1-1-2011

RFID security protocols simulation and prototyping

Xiaohui Yu
Ryerson University

RFID SECURITY PROTOCOLS SIMULATION AND PROTOTYPING

by

Xiaohui Yu

B. Eng, South China University of Technology, China, 1993

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2011

Author's Declaration

Abstract

RFID Security Protocols Simulation and Prototyping

©Xiaohui Yu, 2011

Master of Applied Science
Electrical and Computer Engineering
Ryerson University

As Radio Frequency Identification (RFID) technology achieves commercial success, its privacy and security issues are becoming a barrier to limit its potential for future state of the art applications. In this report, we present an investigation of the past and current research related to RFID security algorithms and protocols for product authentication. We also present a novel RFID security protocol based on eXtended Tiny Encryption Algorithm (XTEA). Analysis of the security and privacy level of our proposed protocol is performed using SystemC based modeling and different attack models are simulated to show that the protocol is robust and safe against major attacks. Furthermore, to evaluate the functionality and workability of the protocol in real application, prototypes of these attack models are implemented on FPGA platform. We also compare our proposed protocol technique with similar protocols presented in the near past that also use symmetric key algorithms to verify and demostrate main advantages of our protocol in terms of security and performance.

Acknowledgements

First and foremost I want to greatly thank my supervisor Dr. Gul N. Khan for his supervision and guidance throughout my research. His inspiration and resource has made this work exciting and fruitaful. His direction allowed me to consentrate on the goal without straying away from it. I highly appreciate his contributions of time, ideas and invaluable feedbacks. Without his patience and enthusiasm, I definately could not have made such a progress.

I would also like to express my innermost gratitude to my superisor Dr. Fei Yuan for his heartfelt support ever since the beginning of my graduate study. I am deeply grateful to him for allowing me to achieve further goals in this program. His considerate help and advice strongly encouraged me at critical moments in this work and his spirit of integrity and commitment greatly motivated me throughout my graduate study.

My time in the Microsystems Research lab was made enjoyable due to a group of labmates that has become a part of my life. My knowledge definately was enriched by the warm discussion and information sharing among group members. I want to sincerely thank all of them for their help when I encountered difficulty and needed their support.

My special thanks go to all my family members. Without their love and understanding, this work could never have been completed. I feel proud to have their strong support beside me whenever I need strength to move forward. Their happiness and joy is the best reward to me.

Table of Contents

1. Introduction 1

1.1 Background 1

1.2 The RFID Security 5

1.3 Security and Authentication Approaches 6

1.4 Motivation 7

1.5 Objective 8

1.6 Contribution of This Work 9

2. RFID Security and Encryption Techniques 10

2.1 RFID Security Attacks 10

2.2 Cryptographic Algorithms 11

2.2.1 Unkeyed Algorithms 12

2.2.2 Symmetric Key Algorithms 15

2.2.3 Asymmetric Key Algorithms 16

2.3 Symmetric Key Encryption and Decryption Algorithms 17

2.3.1 Advanced Encryption Standard 18

2.3.2 Tiny Encryption Algorithm 22

2.3.3 Extended Tiny Encryption Algorithm 24

2.4 Related Works of RFID Security Protocols 26

3. Novel RFID Authentication Key Management Protocol 37

3.1 System Assumptions 37

3.2 Robust RFID Authentication Key Management Protocol Design 38

3.3 Security Analysis 41

4. Modeling and Simulation 44

4.1 Simulation of Protocol Using SystemC 44

4.2 Cloning Attack Model 47

4.3 Replay Attack Model 49

4.4 Man-in-the-Middle Attack Model 52

4.5 Denial-of-Service Attack Model 54

4.6 Active Scanning Attack Model 55

5. Protocol Analysis by Prototyping 58

5.1 RFID Security Protocol Experimental Analysis System 58

5.2 The Active Scanning Attack Model 59

5.3 The Cloning Attack Model 61

5.4 The Replay Attack Model 64

5.5 The Denial-of-Service Attack Model 67

5.6 The Man-in-the-Middle Attack Model 70

6. Performance and Security Analysis 73

6.1 Security Level Analysis 73

6.2 Performance Analysis 78

6.2.1 Code Size 78

6.2.2 Communication Cost 82

6.2.3 Number of Clock Cycles 83

6.2.4 Cipher 91

6.2.5 Scalability 93

7. Conclusion and Future Works 95

Reference 98

List of Tables

2.1	Notation used in Toiruul and Lee’s Protocol	31
2.2	Notation used in Kim et al.’s Protocol	34
5.1	Active Scanning Attack Messages Exchanged	61
5.2	Cloning Attack Messages Exchanged	63
5.3	Replay Attack Messages Exchanged	66
5.4	Denial-of-Service Attack Messages Exchanged	69
5.5	Man-in-the-Middle Attack Messages Exchanged	72
6.1	Security Level Comparison	77
6.2	Code Size Comparison	81
6.3	Communication Cost Comparison	82
6.4	Timestamp Difference Measured in Our Protocol	85
6.5	Timestamp Difference Measured in Toiruul and Lee’s Protocol	87
6.6	Timestamp Difference Measured in Kim et al.’s Protocol	89
6.7	Number of Clock Cycles Comparison	90
6.8	Cipher Usage Comparison	91
6.9	Scalability Analysis	94

List of Figures

1.1 Basic Components of a RFID System 2

2.1 Symmetric Key Algorithm 15

2.2 Asymmetric Key Algorithm 17

2.3 AES Encryption 19

2.4 AddRoundKey Operation 20

2.5 S-Box Operation 20

2.6 ShiftRows Operation 21

2.7 AES Decryption 22

2.8 One Cycle of Tiny Encryption Algorithm 24

2.9 One Cycle of eXtended Tiny Encryption Algorithm 26

2.10 Hash-Locking Protocol 27

2.11 Randomized Hash-Locking Protocol 27

2.12 Hash-chain Protocol 28

2.13 Improved Security Solution 29

2.14 Toiruul and Lee’s Protocol 32

2.15 Kim et al.’s Protocol 35

3.1 Novel XTEA Based RFID Security Protocol 38

3.2 Key Update Message Format 39

3.3 Key Update Acknowledgement Format 40

4.1 SystemC Model of Proposed Protocol 45

4.2	State Diagram of the Reader	45
4.3	State Diagram of the Tag	46
4.4	SystemC Modeling and Simulation of the Proposed Protocol	47
4.5	Cloning Attack Model	48
4.6	Cloning Attack Model Simulation	49
4.7	Replay Attack Model	50
4.8	Replay Attack Model Simulation	51
4.9	Man-in-the-Middle Attack Model	52
4.10	Man-in-the-Middle Attack Model Simulation	53
4.11	Denial-of-Service Attack Model	54
4.12	Denial-of-Service Attack Model Simulation	55
4.13	Active Scanning Attack Model	56
4.14	Active Scanning Attack Model Simulation	56
5.1	Active Scanning Attack Model Implementation using Nios-II System	59
5.2	Active Scanning Attacker Module	60
5.3	Cloning Attack Model Implementation using Nios-II System	62
5.4	Reader Module for the Cloning Attack Model	63
5.5	Replay Attack Model Implementation using Nios-II System	64
5.6	Attacker Module for the Replay Attack Model	65
5.7	Denial-of-Service Attack Model Implementation using Nios-II System	67
5.8	Attacker Module for the Denial-of-Service Attack Model	68
5.9	Man-in-the-Middle Attack Model Implementation using Nios-II System	70
5.10	Attacker Module for the Man-in-the-Middle Attack Model	71

6.1 Toiruul and Lee's protocol Simulation 74

6.2 Simulation of Denial-of-Service Attack in Toiruul and Lee's Protocol 75

6.3 Simulation of Denial-of-Service Attack in Kim et al.'s Protocol 76

6.4 Nios-II System with Reader and Tag CPUs 79

6.5 Compilation of the Reader Code 80

6.6 Compilation of the Tag Code 81

6.7 Reader System with Timestamp Timer 83

6.8 Timestamp Difference Measured in our Protocol 84

6.9 Timestamp Difference in Our Protocol 86

6.10 Timestamp Difference Measured in Toiruul and Lee's Protocol 87

6.11 Timestamp Difference in Toiruul and Lee's Protocol 88

6.12 Timestamp Difference Measured in Kim et al.'s Protocol 89

6.13 Timestamp Difference in Kim et al.'s Protocol 90

Chapter 1

Introduction

1.1 Background

The “identification friend-or-foe” (IFF) cryptographic system for military aircraft command and control used during the second world war II is commonly regarded as the first Radio Frequency Identification (RFID) application [1]. However, the first patent of modern RFID transponder was submitted by Cardullo and Parks and published in 1973 [2]. The initial device was passive, powered by the interrogating signal and with a memory of 16 bits.

The popularity of this technology has been grown tremendously during recent years and nowadays, RFID is widely used and can be found almost everywhere. RFID systems are deployed widely in many applications including automation, manufacturing, logistics control and asset tracking. The fundamental trend of ubiquitous computing is to embed tiny advanced electronic device into the environment and make data processing automatic. As compared with other automatic data identification techniques, RFID enjoys numerous advantages such as longer reading range, larger storage capacity, higher data throughput, more rewritable capability and less human intervention [3]. Its support of multiple reading and non-line-of-sight data capture makes it useful in many applications. Even when working in adverse and harsh environments, tags are less sensitive to ambient conditions and produce robust and reliable performance.

A typical RFID system is shown in the Figure 1.1.

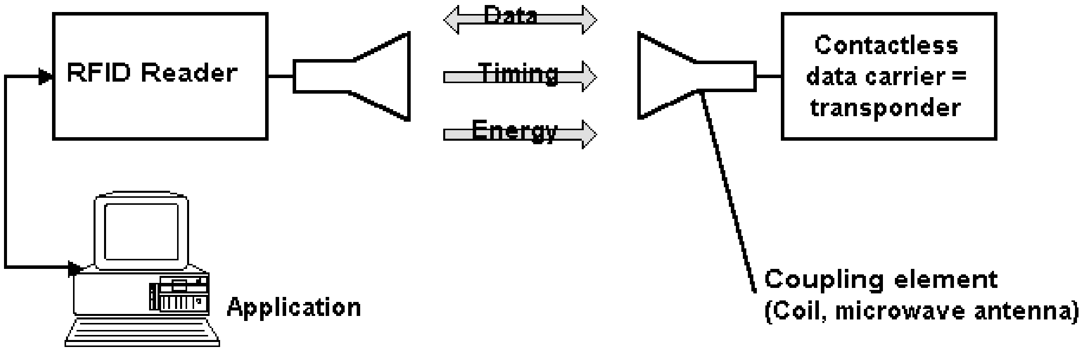

Data
Timing
Energy
RFID Reader
Contactless data carrier = transponder
Application
Coupling element (Coil, microwave antenna)

Figure 1.1 Basic Components of a RFID System (Courtesy www.rfid-handbook.de)

Every RFID application contains three basic components: tag (transponder), reader, and back-end database application or server. Because RFID systems radiate electromagnetic waves, other radio services might be disrupted or impaired by their operation if the frequency and power are not controlled properly. Therefore, it is only possible to use special frequency ranges that are reserved for industrial, scientific or medical applications or for short range devices. They are classified worldwide as Industrial-Scientific-Medical (ISM) frequency ranges and they can also be utilized in RFID applications [4]. Tags can be classified into two categories: the active and the passive [5]. Active tags must be connected to a powered infrastructure or use energy stored in an integrated battery, while the passive ones obtain power from readers. In terms of passive tag design, there are two fundamentally different methods to transfer power from the reader to the tag: magnetic induction and electromagnetic wave capture. Generally, active tags are more expensive due to the requirement of extra power source.

Each tag has a unique number that is often called the tag identification (TID). When the reader queries the tag, the tag responds with its identification number to the reader and then the reader forwards it to the back-end server. If the TID is verified, the back-end server processes the data and notifies the reader if the reading process succeeds or fails.

As RFID technology starts to achieve more commercial success, its security and privacy issues are becoming major concerns to the public and might become a great obstacle for its future development if these risks are not addressed properly. After all, readers have to communicate with tags through an unsecured radio channel, which is vulnerable to different kinds of attacks. A strong attacker is capable of intercepting, recording, analyzing, blocking and modifying the signals over the air. The wide deployment of RFID tags also implies that using this technology might bring unintended risks, because unauthorized data collection may happen.

Currently, a number of standards for RFID systems coexist. Most important standards include ISO 11784-5 for animal identification, ISO 15693 for vicinity card and ISO14443 for high-end RFID tags used as contact-less smartcards. EPC Global is a joint venture between European Article Number (EAN) international in Europe and Uniform Commercial Code (UCC) in USA, and currently it is responsible for the standardization of present barcode systems. The organization proposed electronic product code (EPC) Class-1 Generation-2 (C1G2) standard tags to replace or complement the existing barcode technology [6]. Normally, RFID tags have to be low-cost and highly efficient. It is estimated that the total cost per tag should not be more than five cents [7]. However, very limited security features can be deployed on a C1G2 tag because it only supports functions used in passive tags such as anti-collision, 16-bit cyclic redundancy code (CRC) checksum, 10-bit pseudo-random number generator (PRNG), and a simple kill command to ensure user privacy [8]. If a reader wishes to permanently disable a tag, it must send a kill command to the tag and prove its legitimacy via a 32-bit PIN number. Although this simple approach solves the privacy issue so that the user is not prone to be tracked anymore, the benefit of using RFID to update and analyze data is lost.

To find better solutions, numerous RFID security authentication proposals have been put forwarded and hundreds of papers have been published during recent years, but designing a good protocol to meet the requirements of a secure RFID system still remains challenging. An essential factor that causes the difficulty is the dynamic range of technologies RFID combines with, while the other reason is the wide area of its application. Juels's work has provided an in-depth and valuable analysis of this reality [7].

Active tags have more power source and storage memory than passive ones and therefore additional privacy features can be easily implemented on them to achieve strong security level. Unfortunately, due to the requirements of lower cost and less maintenance effort, most RFID tags are still passive nowadays and come with extremely limited computational resource and memory space. As a result, to achieve the desired level of security, any practical design of RFID protocol has to take into account the constraints and balance between the level of security, cost and performance.

1.2 The RFID Security

There are a number of security concerns in communication systems. Manfred and others summarize that a secure communication system should achieve following objectives [9]:

i) Confidentiality: Information is unreadable for adversaries. Only authorized party is able to interpret the information.
ii) Authentication: A party is able to be sure that the information received is from the source it is claimed to be.
iii) Integrity: It is assured that the message is not modified after being transmitted.
iv) Non-Repudiation: It can prove that the sender has actually sent out the message and the receiver has actually received the message.
v) Access Control: It refers to the capability to restrict a system and control the access to it.
vi) Availability: It means that whenever it is needed, the system is available. The system is guarded from loss of service and reduction in availability.

Since there is a lack of categorization of security level, Schechter proposed the concept of Cost To Break (CTB) as the measurement of security level [10]. He suggests a cryptographer should estimate the cost for an adversary to break a system by purchasing the latest and the cheapest tools and using the best known techniques. As an example, a system that takes an adversary not less than $1000 to break is secure enough to protect an item of $100 in value. In addition, another matrix of security level in a system is what punishment an attacker will get if it is caught by the security mechanism, and how likely hacking activities can be detected [11]. Apparently, more stringent punishment makes intruders more hesitant to launch attacks.

In order to define the concept of secure and privacy, A. Juels suggests that a formal security model that characterizes the capabilities of potential adversaries should be constructed [7]. As an example to compare with, in a security model for the Internet an adversary can have access to a server at any time, because the server have to respond to the queries coming from around the world all the time. On the other hand, around-the-clock access by attackers to RFID tags is an assumption that is too strong. An attacker has to come to a tag's proximity to transmit and receive messages and thus it is only a sporadic event. In another work, a "minimalist" security model is proposed such that an adversary comes into scanning range of a tag on a limited rate basis [12]. In the model, there is a cap on the number of times that an attacker may scan a real tag and impersonate a valid reader. Once the limit is reached, the tag should continue to interact with a legitimate reader.

1.3 Security and Authentication Approaches

RFID technology is very helpful in authenticating genuine products. Lehtonen et al. define authentication as "Product authentication = Identification + Verification of the Claimed Identity" [13]. There are many different ways to achieve authentication using RFID technology. In another work, Lehtonen et al. classify in details the approaches to authenticate RFID-based product [14]. Product authentication can be based on the authentication factors of what the product is (object-specific features-based authentication), what the product has (tag authentication) and where the product is (location-based authentication).

The object-specific features based authentication approach was proposed by Nochta et al. and their proposal is to make products hard to be cloned [15]. In this approach, the information about physical or chemical features which are unique to a particular product (such as weight, electric resistance, geometrics, and a serial number printed on the product itself) are stored on its tag. However, the cost of checking each product is generally very high.

Tag authentication approach takes advantages of tag security features that are hard to be cloned. General method is that a reader, through a challenge-response protocol, checks if its tag knows a secret shared by the authentication database in the back-end server. Normally, the secret should not be revealed in plain messages so that attackers may not be able to learn the secret.

The third approach to authenticate products is based on their locations. In such a system, the product carries only an identifier, whereas the back-end server registers all the information about the locations of genuine products. If the track and trace records show that a genuine product is at one place and in another place another product with the same identifier is found, it can be concluded that the second product is a fake one. Such an approach put the complexity of the system into the back-end server.

So far, the second approach appears to be the most economic way to enforce security in RFID systems. How to use robust authentication protocols to protect critical messages has drawn attention from researchers. As technologies are evolving and both the security systems and the potential adversaries are becoming more sophisticated, designing and implementing strong RFID authentication schemes will continue to be dynamic and challenging.

1.4 Motivation

Although, numerous RFID security authentication proposals have been put forward recently, very few of them address the security issues successfully and provide strong security level. A basic assumption made in the past works is that strong encryption algorithms such as symmetric key ciphers are beyond considerations for passive tags. For example, in Juels' early work, it is concluded that a basic passive tag cannot contain Advanced Encryption Standard (AES) algorithm, which is a symmetric key cipher [7]. However, a number of recent works have already presented efficient implementation of AES on the latest Complementary Metal Oxide Semiconductor (CMOS) technology [36, 51 and 52]. As CMOS technology continues to improve, symmetric key ciphers will become more and more common in robust RFID security solutions. However, how the security keys should be managed and handled safely and efficiently in the context of limited tag hardware resource still need more investigation and results should be proved and verified with experimental analysis and performance evaluation.

1.5 Objective

Although there are many considerations in a practical RFID security system, the practical requirements of any RFID protocol fall into only two basic categories:

i) Strong security level
ii) High performance level.

In this work, we will select a lightweight symmetric key algorithm suitable for the passive RFID application to provide desired security level. Based on that, we propose a protocol in which messages are securely handled so that hackers cannot obtain secrets easily. Furthermore, secrets must be dynamically updated as it is not safe to assume that secrets can never be disclosed.

Instead of relying only on theoretical analysis, the practical approach in this work will provide experimental analysis on simulation and prototyping. In order to demonstrate the advantages of our proposal, security and performance levels of our protocol must be compared with similar works that also use symmetric key algorithms.

1.6 Contribution of This Work

In this work, we propose a novel RFID security key management protocol based on symmetric key algorithm eXtended Tiny Encryption Algorithm (XTEA). We summarize our contributions as follows:

i) First of all, we analyze related works regarding RFID security algorithms and protocols and the issues of existing solutions. We review some recent development in RFID hardware resource for security solutions.
ii) Then, we propose a novel RFID authentication key management protocol based on XTEA. Key management is handled efficiently so that one sub-key is dynamically updated in every session.
iii) The proposed protocol is analyzed using SystemC based modeling and we present the simulation and modeling results of different types of attacks on the protocol to evaluate its security and privacy level.
iv) To evaluate the functionality and workability in real application, prototypes of these models are implemented on Altera Field-Programmable Gate Array (FPGA) platform using multi-processor architecture.

