transactions on Wireless Communications, Volume 7, Issue 4, Page(s): 1400-1407, April 2008.

[17] J. Collins, "*Marks & Spencer Expands RFID Trial*", RFID Journal, http://www.rfidjournal.com/article/articleview/791/1/1/, February 2004.

[18] S. E. Sarma, S. A. Weis, and D. W. Engels, "*RFID Systems, Security and Privacy Implications*", 4th International Workshop on Cryptographic Hardware and Embedded Systems, Redwood Shores, CA, USA, Page(s): 454-469, 2002.

[19] S. Inoue and H. Yasuura, "*RFID Privacy Using User-Controllable Uniqueness*", RFID Privacy Workshop, MIT, Massachusetts, USA, November 2003.

[20] N. Good, J. Han, E. Miles, D. Molnar, D.Mulligan, L. Quilter, J. Urban, and D. Wagner, "*Radio Frequency Identification and Privacy with Information Goods*", Proceedings of the ACM workshop on Workshop on Privacy in the Electronic Society, Washington DC, USA, Page(s): 41-42, 2004.

[21] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, "*An Elliptic Curve Processor Suitable For RFID-Tags*", Cryptology ePrint Archive, Report 2006/227, Available at http://eprint.iacr.org, July 2006.

[22] H. M. Sun, C. E. Lu, and S. M. Chen, "*An authentication Protocol in Mobile RFID Environment*", IEEE Region 10 Conference, Taipei, Taiwan, Page(s): 1-4, October-November 2007.

[23] A. Juels and R. Pappu, "*Squealing Euros: Privacy Protection in RFID-Enabled Banknotes*", Financial Cryptography, Volume 2742, Page(s): 103-121, Springer Berlin/Heidelberg, 2003.

[24] A. Juels, R. L. Rivest, and M. Szydlo, "*The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy*", Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington DC, USA, Page(s): 103-111, 2003.

[25] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "*Universal Re-Encryption for Mixnets*", The Cryptographers' Track at the RSA Conference Cryptographers, San Francisco, CA, USA, Page(s): 163-178, February 2004.

[26] A. Juels, "*Minimalist Cryptography for Low-Cost RFID Tags*", 4th International Conference on Security in Communication Networks, Volume 3352, Lecture Notes in Computer Science, Page(s): 149-164, September 2004.

[27] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "*Strong Authentication for RFID Systems Using the AES Algorithm*", Workshop on Cryptographic Hardware and Embedded Systems, Cambridge, MA, USA, Volume 3156, Page(s): 357-370, Springer, August 2004.

[28] M. Lehtonen, T. Staake, F. Michahelles, and E. Fleisch, "*From Identification to Authentication - A Review of RFID Product Authentication Techniques*", Printed handout of Workshop on RFID Security, Graz, Austria, July 2006.

[29] S. Weis, S. Sarma, R. Rivest, and D. Engels, "*Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems*", 1st International Conference of Security in Pervasive Computing, Boppard, Germany, Page(s): 201-212, March 2003.

[30] D. Henrici and P. Muller, "*Hash-based Enhancement of Location Privacy for Radio-Frequency Identification Devices using Varying Identifiers*", Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, Orlando, FL, USA, Page(s): 149-153, March 2004.

[31] D. Molnar and D. Wagner, "*Privacy and security in library RFID: Issues, Practices, and Architectures*", Proceedings of the 11th ACM conference on Computer and communications security, Washington DC, USA, Page(s): 210-219, October 2004.

[32] M. Ohkubo, K. Suzuki, and S. Kinoshita, "*RFID Privacy Issues and Technical Challenges*", IEEE Engineering Management Review, Volume 35, Issue 2, Second Quarter 2007, Page(s): 51 - 51, 2007.

[33] D. H. Seo, J. M. Baek, J. Li, S. Y. Kang, I. Y. Lee, and H. G. Oh, "*A Study on Improved RFID Authentication Scheme*", International Conference on Multimedia and Ubiquitous Engineering, Seoul, Page(s): 567-572, April 2007.

[34] Y. K. Lee and I. Verbauwhede, "*Secure and Low-cost RFID Authentication Protocols*", Proceedings of the 2nd IEEE Workshop on Adaptive Wireless, St. Louis, Missouri, November 2005.

[35] Y. S. Jeong, N. Sun, Y. C. Hwang, K. S. Kim, and S. H. Lee, "*RFID Authentication Protocol Using Synchronized Secret Information*", Proceedings of the First International Symposium on Data, Privacy, and E-Commerce, Chengdu, China, Page(s): 459-461, November 2007.

[36] A. Juels and S. Weis, "*Authenticating Pervasive Devices with Human Protocols*", 25th Annual International Cryptology Conference, Santa Barbara, California, USA, Volume 3621, Page(s): 293-308, August 2005.

[37] N. J. Hopper and M. Blum, "*Secure Human Identification Protocols*", Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, Gold Coast, Australia, Page(s): 52-66, Volume 2248, December 2001.

[38] J. Bringer, H. Chabanne, and E. Dottax, "*HB++: A Lightweight Authentication Protocol Secure Against Some Attacks*", 2nd International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, Lyon, France, Page(s): 28-33, June 2006.

[39] D. Ranasinghe, D. Engels, and P. Cole, "*Security and Privacy: Modest Proposals for Low-Cost RFID Systems*", Auto-ID Labs Research Workshop, Zurich, Switzerland, September 2004.

[40] J. Lee, D. Lim, B. Gassend, G. E. Suh, M. Dijk, and S. Devadas, "*A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications*", Symposium on VLSI Circuits, Honolulu, Hawaii USA, Page(s): 176-179, June 2004.

[41] P. Tuyls, and L. Batina, "*RFID-tags for Anti-Counterfeiting*", In Topics in Cryptology CT-RSA - The Cryptographers' Track at the RSA Conference, San Jose, USA, Page(s): 115-131, volume 3860, February 2006.

[42] G. Tsudik, "*YA-TRAP: Yet another Trivial RFID Authentication Protocol*", Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops, Pisa, Italy, Page(s): 640-643, March 2006.

[43] C. Chatmon, T. V. Le, and M. Burmester, "*Secure Anonymous RFID Authentication Protocols*", Technical Report TR-060112, Florida State University, Department of Computer Science, Tallahassee, Florida, USA, 2006.

[44] A. K. Jones, R. Hoare, S. Dontharaju, S. Tung, R. Sprang, J. Fazekas, J. T. Cain, and M. H. Mickle, "*An Automated, FPGA-based Reconfigurable, Low-Power RFID Tag*", Journal of Microprocessors and Microsystems, Volume 31, Issue 2, Page(s): 116-134, March 2007.

[45] RSA Laboratories Webpage: < http://www.rsa.com/rsalabs/node.asp?id=2176>

[46] D. Henrici and P. Muller, "*Hash-based Enhancement of Location Privacy for Radio-Frequency Identification devices using Varying Identifiers*", Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communication Security, Orlando, FL, USA, Page(s): 149-153, March 2004.

[47] A. Partow, "*General Purpose String Hashing Algorithms*", http://www.partow.net/programming/hashfunctions/index.html

[48] R. Bayer, "*Windows Serial Port Programming*", March 2008.