Chapter 2

RFID Security and Encryption Techniques

In the following sections, we discuss those research efforts that have inspired our work. First of all, we investigate different types of attacks that may happen in a RFID system. A look into the cryptographic primitives that can be applied in RFID application will follow. Furthermore, we will discuss some lightweight symmetric key algorithms proposed to be used in RFID systems. Finally, the last portion of this chapter also reviews and analyzes some related RFID protocols.

2.1 RFID Security Attacks

As potential adversaries can attack a RFID system in many different ways, a secure protocol should prevent following attacks [9]:

i) Active Scanning Attack: An adversary actively queries a genuine tag to get critical information without the authorization of tag's owner.

ii) Cloning Attack: Cloning attack means the duplication of security features in a system such that an attacker is likely to pass authentication inspection.

iii) Replay Attack: An attacker records the response of a tag and uses it as the message to replay when it is queried, though the message itself does not reveal the critical secrets of the tag.

iv) Man-in-the-middle Attack: It is a form of attack where the attacker acts like a reader to a tag as well as a tag to a reader.

v) Denial-of-service Attack: It is a type of attack caused by adversary to block the message exchange between the reader and the tag. The purpose of doing this is to desynchronize the reader and its tag.

vi) Forward Security: Assuming that all the secrets of a tag are compromised by an adversary at present, the security of past sessions is still guaranteed.

vii) Backward Security: Assuming that all the secrets of a tag are compromised by an adversary at present, backward security protects the security of future sessions.

2.2 Cryptographic Algorithms

To achieve high level security and prevent the threats as mentioned above, appropriate cryptographic algorithms must be applied in a communication system. If a series of algorithms must be used in a well-defined way, it is referred to as a protocol [9]. A protocol consists of the cryptographic algorithms that make original messages unreadable to adversaries and the process of how the messages are exchanged between different parties. Essentially, these two factors are equally important in a security solution.

Cryptographic algorithms are classified as unkeyed algorithms, symmetric key algorithms and asymmetric key algorithms [9]. Unkeyed algorithms are not based on any keys in their operations. Symmetric key algorithms use the same key for encryption and decryption, whereas asymmetric key algorithms use a public key for encryption and a private key for decryption.

2.2.1 Unkeyed Algorithms

Unkeyed algorithms can be briefly separated into two categories: hash functions and random number generators.

A hash function is a computationally efficient function y = h(x) with the following two properties [16]:

i) Compression – The hash function maps a string of arbitrary length to a sting of fixed length.

ii) Ease of computation – Given a hash function h(x) and an input x, it is easy to compute h(x).

For a good hash function to be used in cryptographic tools, a few requirements have to be satisfied. Firstly, the hash function must be a one way function and secondly, the function must be collision resistant. The first condition implies that there is no simple way and is computationally infeasible to reverse a good hash function. Since the number of input for the function is infinite while the output length is fixed, collision at the output is unavoidable. However, there should not be any simple method to find a pair of input x1 and x2 such that h(x1) = h(x2). Practically, the function should be a random mapping, which implies that distinguishing a hash function from a random mapping is computationally infeasible. Hash functions are very important primitives in cryptography. They are used when fixed length output is required instead of arbitrary length results.

Unkeyed hash functions are often called Modification Detection Codes (MDC) because of their capability to detect whether an original message is altered. Well-known MDCs include Message-Digest Algorithm 5 (MD5) and Secure Hash Algorithm 1 (SHA-1). MD5 is designed by Ron Rivest in 1992 to replace the earlier design of Message-Digest Algorithm 4 (MD4) [17]. It is simple to implement and provides 128-bit message digest of an arbitrary length input. It is intended for applications where a large file must be compressed. However, it has been shown that MD5 is not as collision resistant as it is claimed to be [18]. Therefore, it is no longer secure enough to be applied in protocols, such as the Secure Socket Layer (SSL) protocol for communications over Internet. SHA-1 is designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). SHA-1 is very similar to its predecessor Secure Hash Algorithm 0 (SHA-0), but it corrects a significant error found in SHA-0 that collision can be identified by using two differing 2048 bit files. It is the most widely used among the existing SHA hash functions. However, Wang et al. found its security flaws and attacks were announced [19]. It is shown in their work that collisions of SHA-1 can be found with complexity less than 2^{69} hash operations on the full 80 step SHA-1.

NIST published four additional hash functions in SHA-2 family, named after their digest length in bits, SHA-224, SHA-256, SHA-384, and SHA-512. Collectively these four algorithms are called SHA-2. The attacks found in SHA-1 are not extended to SHA-2. A new standard, specifically SHA-3 is currently under development. An NIST held public competition is underway and the new standard for SHA-3 is expected to be announced in 2012 [20]. Finding a good hash function is a challenging and on-going research area.

Another important type of unkeyed algorithms in cryptography is random number generators. Ideal source of random number should be based on physical means similar to coin flipping and dice rolling. Most random number generators are based on software algorithm. Since pure software algorithms are deterministic, they are not completely random. Therefore, these random number generators are called pseudo-random number generators [9].

Physical Unclonable Function (PUF) is another type of random number generators. A PUF is a function embedded in a physical structure that is easy to evaluate but hard to characterize. The first PUF device is invented by Naccache and Fremanteau [21]. Ranasinghe et al. presented an interesting design based on metastability and thermal noise and a PUF random number generator circuit was implemented using CMOS 0.18 μm technology [22]. It is proved that true random number source based on hardware method is feasible on standard CMOS technology. Recent literatures mostly focus on four types of PUF design [23]:

i) Ring Oscillator based PUFs

ii) Arbiter PUFs

iii) Memory based PUFs

iv) Glitch Counter-based PUFs

Unkeyed algorithms are important cryptography tools and are widely used in RFID security systems. As compared to symmetric key algorithms and asymmetric key algorithms, they require less hardware resource for implementation.

2.2.2 Symmetric Key Algorithms

Symmetric key algorithm is a system in which the sender and the receiver of a message share a common private key for encryption and decryption.

A system based on this type of algorithms is described in Figure 2.1.

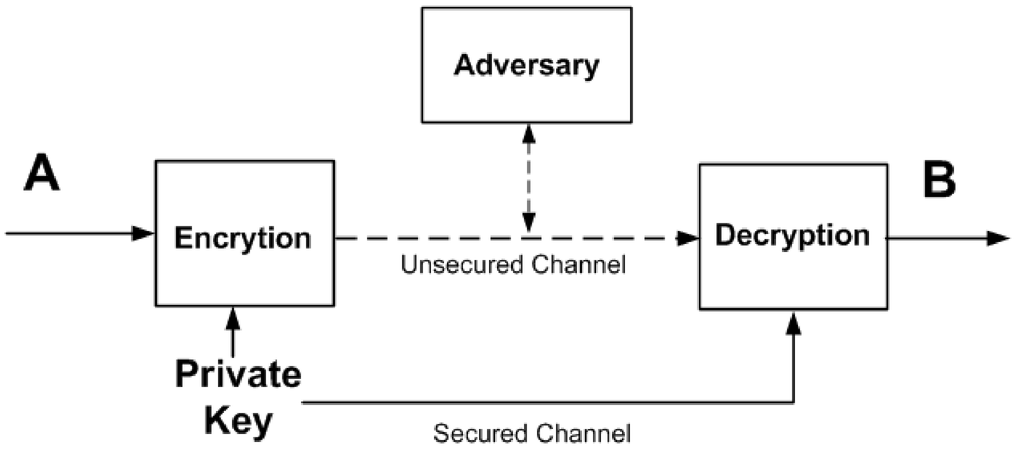


Figure 2.1 Symmetric Key Algorithm

In this cryptographic system, a sender A intends to transmit a confidential message to a receiver B through an unsecured channel without allowing any unauthorized party, such as an attacker, to understand the meaning. In order to achieve this goal, A and B agree to use a secret key for encryption and decryption. The private key has to be distributed to A and B through a secured channel. A uses an encryption algorithm to encrypt the message using the private key. Although the message is unreadable to adversaries, B is able to decrypt the message using the same private key. Symmetric key algorithms are simple and fast, but they have a drawback that the two parties must exchange the key in a secure way. As for a good symmetric key algorithm, it is computationally infeasible to recover the cipher text without knowing the key.

Symmetric key algorithms can be divided into block ciphers and stream ciphers. Block ciphers divides the message into fixed size blocks. Each block is transformed into a block of cipher text of the same length. The fixed length of the block is called the block size [9]. Decryption is performed using the same secret key. Typical block ciphers include Data Encryption Standard (DES), Advanced Encryption Standard (AES), Tiny Encryption Algorithm (TEA) and eXtended Tiny Encryption Algorithm (XTEA) etc.

Unlike block ciphers, stream ciphers operate on smaller units of data, such as bits [16]. A stream cipher generates what is called a keystream (a stream of random bits used as a key). Encryption is performed using the keystream to combine with the information that the sender needs to transmit. One example of stream cipher is Rivest Cipher 4 (RC4), which is designed by Rivest [24]. It is widely used in many protocols, such as Secure Socket Layer (SSL) and TLS (Transport Layer Security).

2.2.3Asymmetric Key Algorithms

Asymmetric key algorithms also referred to as public-key encryption algorithms assume that there exists a key source for a sender to obtain an authentic copy of the receiver’s public key [16]. It is also assumed that the receiver owns a private key which is abstained in a secure way. For a receiver, its public and private keys are normally different and not related. A system based on asymmetric key algorithm is shown in Figure 2.2. When A needs to send a message to B, it has to obtain B’s public key from the key source at first. Then the message is encrypted using the public key and is sent to B. Upon receiving the message, B decrypts it using its private key. Without knowing the private key, an adversary is not able to decrypt the message sent to B through the communication channel. Despite the fact that the public key is available to everyone, it cannot be utilized for decryption.

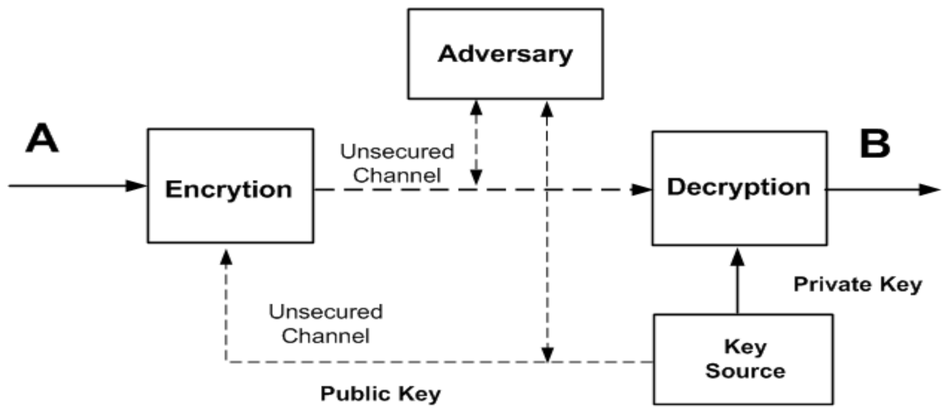


Figure 2.2 Asymmetric Key Algorithm

There are many techniques by which authentic public keys are distributed, such as exchanging keys over a trusted channel, using a trusted public file, using an on-line trusted server or using an off-line server and certificates. Public-key encryption schemes normally are substantially slower than symmetric key encryption algorithms [16]. Therefore, they are most commonly used to transport keys subsequently used for bulk data encryption by symmetric algorithms or to encrypt small data items such as credit card numbers or PIN number. Major examples of asymmetric key algorithms include Data Security Standard (DSS) and Rivest-Shamir-Adelman (RSA).

2.3 Symmetric Key Encryption and Decryption Algorithms

In the past, it is generally believed that symmetrical key algorithms are very power consuming, thus not suitable for traditional RFID technology. However, with the improvement of CMOS technologies, some lightweight symmetrical key algorithms have shown great potentials to be implemented in next generation passive tags.

2.3.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric key encryption and decryption algorithm that attracts a great deal of attention in RFID applications. It was first published by the National Institute of Standards and Technology (NIST) in 2001 to replace the Data Encryption Standard (DES) [25]. The algorithm was selected among fifteen qualifying candidates based on their security, performance and algorithm characteristics. The winner algorithm is called Rijndael designed by Daemen and Rijmen [26]. It is a symmetric cipher that can process 128-bit data blocks, using cipher keys with lengths of 128, 192, and 256 bits.

Figure 2.3 shows the encryption process of AES algorithm. Generally, there are four major operations involved (AddRoundKey, SubBytes, ShiftRows and MixColumns). All the operations in the block diagram are performed on a two-dimensional array of bytes that is called state. A state can be regarded as a rectangular array of bytes consisting of four rows and a number of columns defined by the block size in bytes divided by four. For example, a 128-bit block can be depicted as a state of four rows by four columns. Each round of operation is based on a round key derived from the encryption key. The number of rounds is dependent on the key size. For AES-128, the key length is 128-bits and the number of rounds is 10. The number of rounds for AES-192 and AES-256 are 12 and 14 respectively.

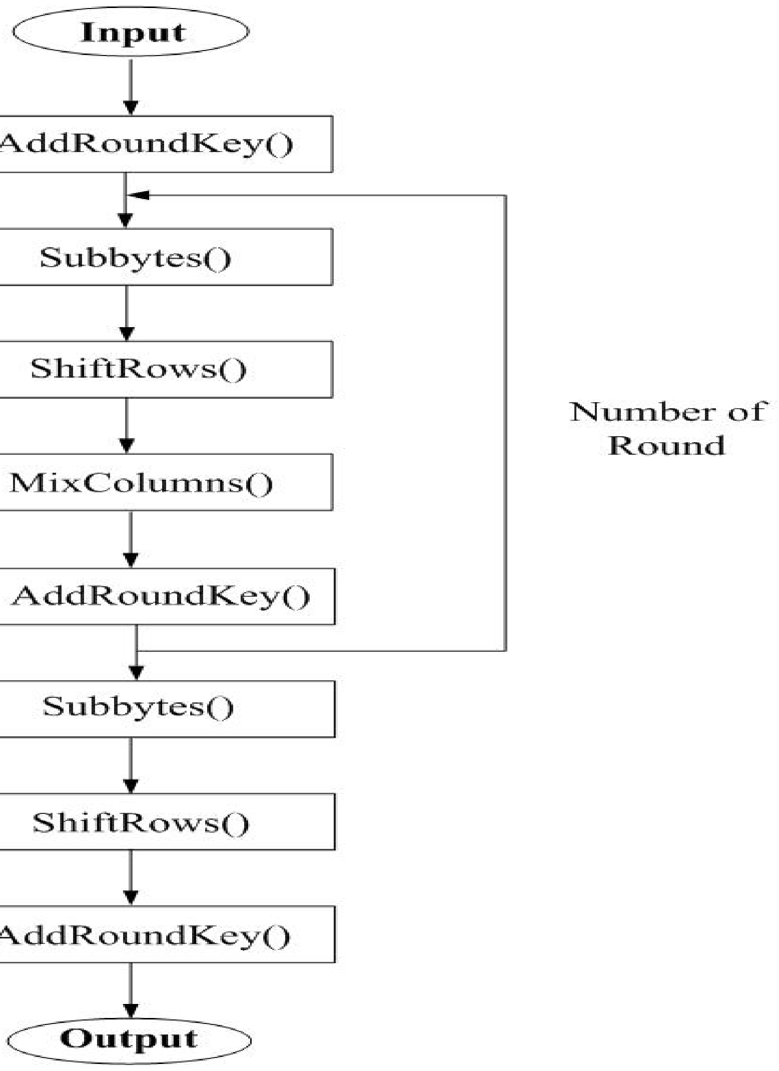


Figure 2.3 AES Encryption

AddRoundKey operation combines a 128-bit state with a 128-bit round key using exclusive-or operation. The operation of this stage is described in Figure 2.4, where the input state is A and the output state is B. State B is obtained by exclusive-oring corresponding bytes in the state A and the round key K that are derived from the cipher key using a key expansion routine known as Rijndael's key schedule.

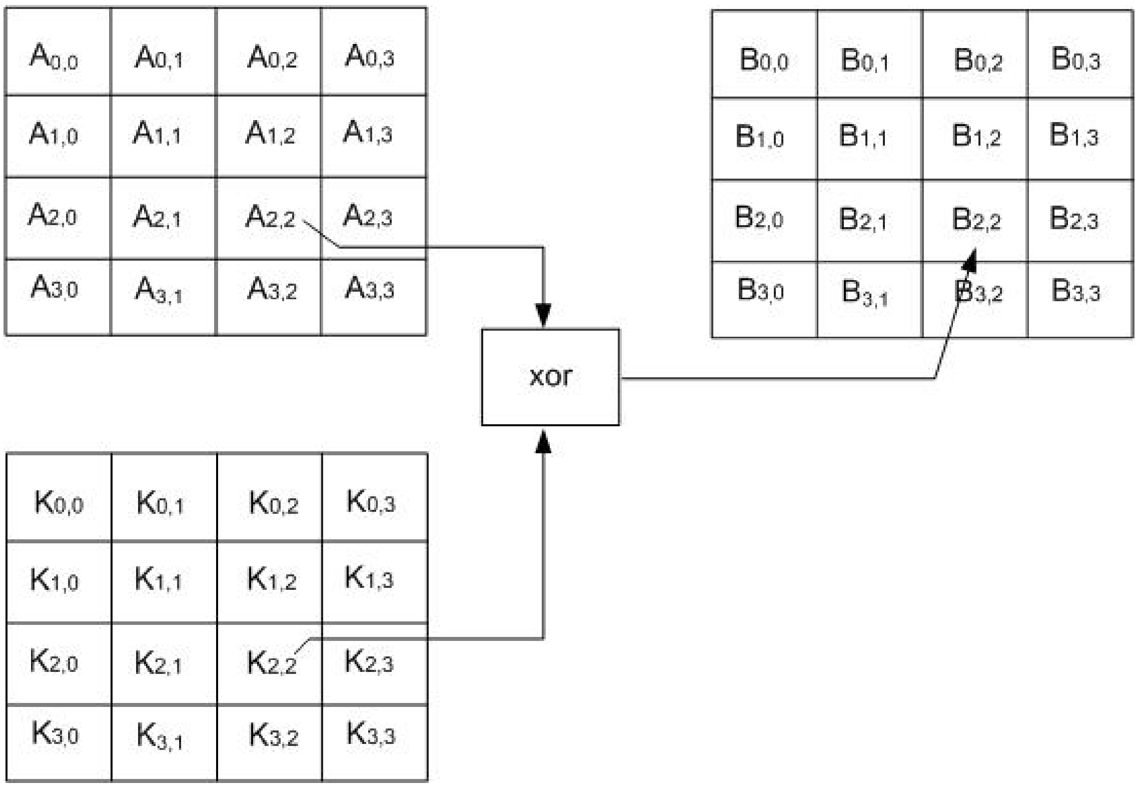


Figure 2.4 AddRoundKey Operation

SubBytes is an operation that substitutes each byte of a state. It is non-linear and sometimes the transformation is called S-Box. It is often implemented as a look-up table. Figure 2.5 illustrates the effect on the state.

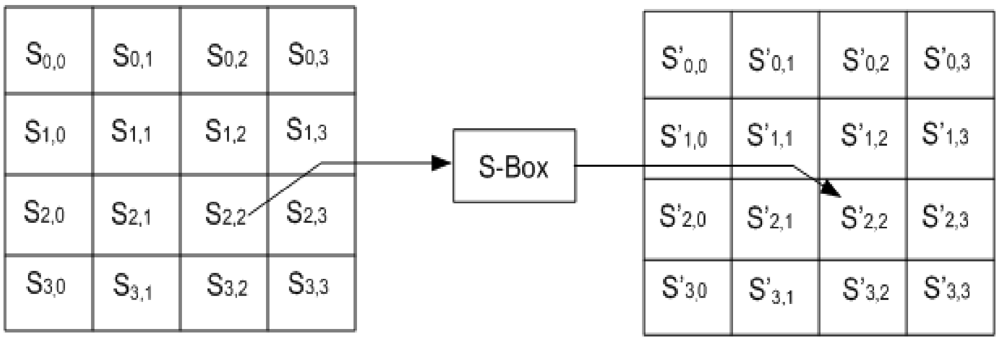
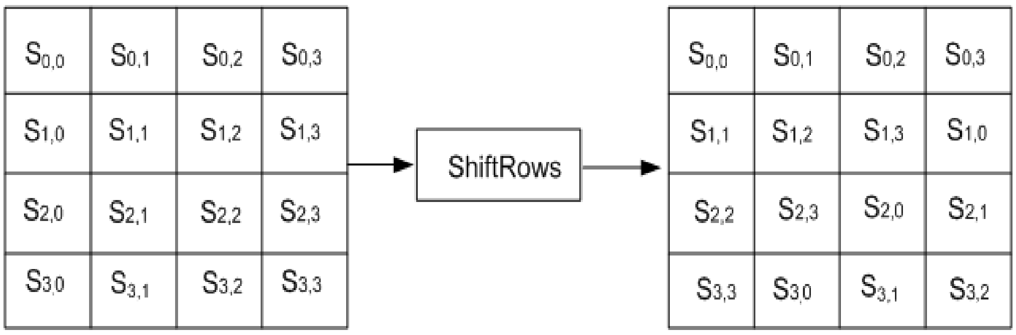


Figure 2.5 S-Box Operation

ShiftRows is a cyclic shift operation of the bytes in the rows. The actual value of the offset is equal to the row index. For example, the first row is not rotated but the last row is rotated three bytes to the left. The impact to the state is illustrated in Figure 2.6.



$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
ShiftRows
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,0}$
$S_{2,2}$ $S_{2,3}$ $S_{2,0}$ $S_{2,1}$
$S_{3,3}$ $S_{3,0}$ $S_{3,1}$ $S_{3,2}$

Figure 2.6 ShiftRows Operation

MixColumns transforms columns of the input state using matrix multiplication. Following matrix is used in AES-128 algorithm and each column in the output state is obtained by multiplying the matrix with the corresponding column in the original state. The computation is performed such that each column is treated as a polynomial over GF (2^8) and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $3x^3 + x^2 + x + 2$.

$$\begin{pmatrix} S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \\ S'_{4,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{1,c} \\ S_{2,c} \\ S_{3,c} \\ S_{4,c} \end{pmatrix}$$

Because all the operations in the algorithm are invertible, decryption is possible. The decryption process is the reverse of the encryption also consisting of four major operations, which are

AddRoundKey, InvShiftRows, InvSubbytes and InvMixColumns respectively. The decryption algorithm is shown as Figure 2.7.

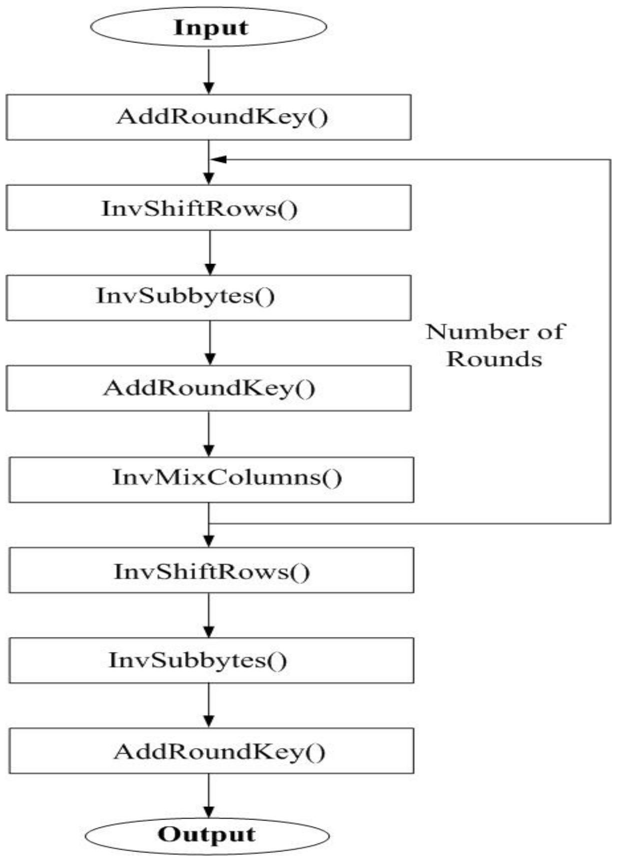


Figure 2.7 AES Decryption

2.3.2 Tiny Encryption Algorithm

Tiny Encryption Algorithm (TEA) family algorithms belong to symmetric key category in cryptography and became of interest in RFID security solutions during recent years. TEA was proposed by Roger Needham and David Wheeler in 1994 [27]. Their work was to create a Feistel itineration based routine using a large number of rounds rather than complicated and memory-consuming approaches.

The Tiny Encryption Algorithm is believed to be one of the fastest and most efficient cryptographic algorithms. Tiny Encryption Algorithm is a Feistel cipher that uses only XOR, ADD and SHIFT operations to provide Shannon's properties of diffusion and confusion necessary for a secure block cipher without the need for Permutation box (P-box) and Substitution box (S-box). S-box is a non-linear operation, and is typically keyless. It can be applied in an intermediate stage of an algorithm. In cryptography, P-box is used to perform bit-shuffling across S-boxes inputs. Tiny Encryption Algorithm operates on a 64-bit data block using a 128-bit key. The C language implementation of encoding and decoding could be expressed as follows:

```
void code(long* v, long* k)
{
     unsigned long y=v[0],z=v[1], sum=0, /* set up */
     delta=0x9e3779b9, /* a key schedule constant */
     n=32 ;
     while (n-->0)
     { /* basic cycle start */
          sum += delta ;
          y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
          z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
     } /* end cycle */
     v[0]=y ;
     v[1]=z ;
}

void decode(long* v,long* k)
{
     unsigned long n=32, sum, y=v[0], z=v[1],
     delta=0x9e3779b9 ;
     sum=delta<<5 ;
     while (n-->0)
     {/* start cycle */
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
          sum-=delta ;
     }/* end cycle */
     v[0]=y ;
     v[1]=z ;
}
```

The code and decode routines use 32 bit words in operations. The inputs of the routines are two words for data, which is 64 bits totally. There are four words for keys, thus the total size of the key is 128-bits. The routine uses XOR, ADD and bit shift operations to provide non-linearity for encryption. One cycle of the encryption is shown in Figure 2.8.

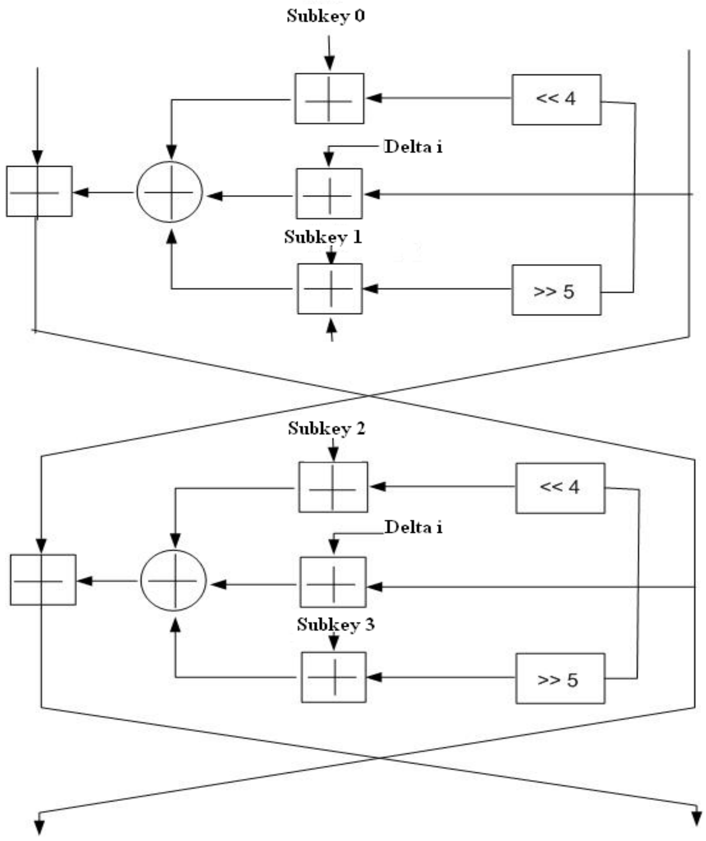


Figure 2.8 One Cycle of Tiny Encryption Algorithm

2.3.3 Extended Tiny Encryption Algorithm

Although TEA algorithm is compact to implement for different platforms, it is reported to suffer from related key attacks and its actual key size is reduced to 126-bits [28]. In order to correct the weakness, Roger Needham and David Wheeler proposed the eXtended Tiny Encryption Algorithm (XTEA) [29]. As compared to TEA, the key scheduling has been changed and its key material is introduced more slowly. Despite the adjustment that was made on the key operation, XTEA remains as compact as TEA. The reference code in C programming language is shown as follows:

```
tean( long * v, long * k, long N)
{
     unsigned long y=v[0], z=v[1], DELTA=0x9e3779b9 ;
     if (N>0)
     {
          /* coding */
          unsigned long limit=DELTA*N, sum=0 ;
          while (sum!=limit)
          y+= (z<<4 ^ z>>5) + z ^ sum + k[sum&3],
          sum+=DELTA,
          z+= (y<<4 ^ y>>5) + y ^ sum + k[sum>>11 &3] ;
     }
     else
     {
          /* decoding */
          unsigned long sum=DELTA*(-N) ;
          while (sum)
          z-= (y<<4 ^ y>>5) + y ^ sum + k[sum>>11 &3],
          sum-=DELTA,
          y-= (z<<4 ^ z>>5) + z ^ sum + k[sum&3] ;
     }
     v[0]=y, v[1]=z ;
     return;
}
```

Like TEA algorithm, the block size is two words and the key size remains four words. XTEA algorithm completes encryption and decryption in 32 Festiel cycles. The only disadvantage is that complete change diffusion is delayed by one cycle, but there is no need to change the number of cycles from 32 to 33. Figure 2.9 shows the structure of one cycle of encryption.

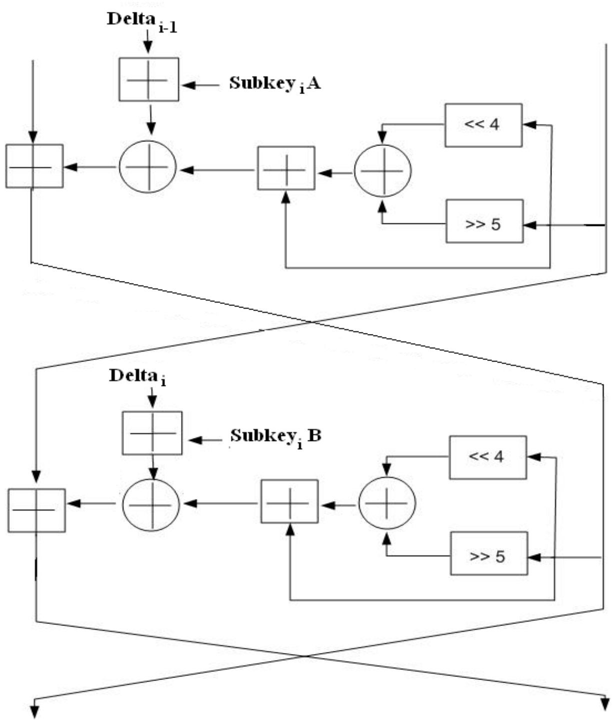


Figure 2.9 One Cycle of eXtended Tiny Encryption Algorithm

Under certain weak key assumption, a related-key rectangle attack on the first 36 rounds of XTEA has been reported [30]. The time complexity of the attack is about $2^{104.33}$, which is faster than exhaustive key search among the $2^{110.67}$ weak keys. The success probability of the attack is about 80%. It is claimed to be the best known attack on XTEA algorithm. Since XTEA uses 64 rounds, it is still considered to be a very secure cipher. TEA and XTEA are not subject to any patents and are free to use in security applications.

2.4 Related Works of RFID Protocols

An authentication protocol is a safe way to verify if a tag is genuine or not. During recent years, many researchers proposed different solutions. Sarma et al.'s early work defined a so-called Hash-based Access Control (HAC) scheme [31]. The scheme is presented in Figure 2.10.

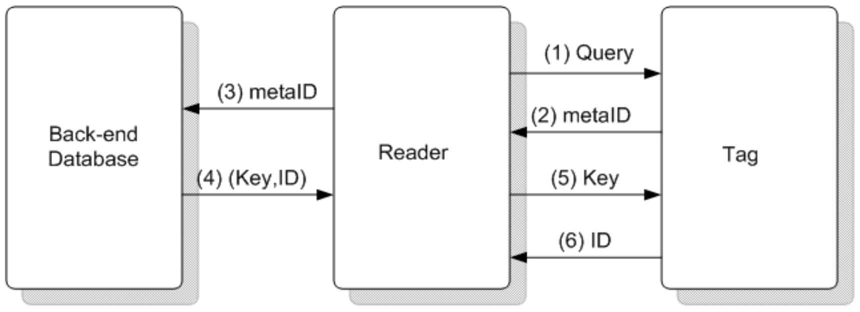


Figure 2.10 Hash-Locking Protocol

This scheme assumes that tag is only equipped with a hash function. When being queried, the tag sends out the hash of a random key as its metaID instead of revealing the tag ID directly. The reader forwards it to the back-end server to search the key and unlock the tag. If the key is found, it is passed to the tag. The tag hashes the key and compares it to the stored metaID. If the reader has sent the right key to the tag, the hash value of the key will match the metaID. If the values match, the tag enters un-locked state and sends the ID to reader for authentication. The protocol relies on a back-end server for authentication. However, the key, metaID and ID could be obtained by an eavesdropping adversary and therefore the tag can be cloned easily.

To overcome the weakness in Hash-Locking protocol, Sarma et al. also proposed another so-called Randomized Hash-locking Protocol [31]. The scheme is presented in Figure 2.11.

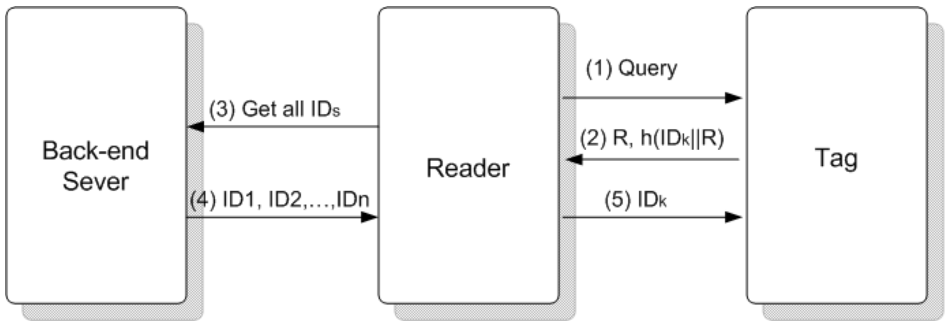


Figure 2.11 Randomized Hash-Locking Protocol

It is assumed that the tag is equipped with a hash function and a random number generator. The tag responds to the reader queries by generating a random number R, then computing the hash value h(IDk || R). Both of them are sent back to the reader. The reader performs a brute-force search for the tag ID by calculating the hash value h(IDk || R) for each tag until it finds one value that matches the tag response. Although this protocol works well with a few tags, it is not practical for retail application where a large number of tags are deployed.

Ohkubo et al. proposed a Hash-chain protocol to solve the secrecy and forward security issue [32]. The message exchange process of the protocol is shown in Figure 2.12.

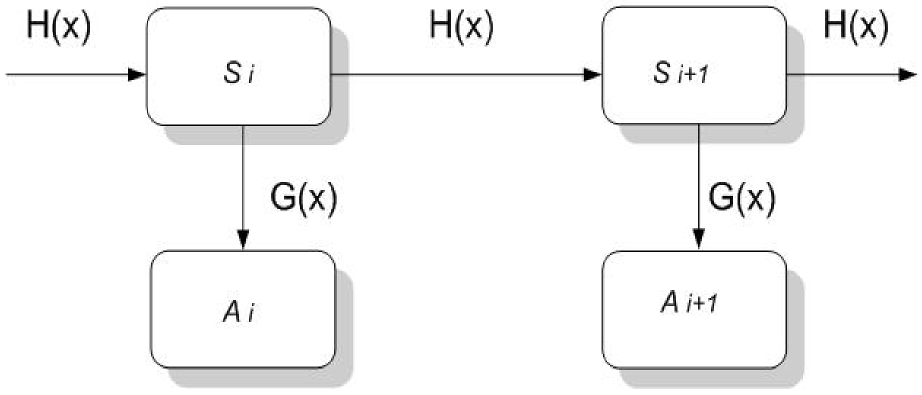


Figure 2.12 Hash-chain Protocol

In this protocol, the tag ID is never revealed in plaintext. It is assumed that the tag and back-end server share an initial secret of S1 and it is equipped with two hash functions, H(x) and G(x). In the i-th protocol session, when the tag is queried, it sends out a message Ai = G(Si) to the reader. The reader forwards it to the back-end server for authentication. The back-end server maintains a list of pairs (ID, S). When receiving Ai, it checks all the existing tag secrets and computes the hash value for each of them using function G(x). If a matched value is found, the tag ID is sent to the reader. In every new session, the tag and the back-end database update the secret using H(x) function to get $S_{i+1} = H(S_i)$. The main weakness of the protocol is that the back-end is much overloaded since it has to perform brute force search for the tag ID. Furthermore, if a malicious attacker keeps sending query to the tag, the tag will continuously update its own secret, which will make the tag desynchronized with the server. Furthermore, denial-of-service attack is another possible threat for this protocol.

An improved security solution for RFID security is proposed by Wang et al., which corrects the weaknesses found in hash lock scheme [33]. Their scheme is shown in Figure 2.13.

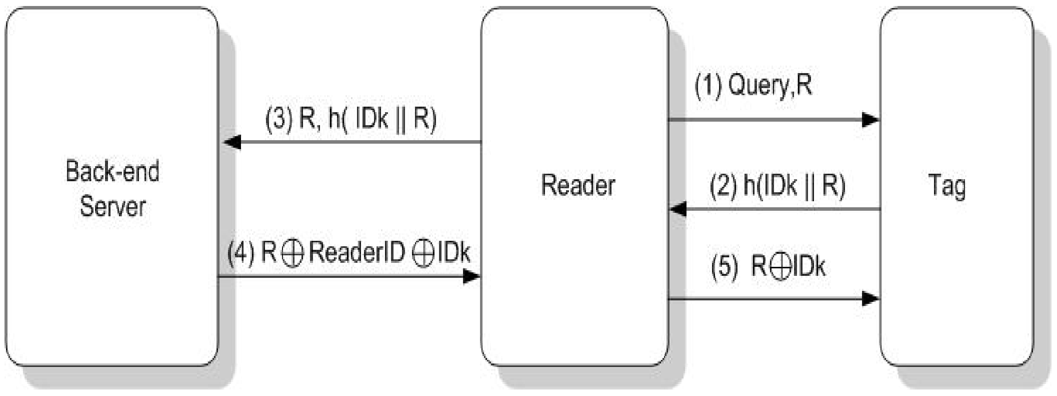


Figure 2.13 Improved Security Solution

In this protocol, the reader first generates a random number and sends it to the tag when querying it. The tag calculates the hash value h(IDk || R) and sends it back to the reader in response, where IDk is the tag ID. The back-end server search for an IDt where h(IDt || R) = h(IDk || R). The back-end server does not assume that the reader is legitimate and therefore generates a message to be sent to the reader by exclusive-oring the correct reader ID, the random number R and the tag ID. The reader has to generate a reply by exclusive-oring its own ID to get (R $\oplus$ IDk). The tag receives this message and exclusive-ors it to derive IDk. If it matches with its own ID number, the reader is authenticated. Nevertheless, the protocol shows a weakness in the last step.

An attacker can get the tag ID by exclusive-oring this message with R, which can be eavesdropped in the first step.

Toiruul and Lee proposed an advanced mutual authentication protocol to address three major problems: forgery of the tags, unwanted tracking of the tags, and unauthorized access to tag memory [34]. Their objective is to provide reader authentication to a tag, exhibit forgery resistance against a simple copy, and prevent the counterfeiting of RFID tags. We use notation shown in Table 2.1 to illustrate this protocol.

Table 2.1 Notation used in Toiruul and Lee's Protocol

Symbols	Notations
T	RF tag, or transponder
R	RF tag reader, or transceiver
B	Back-end server, which has a database
k1 , k2	Random secret keys, shared between T and B
k	Cryptographic key, shared between T and B
IDk	Unique identification number of T, shared between T and B
Ek	AES cipher using key k
*	Operation or the value in the real T
$\oplus$	Exclusive-or

It is assumed that the tag has an AES encryption cryptographic hardware and a memory space to store the identification number IDk and two secrets, k1 and k2. Toiruul and Lee's protocol is presented in Figure 2.14. Initially each T is given two fresh random secrets, k1 and k2, and a unique identification, IDk. The database of B also stores them as the shared secrets and manages a record pair for each tag consisting of (IDk, TagID). T has an AES-128 encryption circuit while both T and B have a 128-bit cipher key k. In the first challenging step, the reader R applies a collision protocol to singularize T out of many tags. In their work, the binary tree walking protocol was suggested as an example [35]. The Reader, R receives Ek(k1 $\oplus$ k2) from the back-end server, B. Then R sends Ek(k1 ⊕ k2) to query T. Ek(k1 ⊕ k2) is used to authenticate the validity of R.

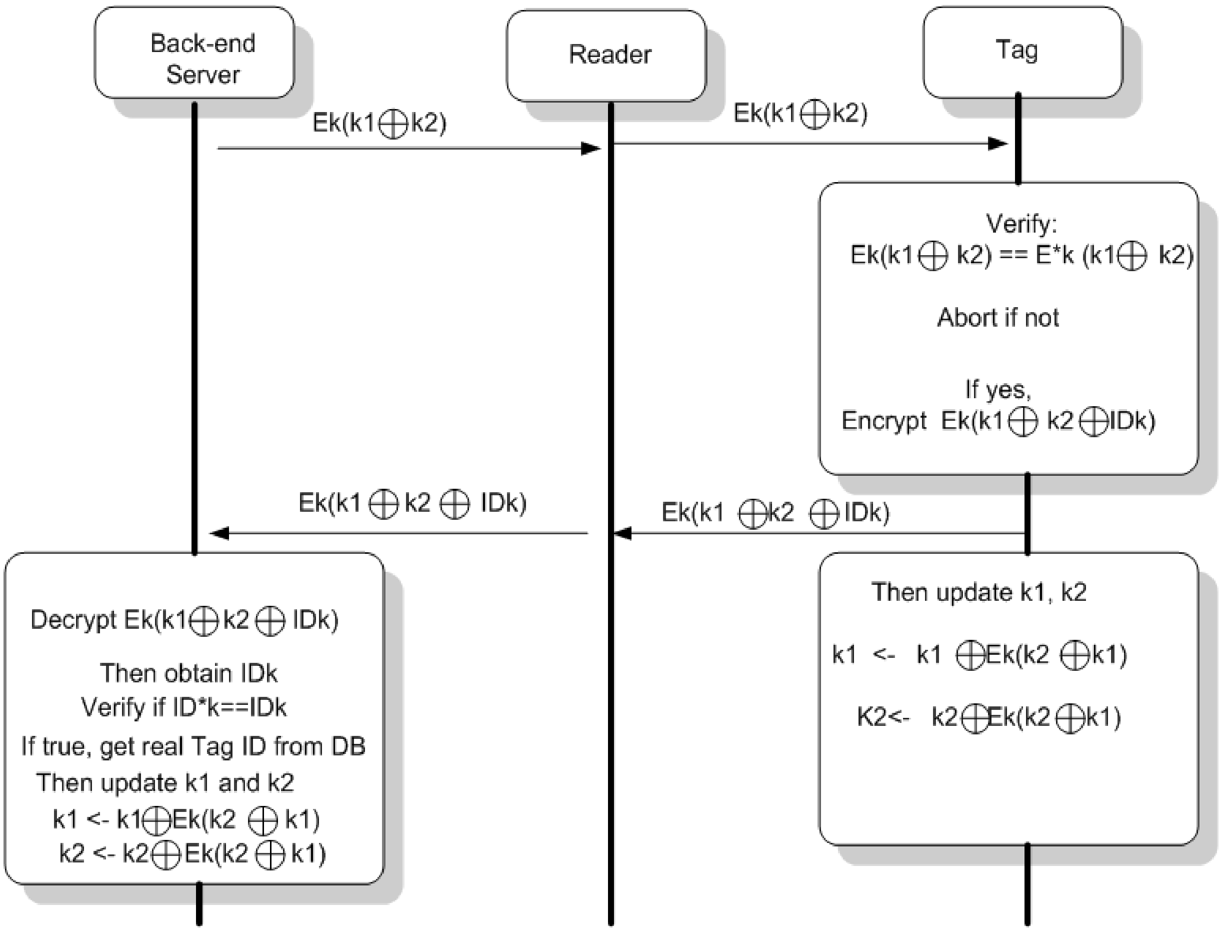


Figure 2.14 Toiruul and Lee's Protocol [34]

In the second stage of authentication, the tag authenticates the reader's validity. When being queried, T generates E*k (k1 ⊕ k2) and checks whether the received message Ek(k1 ⊕ k2) equals to E*k(k1 ⊕ k2) or not. If they do not match, R is not authenticated and T will keep silent. If they match, T authenticates R and sends out Ek(k1 ⊕ k2 ⊕ IDk), which is the encryption of AES-128, to the reader. Then T updates its secrets k1 = k1 ⊕ Ek(k2 ⊕ k1) and k2 = k2 ⊕ Ek(k2 ⊕ k1).

In the third authentication stage, the reader authenticates the tag. R simply forwards Ek(k1 $\oplus$ k2 $\oplus$ IDk) to B and B decrypts it using cipher key k. IDk is obtained by exclusive-oring the result with k1 and k2. Then B verifies whether IDk is valid by comparing the IDk with ID*k, which is the real tag ID. The secrets, k1 and k2, and the cipher key, k are shared only between B and T. Therefore, B can detect an illegal T and discards the message. When T is authenticated, B retrieves the records corresponding to IDk and sends the real TagID to R. Then it update its secrets k1 and k2 in the same way, k1 = k1 $\oplus$ Ek(k2 $\oplus$ k1) and k2 = k2 $\oplus$ Ek(k2 $\oplus$ k1). The protocol is claimed to withstand active attacks such as the man-in-the-middle attack, the replay attack, the eavesdropping attack, and the unwanted tracking of customers.

Inspired by the efficient implementation of AES by Feldhofer et al. [36, 37], Kim et al. proposed a lightweight RFID authentication protocol using step by step symmetric key change [38]. To illustrate the protocol, we use notations in Table 2.2. In a RFID system which may consist of a server, a reader and one or multiple tags, it is assumed that the tag and back-end server can perform AES encryption and decryption operation. They are also equipped with pseudo-random number generators that can generate random numbers.

Table 2.2 Notation used in Kim et al.'s Protocol

Symbols	Notations
E( )	Block cryptogram
M	The mask bit of the random number
PRNG	Pseudo-random number generator
Rtag	The random number generated in the tag
Rreader	The random number generated in the reader
Rserver	The random number generated in the server
||	Concatenation
K0	Initial symmetric key
K1, K2, K3	Changed symmetric key
$\oplus$	Exclusive-or
ID	The unique identifier of the tag

The operation of this protocol can be divided into initialization stage and authentication stage. In the initialization stage, the tag and the server are assigned the same symmetric key (K0) in advance. The tag and the reader share the same mask bit (M) for masking the generated random number. The operation of the authentication stage can be illustrated in Figure 2.15.

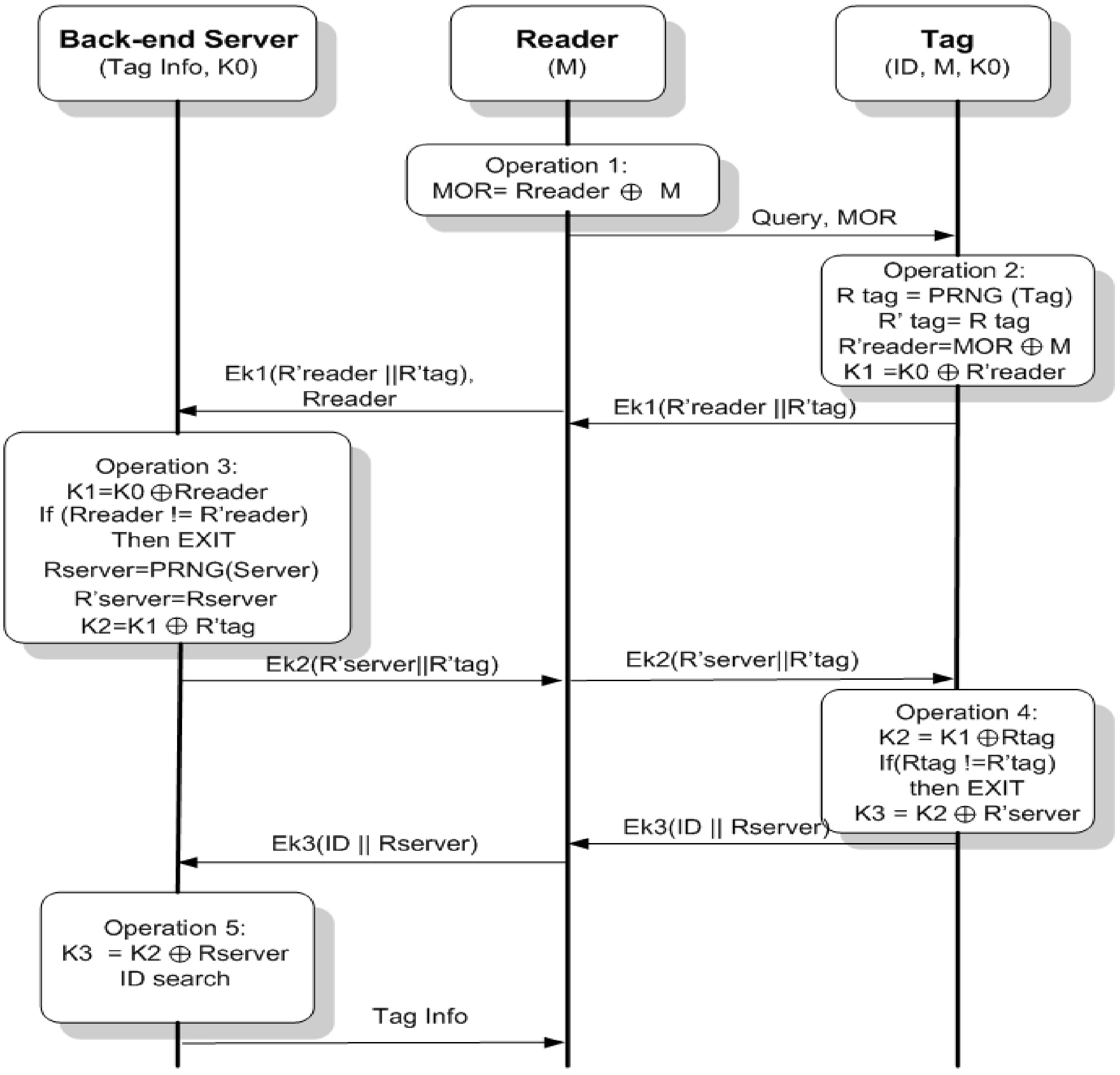


Figure 2.15 Kim et al.'s Authentication Protocol

The protocol consists of five operations. In operation one, the reader generates a random number and exclusive-ors it with M to obtain a mask variable called MOR. Then the reader sends MOR with the query to the tag.

In operation two, the tag uses M to exclusive-or with MOR to obtain the random number generated by the reader. Then it is exclusive-ored with K0 to generate the new key K1 which is used to encrypt R'reader || R'tag. The encrypted message is sent to the reader and then is passed to the back-end server.

In the third operation, the server obtains Rreader transmitted by the reader. Then K1 is obtained by exclusive-oring it with K0. The server decrypts the message using K1 to get R'reader and R'tag. The authentication is performed by comparing R'reader with Rreader delivered from the reader. If the comparison fails, the operation exits. If it is successful, the back-end server generates the key K2, which is used to encrypt R'server || R'tag. The encrypted message is sent to the reader and is then passed to the tag.

In operation four, the tag obtains K2 by exclusive-oring K1 with Rtag. The message from the server is decrypted using K2, and R'tag and R'server is obtained. Comparing R'tag and tag's Rtag, the tag authenticates the reader. If the operation is successful, the tag generate K3 = K2 $\oplus$ R'server. Then the tag encrypts the ID || R'server using K3, and sends it to the server.

In operation five, K3 is computed by the back-end server using its own K2 and Rserver. It is used to decrypt the message from the tag to obtain ID and R'server. If the R'server matches Rserver of its own, the response from the tag is confirmed. By searching the ID, the back-end finds the tag info and sends it to the reader.

Their work is to solve the problem of using fixed key in authentication to prevent eavesdropping, replay attack and location tracking security issue. In the mutual authentication process, random numbers are generated by the server, reader and tag to make the messages different in each round.

Chapter 3

Novel RFID Authentication Key Management Protocol

Some previous works used hash functions because symmetric key algorithms are considered to be too resource consuming at early development of RFID, while other recent works tried to achieve more security level by using symmetric key algorithm. The objective of our proposal is to achieve mutual authentication using symmetric key algorithm. The key should be dynamically updated and the process should be initiated from the back-end server in a synchronized way to avoid possible vulnerabilities and to make tags as lightweight as possible.

3.1 System Assumptions

Previous research has shown that XTEA has great potential to be utilized in new RFID authentication protocols [42, 49], because of its low power consumption, high speed and high security level characteristics. In order to solve security and privacy issues in RFID systems, we are proposing a novel and robust authentication protocol based on XTEA algorithm. In this protocol, we assume that a tag has a 64-bit non-volatile tag ID number and 128-bit key set, which can be dynamically updated. It is also assumed that the tag has the encryption and decryption capability using XTEA algorithm. One key set consists of four 32-bit sub-keys. Each tag and the back-end server share a random secret key set at initialization. We also assume that the links between back-end server and readers are secure against attacks. This assumption is based on the fact that advanced security mechanisms can be deployed in networked RFID infrastructures.

3.2 Robust RFID Authentication Key Management Protocol Design

Our proposed protocol is summarized in Figure 3.1.

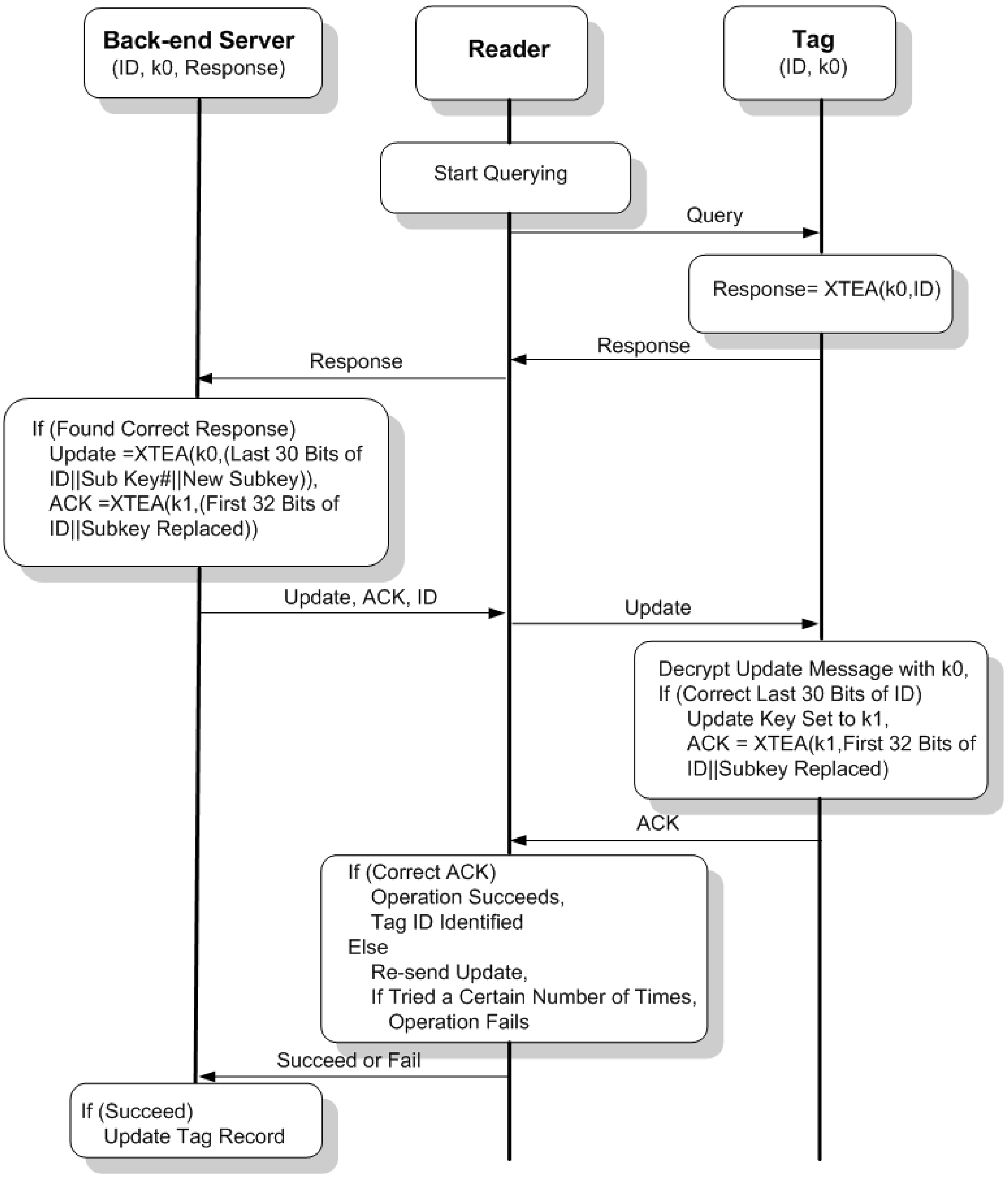

Back-end Server
(ID, k0, Response)
Reader
Tag
(ID, k0)
Start Querying
Query
Response= XTEA(k0,ID)
Response
Response
If (Found Correct Response)
Update =XTEA(k0,(Last 30 Bits of ID||Sub Key#||New Subkey)),
ACK =XTEA(k1,(First 32 Bits of ID||Subkey Replaced))
Update, ACK, ID
Update
Decrypt Update Message with k0,
If (Correct Last 30 Bits of ID)
Update Key Set to k1,
ACK = XTEA(k1,First 32 Bits of ID||Subkey Replaced)
ACK
If (Correct ACK)
Operation Succeeds,
Tag ID Identified
Else
Re-send Update,
If Tried a Certain Number of Times,
Operation Fails
Succeed or Fail
If (Succeed)
Update Tag Record

Figure 3.1 Novel XTEA Based RFID Authentication Key Management Protocol

The protocol is described as follows.

1) The reader sends a query message to read the tag.

2) Once being queried, the tag transmits an encrypted tag ID. The encryption is carried using XTEA algorithm with current key set k0.

3) The reader forwards this message to the back-end server for authentication.

4) The back-end server checks the database of expected response of known tags to find the corresponding tag ID. Then the back-end server immediately generates a key update message and the corresponding acknowledgement message. The message format of key update message is shown in Figure 3.2. It consists of a field containing the last 30 bits of the tag ID, a 2-bit field indicating the number of the sub-key to be updated and a 32-bit field of new sub-key.

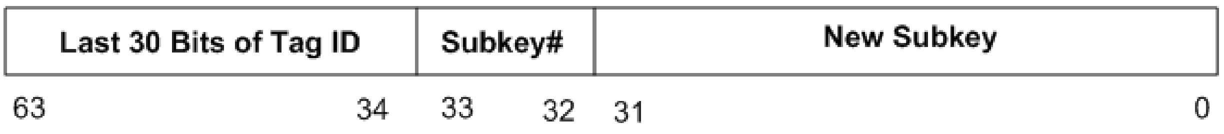


Figure 3.2 Key Update Message Format

The number of the sub-key and the new sub-key value are generated using random number generator, such as PUF. The total length of the message is 64 bits and it must be encrypted using the current key set k0. The encrypted message is also 64 bits in length, but the message content is unreadable for outsiders. The back-end server computes the tag response for the next query based on the new key set k1. Meanwhile, it backups the current response just in case the key update process fails for some reasons.

The back-end server also generates a key update acknowledgement message with the format shown in Figure 3.3 and it is expected to come from the tag after the key update process is successful. The message is also 64-bit and is encrypted with the next key set k1. It consists of the first 32 bits of the tag ID and 32 bits of the sub-key that is to be replaced.

The server sends the tag ID, the key update message and the key update acknowledgement to the reader.

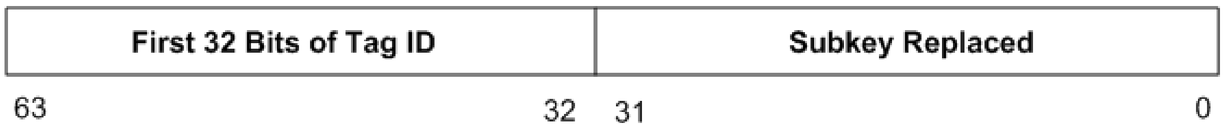


Figure 3.3 Key Update Acknowledgement Format

5) The reader forwards the key update message to the tag. The tag decrypts the message using the current key set k0 and it verifies the field of the last 30 bits of tag ID to see if it matches with its own. If there is a match, it will continue to extract the sub-key value and sub-key number to update the key set to k1. If not, it will continue to wait for other key update message for a time window.

6) The tag generates a key update acknowledgement in the format shown in Figure 3.3 and sends it back to the reader. It indicates that the key update operation has been successful.

7) The reader verifies if the key update acknowledgement is received. After receiving the acknowledgement message successfully, the reader terminates the authentication process.

8) If the reader fails to receive the acknowledgement it will re-send the key update message. The tag will respond with an acknowledgement again even if it has earlier sent it out. In the real implementation, the reader can re-send this key update message several times. If the reader is still not able to receive correct key update acknowledgement, it will send a warning to the back-end server that the tag might be a fake one. If the key-update process fails due to a fake tag, the tag response record in back-end server should be reverted to the previous value.

3.3 Security Analysis

The key update acknowledgement message in this scheme is important, because it is essential to prove that the key is updated properly by the tag. It also indicates that the tag is a genuine one, because only the tag that possesses the correct key set is able to generate a correct acknowledgement.

A good protocol should allow for mutual authentication of the reader and its tags. In our scheme, the tag verifies if the key update message is coming from a legitimate reader by decoding the message using current key set and verifying the ID field. Only the legitimate reader can pass this verification, since the reader can not generate it by itself and it can only obtain it from the trusted back-end server. In this protocol, the only information source of authentication is kept in the back-end server. The legitimacy of a reader should be verified by the back-end server before any message is sent to the reader, which means some security mechanisms should be implemented in the networked RFID infrastructure. This is beyond the discussion of this work.

In our protocol, the tag ID and the key are never disclosed in plain messages. Unless the current key set is disclosed, the messages being exchanged are unreadable to an adversary. The key length of XTEA is 128 bits and is secure enough against any brute force hacking. Since the key is updated dynamically, guessing the key becomes more difficult.

An adversary can not just record a tag response and replay it to pass the authentication process, because the key set is updated in every new session and therefore the correct response is also changed. As the correct key update acknowledgement is also different in every session, it is even harder for an adversary to counterfeit a genuine tag. Replaying a key update message to the tag is also not effective. Since the key set is changed, the tag will find the field of the first 32 bits of its tag ID does not match its own after decrypting a replayed key update message.

Denial-of-service attack is a bigger threat for RFID security system to deal with. It might desynchronize the tag and the trusted server. The acknowledgement message eliminates such a problem, because it notifies that a correct key update process is finished. In case the acknowledgement is blocked or is not received properly because of noisy environment, the reader will re-send it and the tag should response to it even if it has already updated its key and sent the acknowledgement. How many times the reader should repeat this message if it still cannot get the acknowledgement from a tag depends on the real application. On the tag side, it should stop responding with acknowledgement at some point if the key update message is received repeatedly. This is also a consideration that depends on the practical situation, and therefore we do not make any assumption in our protocol.

Prevention of tracking deals with the threat that privacy might be violated without actually identifying tag IDs. If a specific tag response can be associated with a particular item, this message could indicate the location of the item even the tag ID is not disclosed. In our protocol, the tag's response varies every time when it is queried, because the key is updated in every successful round of operation. This makes attackers unable to track the location of the tag.

Forward security means that if an attacker compromises a tag and knows all the information shared between the tag and its reader, it still cannot identify the previous responses of the tag. It is also impossible for the attackers to guess how the key has been updated previously even it manages to compromise a real tag and learns its secrets because how the key set is updated is only controlled by the trusted server. Therefore, the proposed protocol meets requirements of forward security.

Since the back-end server can make the key set update process truly random, even if a tag's current key is compromised, the tag's future key set is still not predictable. In a formal security model, potential attacker cannot always be at the proximity of a tag to send and receive messages [7]. If the tag is authenticated and the key set is updated while the attacker is not close to the tag, it is impossible for the attacker to get the new key set and understand the messages between the reader and tag again. Therefore, our protocol is also able to provide backward security.

Chapter 4

Modeling and Simulation

In this section, we demonstrate the workability and the functionality of our protocol using SystemC modeling technique. In order to verify the security level, we model different kinds of attacks using SystemC. SystemC is a C++ library addressing the modeling of both software and hardware [39]. It supports both high level system modeling and register transfer level (RTL) modeling. However, the major advantage of using this tool in this work is to model a RFID system at a higher abstraction level rather than the RTL level.

First of all, the SystemC modeling of our protocol is explained in details in section 4.1 including the state diagram of the reader and the tag module. Section 4.2 illustrates the cloning attack model and simulation results. Following that, modeling of replay attack is also demonstrated in section 4.3. Then we will discuss man-in-the-middle, denial-of-service and active scanning attack models and their simulation results in Section 4.4, 4.5 and 4.6 respectively.

4.1 Simulation of Protocol Using SystemC

A model of our proposed protocol is implemented using SystemC which is shown in Figure 4.1. This is a simplified model and the back-end server is integrated into the reader, because it is assumed that there is only one reader and one server in this simulation. This means that the reader here is integrated with the capability of saving and retrieving tag messages like a back-end server. Figure 4.2 shows the reader protocol in a state machine diagram. The tag is also implemented as a state machine as shown in Figure 4.3.

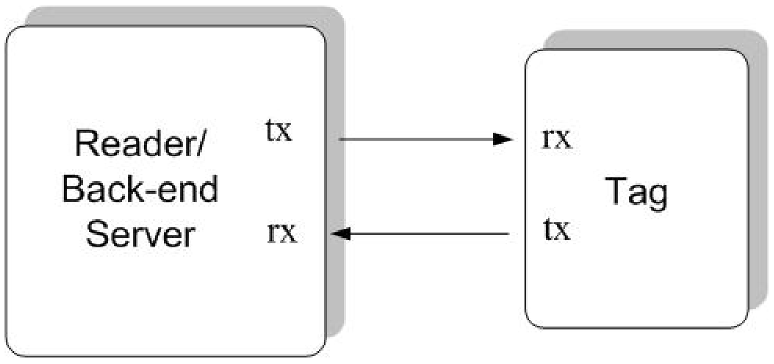


Figure 4.1 SystemC Model of Proposed Protocol

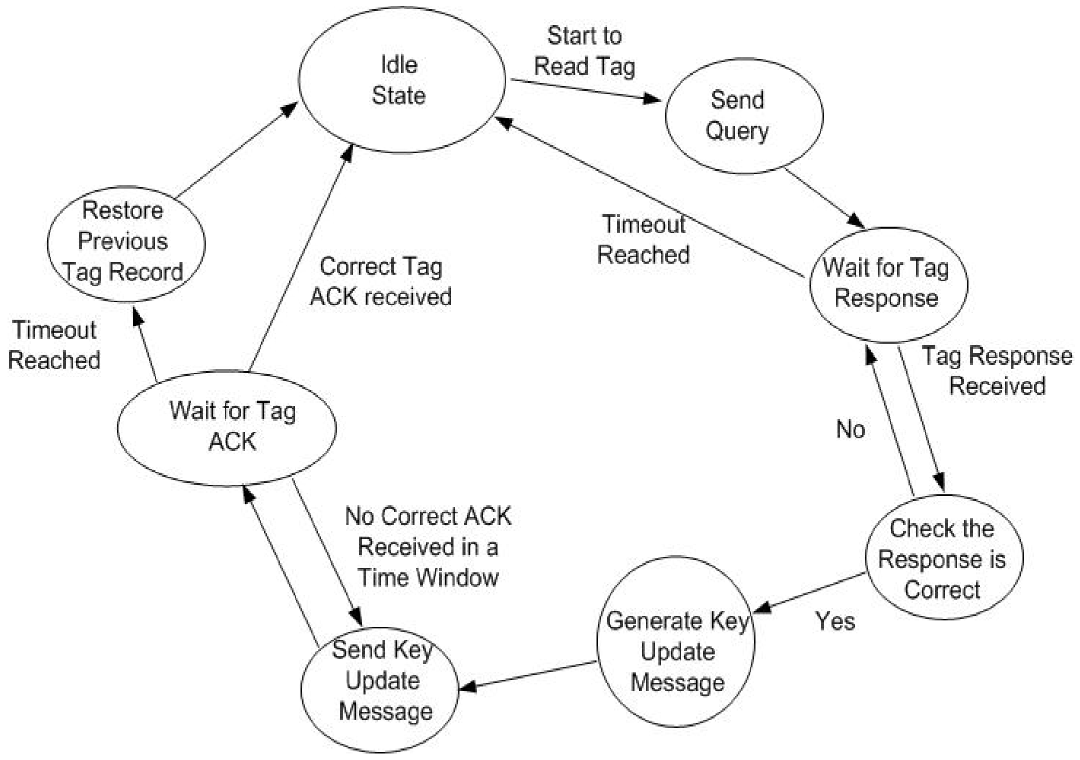


Figure 4.2 State Diagram of the Reader

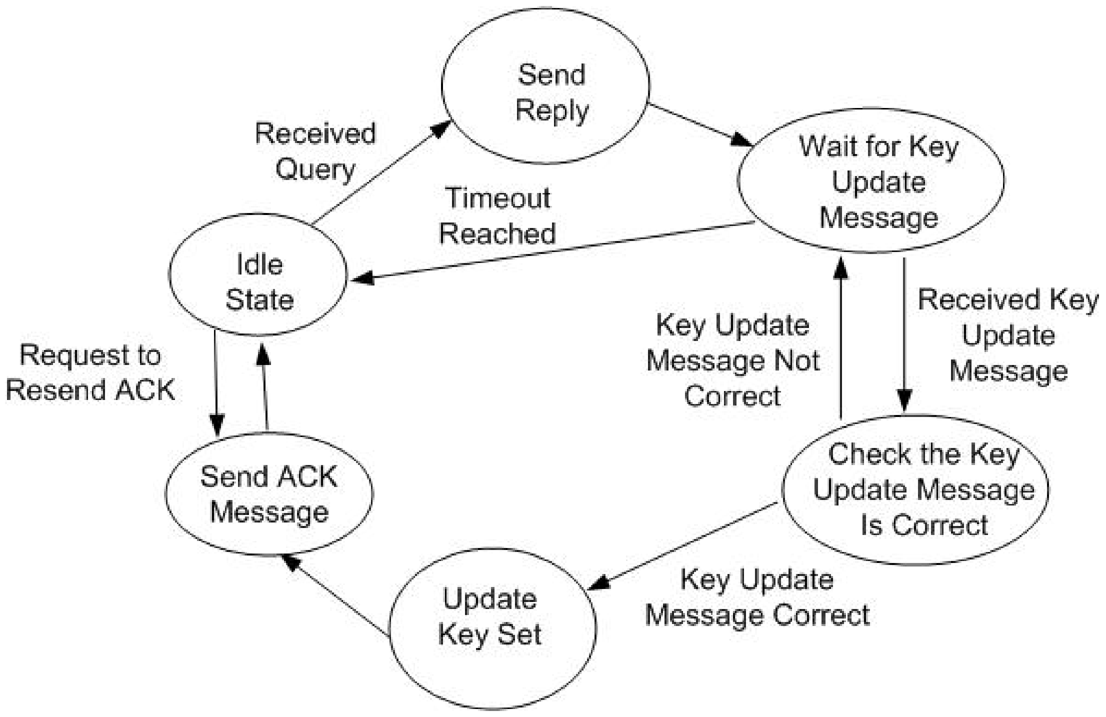
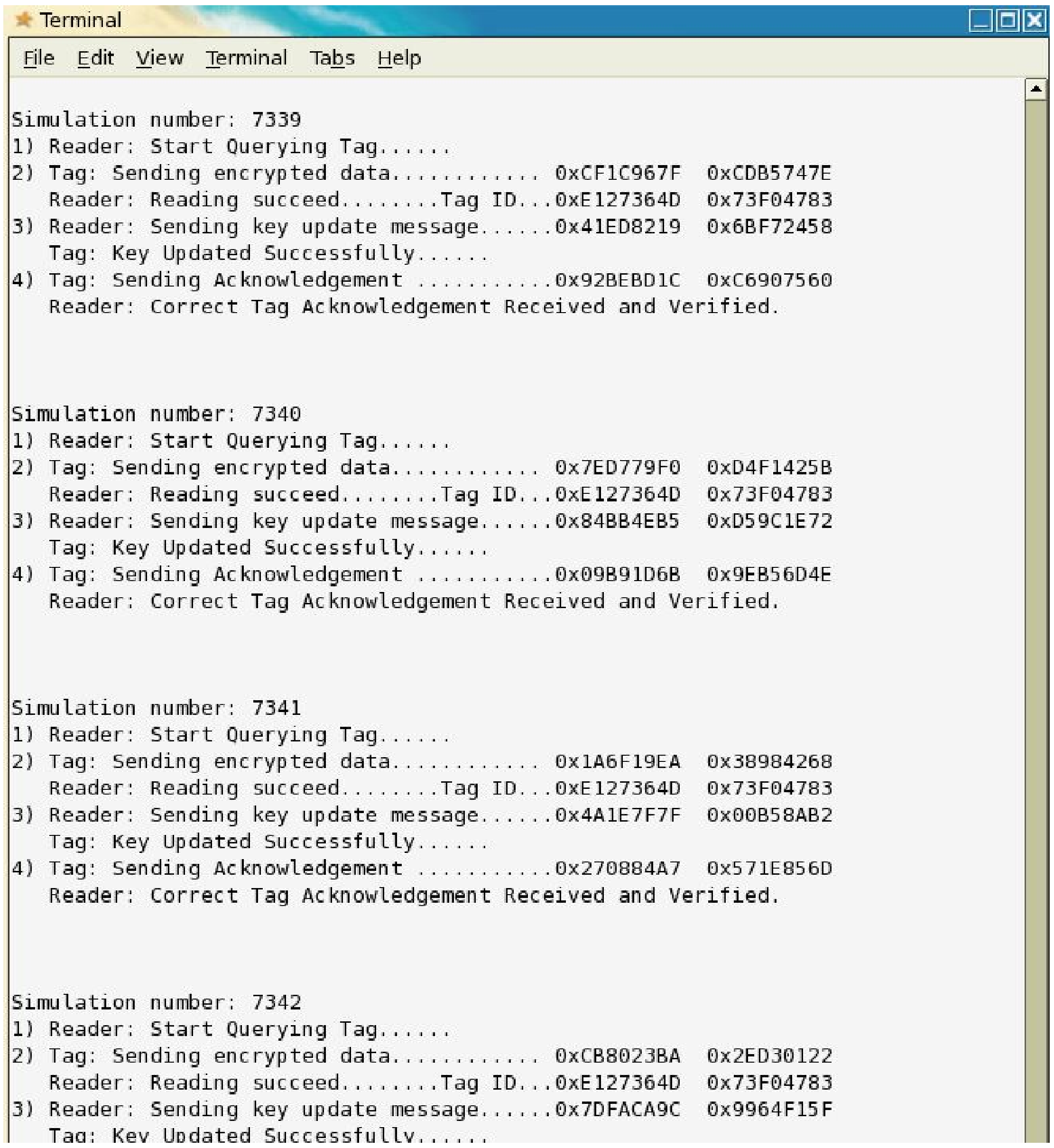

Send Reply
Received Query
Wait for Key Update Message
Timeout Reached
Idle State
Received Key Update Message
Key Update Message Not Correct
Request to Resend ACK
Check the Key Update Message Is Correct
Send ACK Message
Key Update Message Correct
Update Key Set

Figure 4.3 State Diagram of the Tag

The model is simulated on Sun Solaris workstation with SystemC 2.0.1 environment. The results are shown in Figure 4.4. It is observed that in every new session the tag replies with different response which is encrypted with a different key set. However, the reader can decrypt the response using the current key set. The reader generates a different key update message and the tag sends out different key update acknowledgement. The reader can identify the same tag ID in every session of the simulation, and interruption does not happen.



```
Simulation number: 7339
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0xCF1C967F  0xCDB5747E
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x41ED8219  0x6BF72458
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x92BEBD1C  0xC6907560
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 7340
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x7ED779F0  0xD4F1425B
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x84BB4EB5  0xD59C1E72
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x09B91D6B  0x9EB56D4E
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 7341
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x1A6F19EA  0x38984268
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x4A1E7F7F  0x00B58AB2
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x270884A7  0x571E856D
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 7342
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0xCB8023BA  0x2ED30122
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x7DFACA9C  0x9964F15F
   Tag: Key Updated Successfully......
```

Figure 4.4 SystemC Modeling and Simulation of the Proposed Protocol

4.2 Cloning Attack Model

As described in section 2.1, a security system suffers from cloning attack if its security features can be duplicated.

Figure 4.5 is the protocol model constructed to simulate this attack on protocol layer. A cloning attacker module is added in the model to examine whether the attacker can duplicate the tag by eavesdropping the tag's secrets.

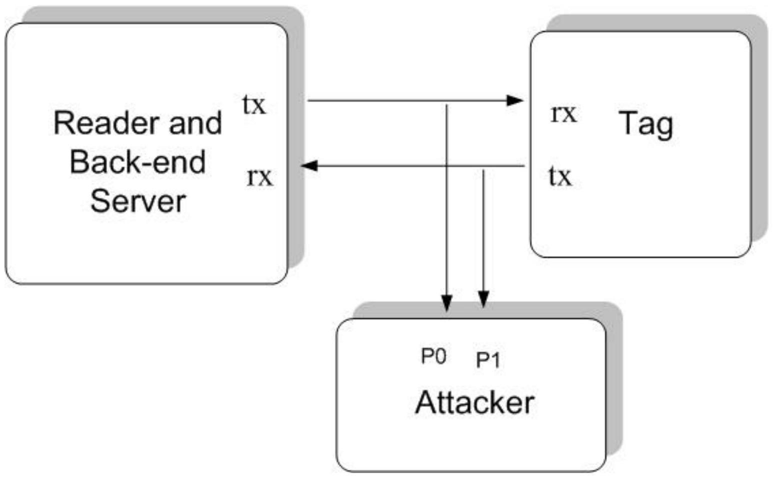
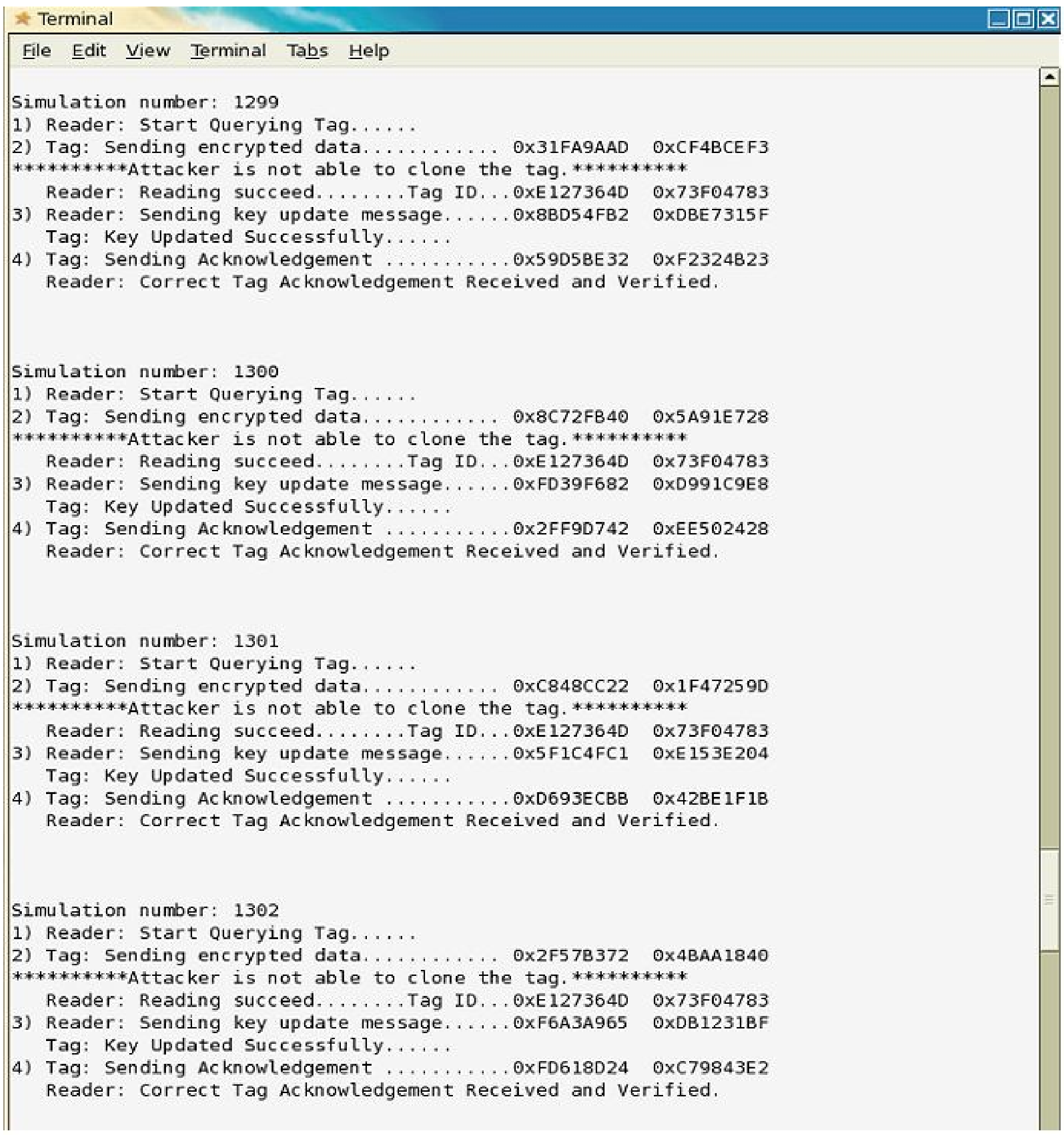


Figure 4.5 Cloning Attack Model

The simulation results are shown in Figure 4.6. When the attacker intercepts a message sent by the tag, the content is compared with the tag ID. If the content happens to be equal to the tag ID, the simulation code will indicate that the attacker is able to clone the tag. The simulation was run for one hour with a new session triggered in every 200 clock cycles, but we did not observe that the attack has been successful. Therefore, it verifies that the communication channel is secure against cloning attacks.

Since the key size of the XTEA algorithm is 128-bits, it is computationally infeasible for the adversary to launch a brute force attack. Therefore, the cipher is also resistant against cloning attack.



```
Simulation number: 1299
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x31FA9AAD  0xCF4BCEF3
**********Attacker is not able to clone the tag.**********
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x8BD54FB2  0xDBE7315F
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x59D5BE32  0xF2324B23
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 1300
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x8C72FB40  0x5A91E728
**********Attacker is not able to clone the tag.**********
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0xFD39F682  0xD991C9E8
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x2FF9D742  0xEE502428
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 1301
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0xC848CC22  0x1F47259D
**********Attacker is not able to clone the tag.**********
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x5F1C4FC1  0xE153E204
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xD693ECBB  0x42BE1F1B
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 1302
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x2F57B372  0x4BAA1840
**********Attacker is not able to clone the tag.**********
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0xF6A3A965  0xDB1231BF
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xFD618D24  0xC79843E2
   Reader: Correct Tag Acknowledgement Received and Verified.
```

Figure 4.6 Cloning Attack Model Simulation

4.3 Replay Attack Model

To prove our protocol is safe against replay attack, a replay attacker module is added to our original model. The attack model is shown in Figure 4.7.

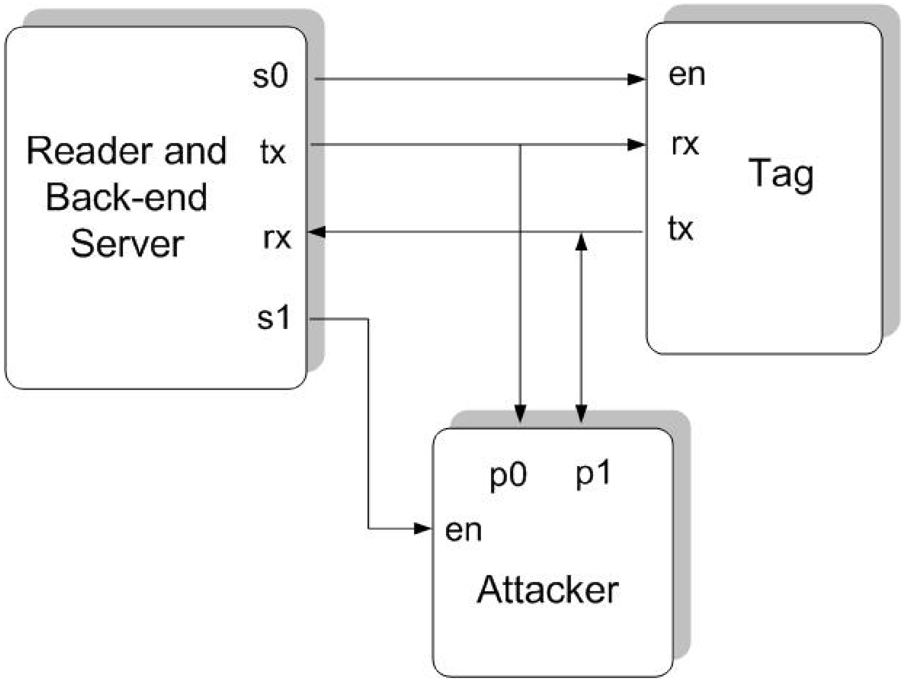
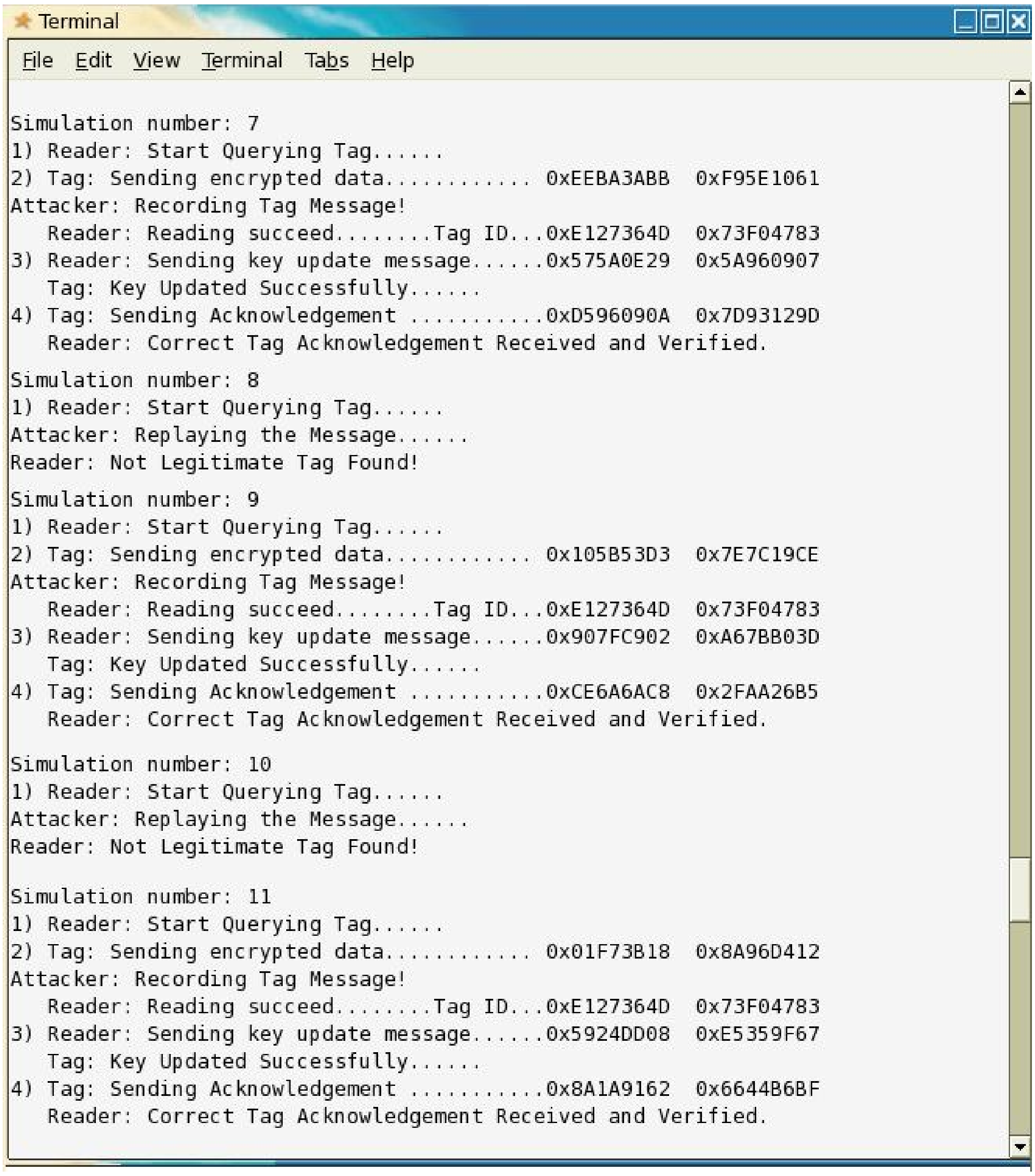


Figure 4.7 Replay Attack Model

In this model, an enable signal is added to both attacker and the tag modules, which allows the reader to query the tag and the attacker individually. The attacker records the tag response when it is not enabled. If the attacker is enabled while being queried, it replays the message it recorded. The simulations results of the replay attack are presented in Figure 4.8.



```
Simulation number: 7
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0xEEBA3ABB  0xF95E1061
Attacker: Recording Tag Message!
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x575A0E29  0x5A960907
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xD596090A  0x7D93129D
   Reader: Correct Tag Acknowledgement Received and Verified.
Simulation number: 8
1) Reader: Start Querying Tag......
Attacker: Replaying the Message......
Reader: Not Legitimate Tag Found!
Simulation number: 9
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x105B53D3  0x7E7C19CE
Attacker: Recording Tag Message!
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x907FC902  0xA67BB03D
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xCE6A6AC8  0x2FAA26B5
   Reader: Correct Tag Acknowledgement Received and Verified.
Simulation number: 10
1) Reader: Start Querying Tag......
Attacker: Replaying the Message......
Reader: Not Legitimate Tag Found!

Simulation number: 11
1) Reader: Start Querying Tag......
2) Tag: Sending encrypted data............ 0x01F73B18  0x8A96D412
Attacker: Recording Tag Message!
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x5924DD08  0xE5359F67
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x8A1A9162  0x6644B6BF
   Reader: Correct Tag Acknowledgement Received and Verified.
```

Figure 4.8 Replay Attack Model Simulation

The results show that when the reader tries to read the attacker, the attacker replays the same tag response message recorded in the last session. However, the reader finds that the attacker's reply is not correct. The attacker cannot decrypt the key update information because it does not know the correct key set. Therefore, in the real world application, a counterfeit tag using the replay attack technique can be detected by the system.

4.4 Man-in-the-Middle Attack Model

Man-in-the-middle is a form of the attack where the attacker hides itself between the reader and the tag. It acts like a reader to the tag and appears like a tag to the reader. In the previous model, it is proved that the attacker cannot simply clone a real tag or deceive the reader by replying with the same tag message. In this section, we provide an analysis to examine whether a man-in-the-middle attacker can be authenticated by the tag or not. The model of man-in-the-middle attack is shown in Figure 4.9.

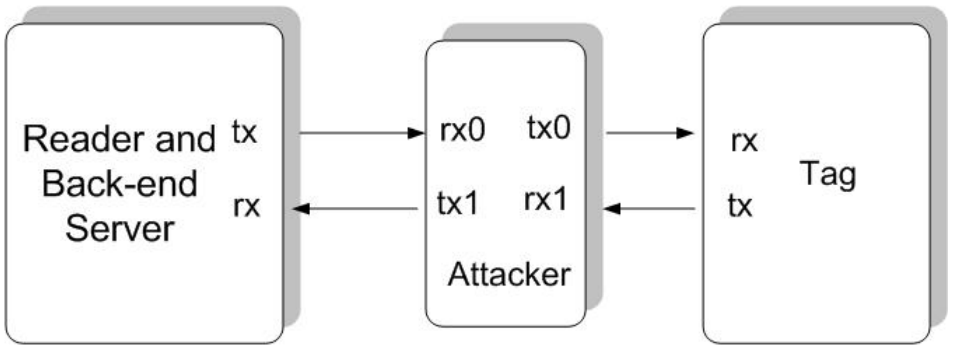
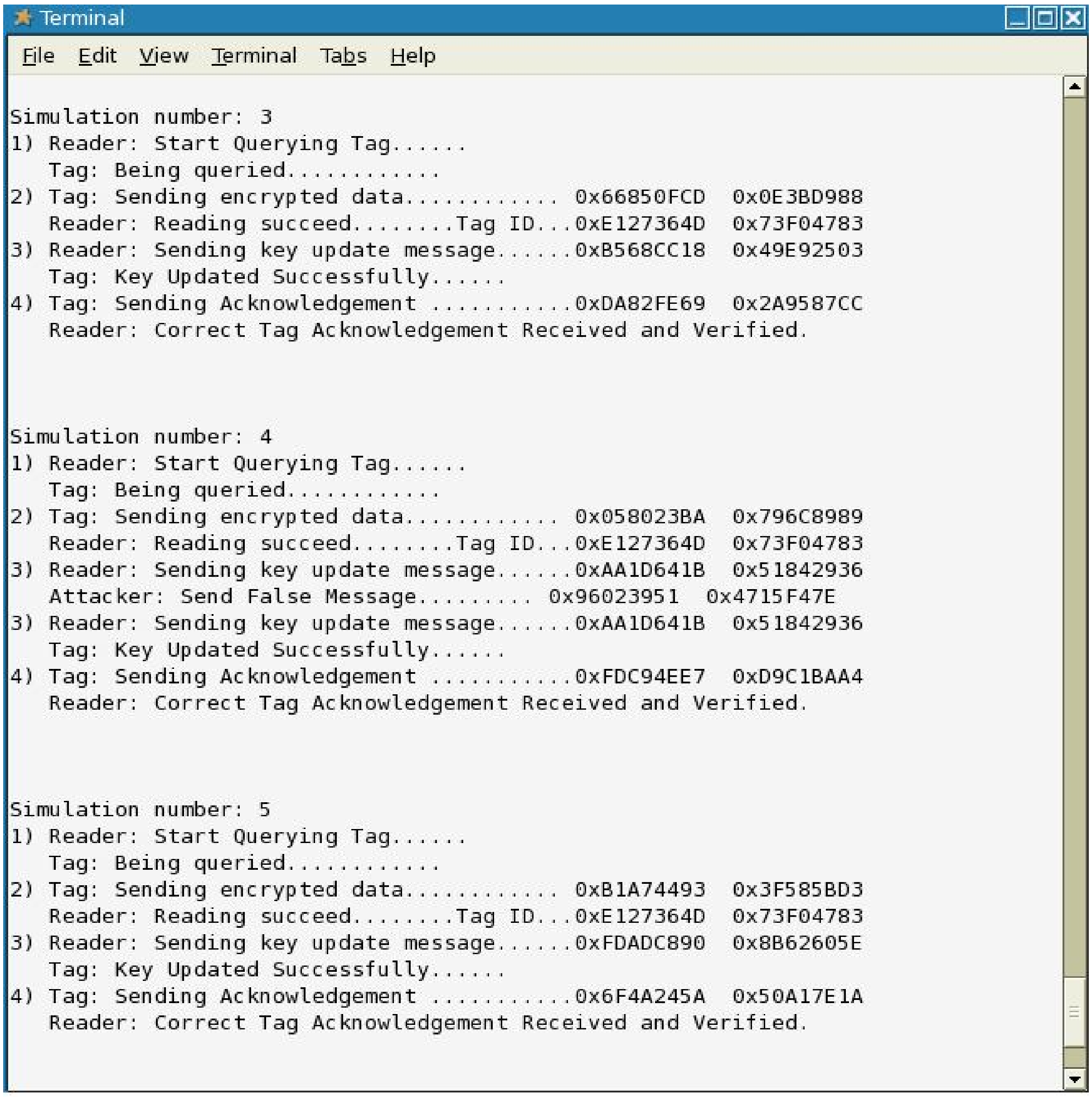


Figure 4.9 Man-in-the-Middle Attack Model

In this simulation, the man-in-the-middle attacker simply passes the signal received from port rx1 to tx1. When receiving the query signal from rx0, it forwards the message to the tx0 port. If the reader sends out a key update message, the attacker tries to alter it randomly and sends it to the tag. The attack is simulated, and the results are shown in Figure 4.10.



```
Simulation number: 3
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0x66850FCD  0x0E3BD988
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0xB568CC18  0x49E92503
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xDA82FE69  0x2A9587CC
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 4
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0x058023BA  0x796C8989
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0xAA1D641B  0x51842936
   Attacker: Send False Message......... 0x96023951  0x4715F47E
3) Reader: Sending key update message......0xAA1D641B  0x51842936
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0xFDC94EE7  0xD9C1BAA4
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 5
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0xB1A74493  0x3F585BD3
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0xFDADC890  0x8B62605E
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x6F4A245A  0x50A17E1A
   Reader: Correct Tag Acknowledgement Received and Verified.
```

Figure 4.10 Man-in-the-Middle Attack Model Simulation

It is observed that in simulation number four, the attacker tries to send a false message to the tag. The tag finds that the key update message received is illegal and therefore will not send out a key update acknowledgement based on that. Furthermore, the tag cannot be desynchronized from the server because the reader re-sends the key update message and the key set is updated finally.

4.5 Denial-of-Service Attack Model

In the denial-of-service attack, an attacker is capable of blocking any specific communication process. In our protocol, blocking the query message in the first step or the tag reply in the second step makes the reader unable to read the tag in the first attempt, but the reader can still read the tag in the next reading operation when the RF channel is not blocked. For this reason, our analysis concentrates on the key update message in this model simulation. The purpose of the attack is to desynchronize the tag and the back-end server. We put the attacker at the downlink to block the key-update message and such a model is described in Figure 4.11.

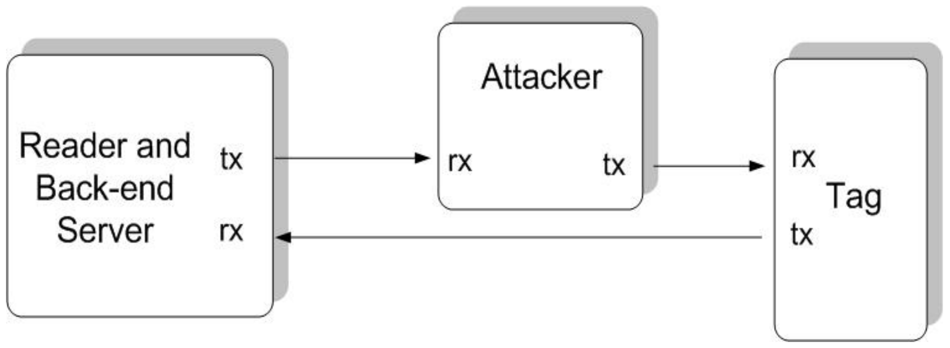
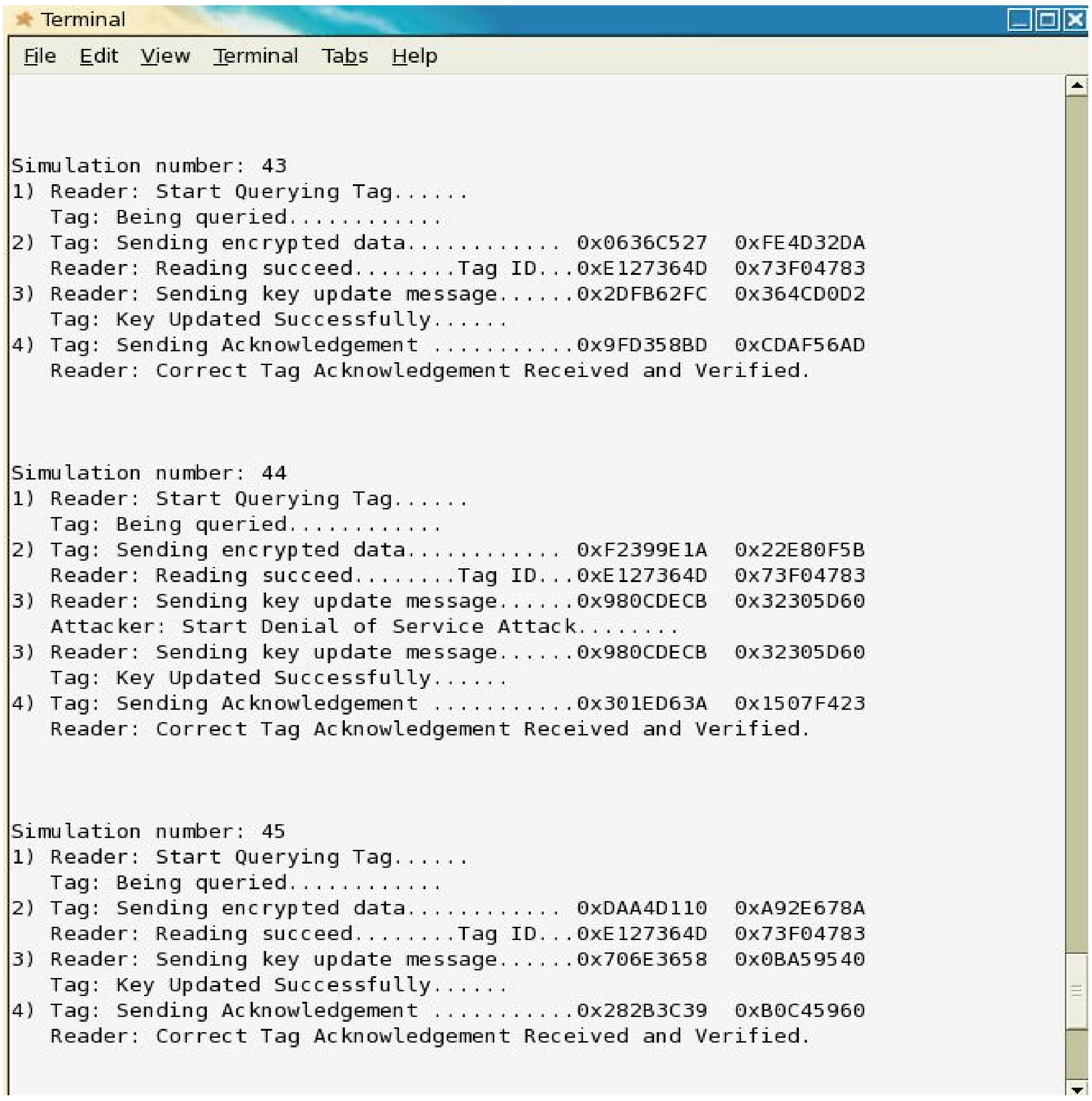


Figure 4.11 Denial-of-Service Attack Model

The simulation results are presented in Figure 4.12. When the attacker launches the denial-of-service attack in every second session, the key update message is blocked and no signal is passed to the tag module. The reader cannot receive an update acknowledgement in an expected period, and therefore re-sends key update message at a later time. When the tag receives the key update message and it generates a key update acknowledgement. Finally, the reader and the tag are able to achieve synchronization. The simulation results show that the system is secure against denial-of-service attacks.



```
Simulation number: 43
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0x0636C527  0xFE4D32DA
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x2DFB62FC  0x364CD0D2
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x9FD358BD  0xCDAF56AD
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 44
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0xF2399E1A  0x22E80F5B
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x980CDECB  0x32305D60
   Attacker: Start Denial of Service Attack........
3) Reader: Sending key update message......0x980CDECB  0x32305D60
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x301ED63A  0x1507F423
   Reader: Correct Tag Acknowledgement Received and Verified.



Simulation number: 45
1) Reader: Start Querying Tag......
   Tag: Being queried...........
2) Tag: Sending encrypted data........... 0xDAA4D110  0xA92E678A
   Reader: Reading succeed........Tag ID...0xE127364D  0x73F04783
3) Reader: Sending key update message......0x706E3658  0x0BA59540
   Tag: Key Updated Successfully......
4) Tag: Sending Acknowledgement ...........0x282B3C39  0xB0C45960
   Reader: Correct Tag Acknowledgement Received and Verified.
```

Figure 4.12 Denial-of-Service Attack Model Simulation

4.6 Active Scanning Attack Model

In the active scanning attack model, the tag is isolated from the reader. An attacker can scan it continuously without authorization of the tag owner. The purpose of this attack is to retrieve useful messages from the tag. The configuration of the model is shown in Figure 4.13.

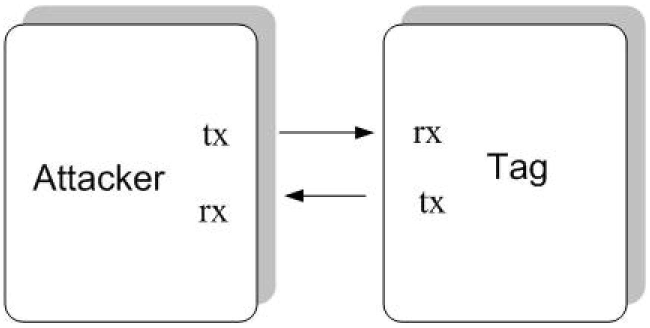
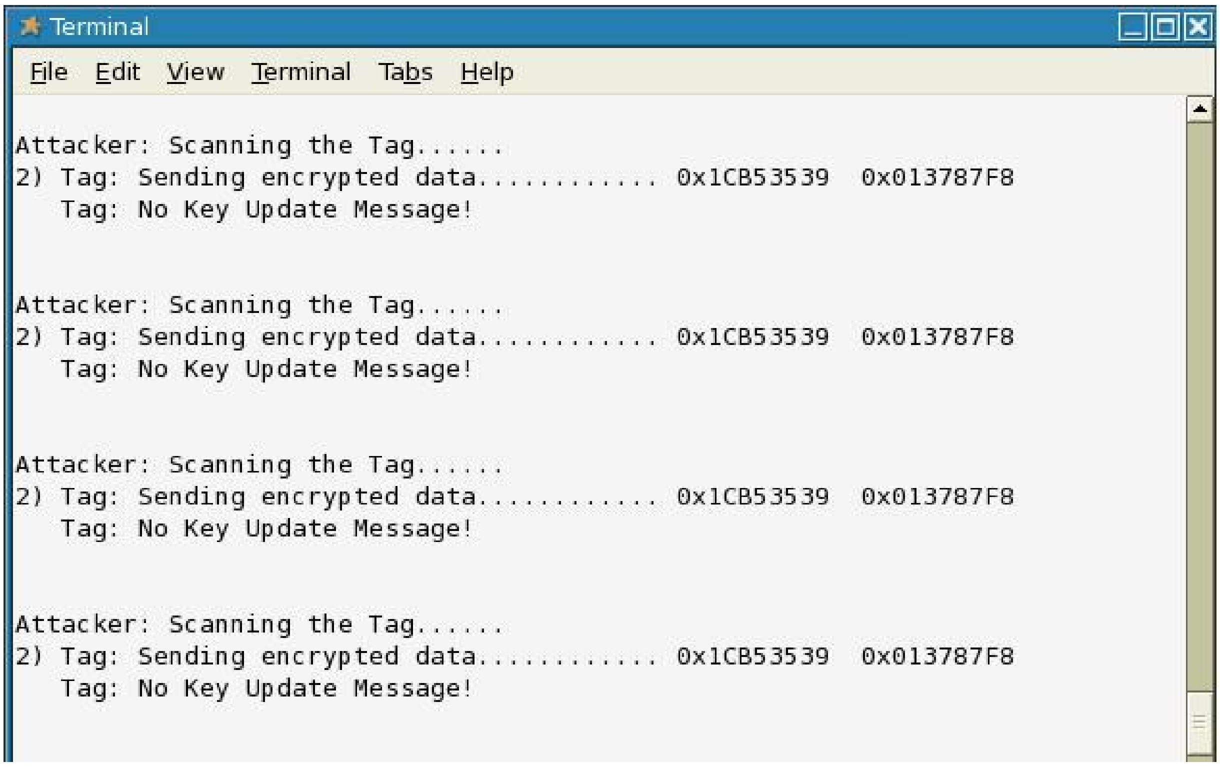


Figure 4.13 Active Scanning Attack Model

In this simulation, the attacker selects the tag as its target and sends query messages repeatedly without giving a key update message. The results of the attack are shown as in Figure 4.14.



```
Attacker: Scanning the Tag......
2) Tag: Sending encrypted data............ 0x1CB53539  0x013787F8
   Tag: No Key Update Message!


Attacker: Scanning the Tag......
2) Tag: Sending encrypted data............ 0x1CB53539  0x013787F8
   Tag: No Key Update Message!


Attacker: Scanning the Tag......
2) Tag: Sending encrypted data............ 0x1CB53539  0x013787F8
   Tag: No Key Update Message!


Attacker: Scanning the Tag......
2) Tag: Sending encrypted data............ 0x1CB53539  0x013787F8
   Tag: No Key Update Message!
```

Figure 4.14 Active Scanning Attack Model Simulation

When the attacker keeps scanning the tag, the tag only responds with an encrypted message. If no other key update message is transmitted to the tag, the tag will go back to its initial state waiting for the next query. Attacker cannot get authenticated by the tag and the tag secrets are not disclosed. The simulation results prove that our scheme is safe against active scanning attacks.

Chapter 5

Protocol Analysis by Prototyping

5.1 RFID Authentication Protocol Experimental Analysis System

In the last chapter, the simulation of our protocol is performed using SystemC based modeling techniques. In order to analyze its workability and robustness, it will be interesting to implement our protocol in a hardware experimental platform.

Previous works on RFID protocols presented different implementation options on various platforms. Luo et al.'s work evaluated encryption algorithms in sensor nodes consisting of ATMEGA128 microcontroller and Chipcon CC1100 transceiver [40]. Chae et al. implemented RC5 algorithm on WISP UHF tag, which is a programmable battery-free tag with back-scatter communication capability [41]. Israsena synthesized TEA and XTEA encryption cores in ASIC using Mentor Graphic's LeonardoSpectrum targeting 0.35μm CMOS technology [42]. Luo et al. analyzed their proposal by implementing reader and tag modules using JAVA TCP/IP socket application and database using MySQL [43]. Gilbert proposed a variable key authentication scheme and a variable round authentication scheme with implementation applied on Altera FPGA platform. The reader module was designed as a Nios-II processor using SOPC tool and the tag module was implemented using hardware description language [44]. Borghei's hash based authentication protocol was inspired by SRAC protocol. In her implementation, the reader design was achieved on a PC console using windows application programming interface and the tag was designed as an Altera Nios-II system with smart card [45]. Angerer et al.'s work proposed a flexible RFID prototyping system consisting of a RF front end, a protocol stack operating on a DSP and a signal processing module implemented on FPAG [46].

The methods used in [40, 41] require complex hardware systems, while the implementation in [42, 43] is essentially software based. Overall, we prefer a system that is implemented on a FPGA platform similar to the works in [44, 45 and 46]. As compared with other implementation schemes, FPGA environment enjoy many advantages, including low cost, time saving and flexible prototyping. In this chapter, we present the implementation of our protocol on an Altera FPGA DE2 board. A DE2 board consists of an Altera Cyclone II EP2C35F672C6 FPGA device and an EPCS16 serial configuration device as well as other components including memories, USB port, Ethernet port, VGA port, audio connectors, a number of switches and LEDs [47].

5.2 The Active Scanning Attack Model

In an active scanning attack, the tag is isolated from legitimate readers and the attacker is constantly scanning the tag without owner's authorization. Figure 5.1 shows the Nios system implementation of this model.

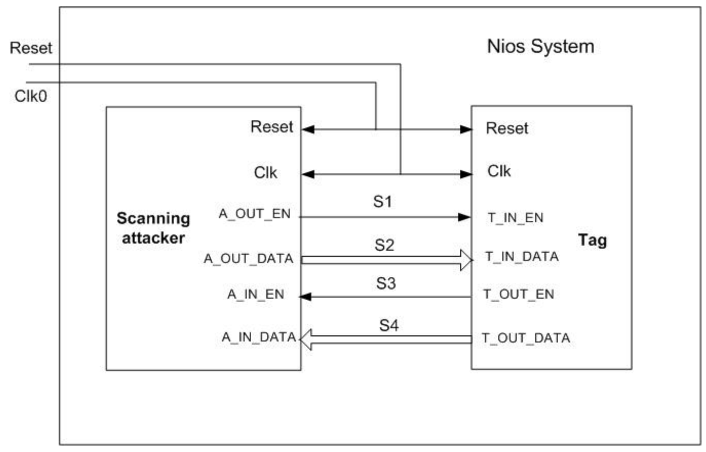
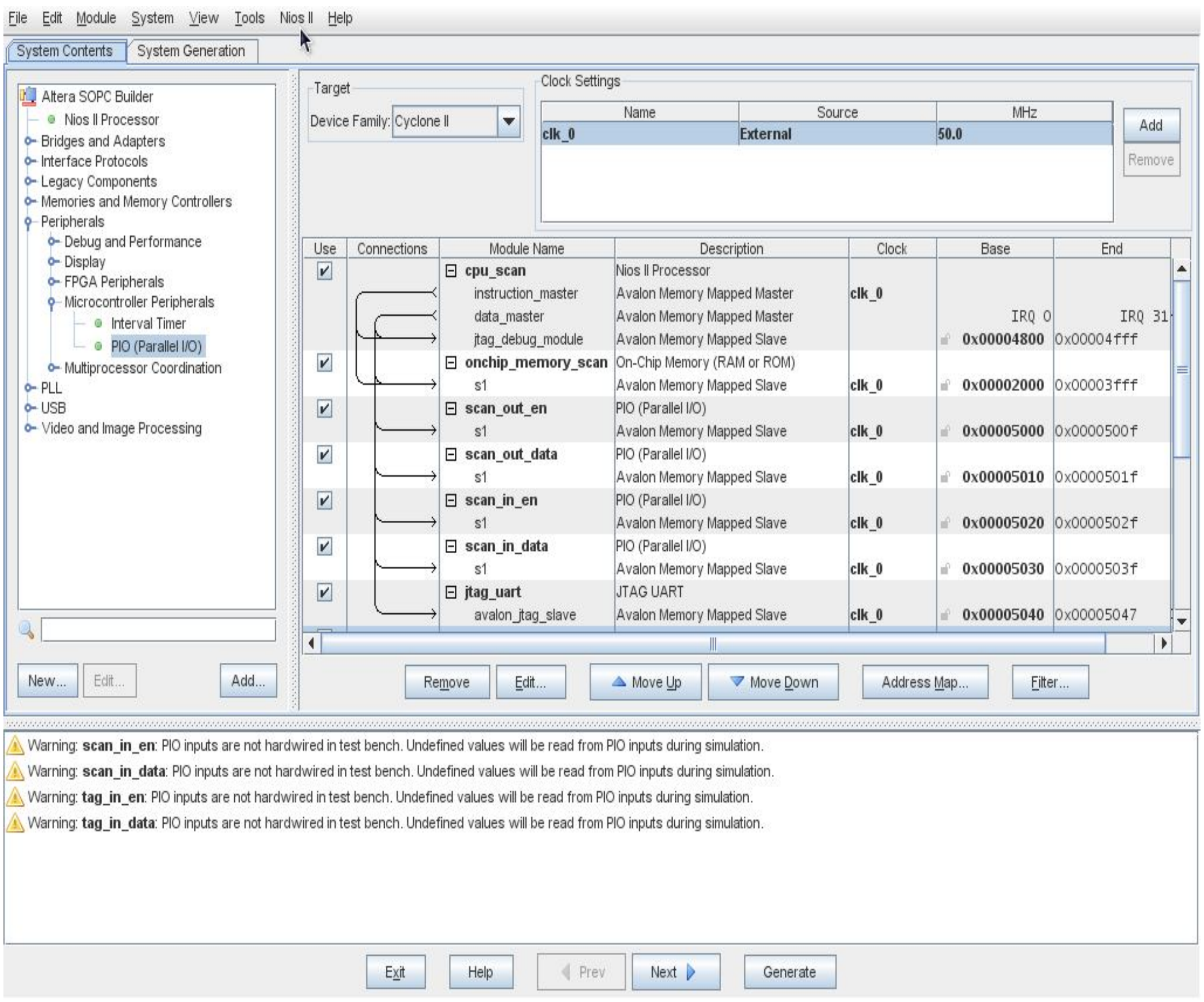


Figure 5.1 Active Scanning Attack Model Implementation using Nios-II System

The model is implemented using SOPC builder, a powerful tool available in Quartus II environment. The attacker module and the tag module are implemented as individual Nios-II processors. Figure 5.2 shows the configuration of the attacker module processor. Parallel ports are used to connect the attacker and the tag modules and allow them to communicate with each other.



Target

Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Module Name	Description	Clock	Base	End
✓	cpu_scan	Nios II Processor			
	instruction_master	Avalon Memory Mapped Master	clk_0		
	data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31
	jtag_debug_module	Avalon Memory Mapped Slave		0x00004800	0x00004fff
✓	onchip_memory_scan	On-Chip Memory (RAM or ROM)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00002000	0x00003fff
✓	scan_out_en	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00005000	0x0000500f
✓	scan_out_data	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00005010	0x0000501f
✓	scan_in_en	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00005020	0x0000502f
✓	scan_in_data	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00005030	0x0000503f
✓	jtag_uart	JTAG UART			
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00005040	0x00005047

```
Warning: scan_in_en: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: scan_in_data: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: tag_in_en: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: tag_in_data: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
```

Figure 5.2 Active Scanning Attacker Module

In this test, we recorded the messages exchanged in the first seven sessions. The results are summarized in Table 5.1.

Table5.1 Active Scanning Attack Messages Exchanged

Protocol Round	Attacker Send Message	Tag Response Message
1	Tag Query	B57183930FE7A64B
2	Tag Query	B57183930FE7A64B
3	Tag Query	B57183930FE7A64B
4	Tag Query	B57183930FE7A64B
5	Tag Query	0000000000000000
6	Tag Query	0000000000000000
7	Tag Query	0000000000000000

As it is observed in the test, if the tag is scanned more than four times, the tag will stop responding to further queries with the encrypted reply message. This is due to a parameter introduced for the tag for additional security considerations. If the tag is queried more than this threshold without receiving correct key update messages, it will respond with a meaningless message. In our implementation, the tag simply replies with a constant string of ‘0’, which also serves as a warning that the tag has been attacked by active scanning attackers. The implementation verifies that this protocol is resistant to the active scanning attacks.

5.3 The Cloning Attack Model

Cloning attack happens when a reader and its tag are communicating in a channel which is insecure and attackers can intercept the messages coming from the tag and the reader. Figure 5.3 shows the Nios-II system of this model. Since there is only one UART interface connected to the reader in the configuration, we combine the function of the channel with the attacker module. In the implementation of this attack model, the reader first queries the tag in a normal operation. Then, the reader is going to check whether the message recorded by the attacker matches with the secrets it currently has. If they do not match, it means our protocol is safe against cloning attack.

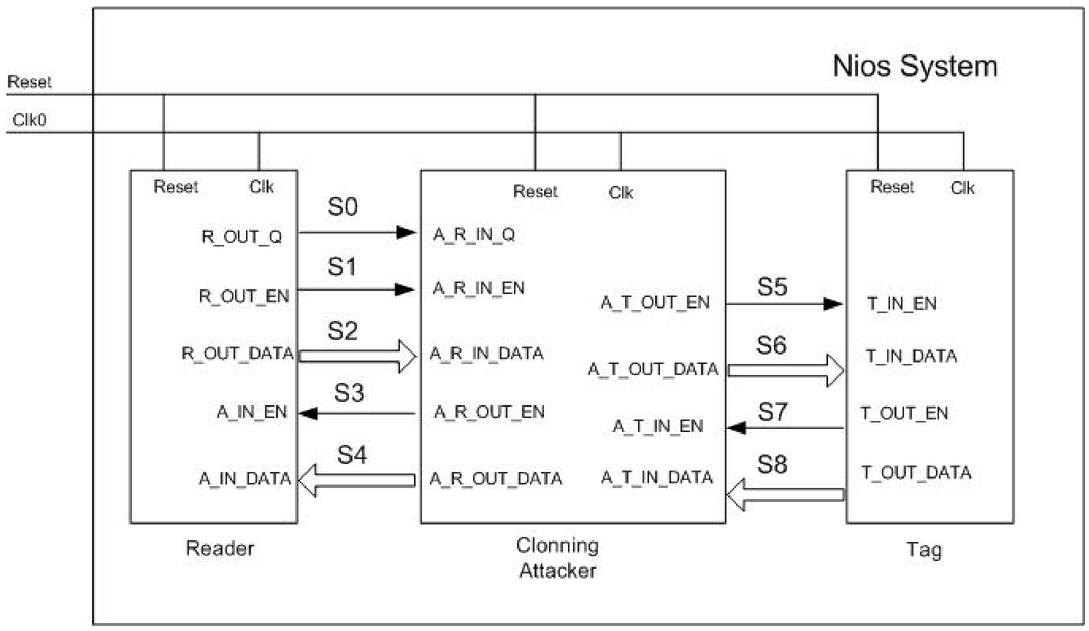
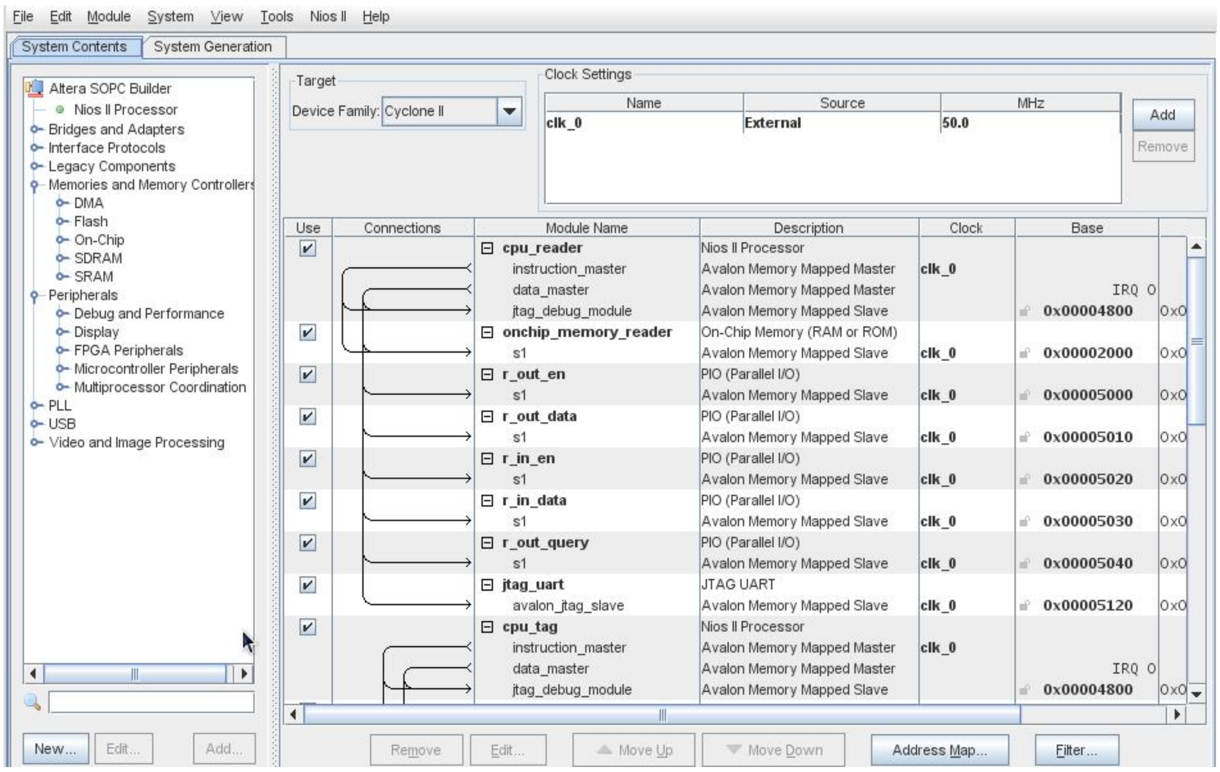


Figure 5.3 Cloning Attack Model Implementation using Nios-II System

The reader, the attacker and the tag modules are implemented as separate Nios-II processors. The communications between these processors are established through parallel ports. In each direction, the rising signal of enable port defines exact timing that the data in the data port is ready for sampling. Figure 5.4 shows the reader modules deployed in this model. For the cloning attack test, we recorded the messages exchanged for a period of an hour with a new session started every five seconds. However, we did not observe any case of failure. The results for the first seven sessions of the protocol operation are given in Table 5.2.



Target

Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Module Name	Description	Clock	Base
✓	cpu_reader	Nios II Processor		
	instruction_master	Avalon Memory Mapped Master	clk_0	
	data_master	Avalon Memory Mapped Master		IRQ 0
	jtag_debug_module	Avalon Memory Mapped Slave		0x00004800
✓	onchip_memory_reader	On-Chip Memory (RAM or ROM)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00002000
✓	r_out_en	PIO (Parallel I/O)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00005000
✓	r_out_data	PIO (Parallel I/O)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00005010
✓	r_in_en	PIO (Parallel I/O)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00005020
✓	r_in_data	PIO (Parallel I/O)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00005030
✓	r_out_query	PIO (Parallel I/O)		
	s1	Avalon Memory Mapped Slave	clk_0	0x00005040
✓	jtag_uart	JTAG UART		
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00005120
✓	cpu_tag	Nios II Processor		
	instruction_master	Avalon Memory Mapped Master	clk_0	
	data_master	Avalon Memory Mapped Master		IRQ 0
	jtag_debug_module	Avalon Memory Mapped Slave		0x00004800

Figure 5.4 Reader Module for the Cloning Attack Model

Table 5.2 Cloning Attack Messages Exchanged

Session Number	Tag Response Message To Query	Tag ID	Effectiveness of Attack
1	B57183930FE7A64B	2233221221314783	No
2	AF4D7FD9DDF60A22	2233221221314783	No
3	97E6D6FC8072E812	2233221221314783	No
4	CF964D1BB89D2299	2233221221314783	No
5	E06B1A8697863DA6	2233221221314783	No
6	9F5649FB374A23EC	2233221221314783	No
7	3A2C61372888A673	2233221221314783	No

As confirmed from the results, the tag replies with an encrypted message in every new session. The cloning attacker cannot obtain the secrets of tag because it does not know the key set. The implementation verifies that our protocol is resistant against cloning attacks.

5.4 The Replay Attack Model

In a replay attack, when a reader and its tags communicate via an insecure channel, an attacker can intercept the messages between the tags and the reader. Figure 5.5 shows the Nios-II system that is prototyped to model such an attack. In this implementation, the reader first queries the tag in a normal way. Then, it queries the attacker to see if the recorded message matches with the secrets it has currently. If there is no match condition, it verifies that our protocol is safe against replay attacks.

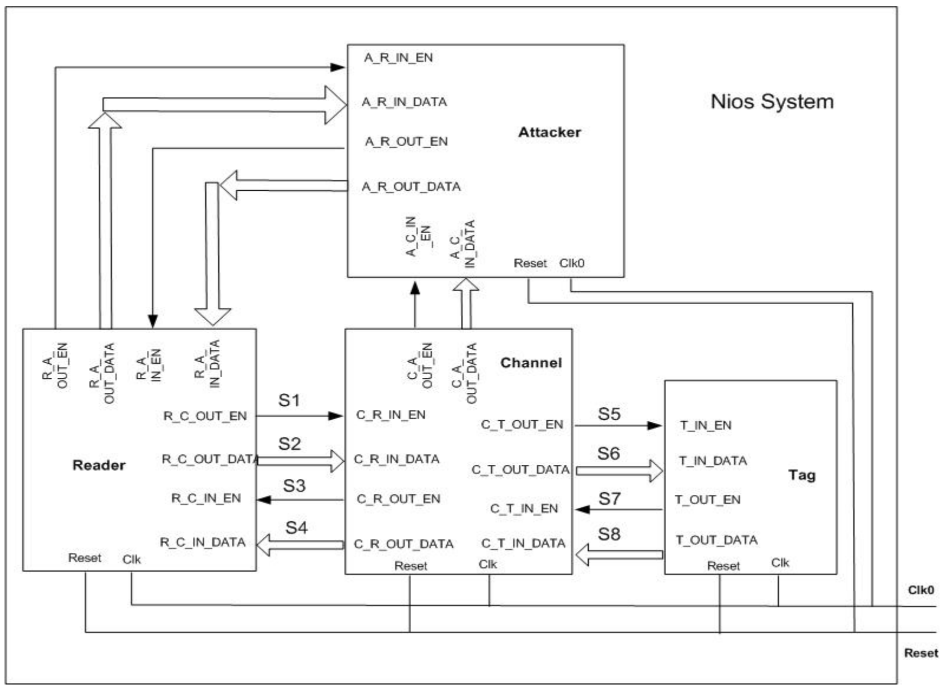
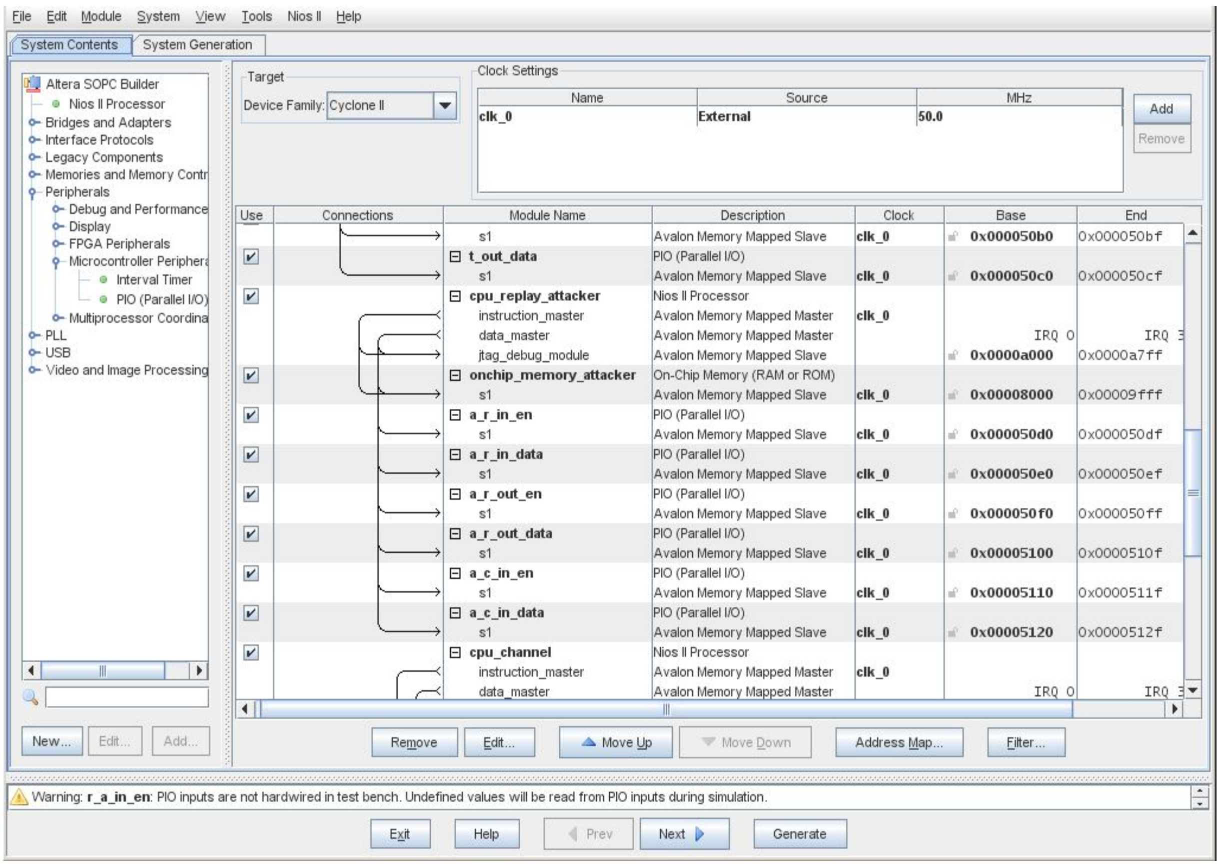


Figure 5.5 Replay Attack Model Implementation using Nios-II System

The reader, attacker, channel and tag modules are implemented as individual Nios-II processors. The processors are connected with each other through parallel ports so that communications between modules can happen. The attacker module is able to monitor all the messages going through the channel module. Figure 5.6 shows the attacker module created using the SOPC Builder tool.



Target

Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Connections	Module Name	Description	Clock	Base	End
		s1	Avalon Memory Mapped Slave	clk_0	0x000050b0	0x000050bf
✓		⊟ t_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050c0	0x000050cf
✓		⊟ cpu_replay_attacker	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	clk_0		
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 3
		jtag_debug_module	Avalon Memory Mapped Slave		0x0000a000	0x0000a7ff
✓		⊟ onchip_memory_attacker	On-Chip Memory (RAM or ROM)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x00009fff
✓		⊟ a_r_in_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050d0	0x000050df
✓		⊟ a_r_in_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050e0	0x000050ef
✓		⊟ a_r_out_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050f0	0x000050ff
✓		⊟ a_r_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005100	0x0000510f
✓		⊟ a_c_in_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005110	0x0000511f
✓		⊟ a_c_in_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005120	0x0000512f
✓		⊟ cpu_channel	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	clk_0		
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 3

Warning: **r_a_in_en**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 5.6 Attacker Module for the Replay Attack Model

During the protocol testing, we recorded the messages exchanged for an hour period with a new reading operation triggered every ten seconds. Then the attacker is queried after the reading is finished to verify the effectiveness of replay attack. We did not observe any case of a replay attack succeeding. The results for the first seven sessions of the protocol operation are summarized in Table 5.3.

Table 5.3 Replay Attack Messages Exchanged

Session Number	Tag Response Message	Tag Key Set	Attacker Replay Message	Effectiveness of Attack
1	B5718393, 0FE7A64B	AB47856C, 902C729A, 70BC93E4 , 247DC618	B5718393, 0FE7A64B	No
2	AF4D7FD9, DDF60A22	AB47856C, 902C729A 70BC93E4, DA6B4E51	AF4D7FD9, DDF60A22	No
3	97E6D6FC, 8072E812	7825DFCB, 902C729A 70BC93E4, DA6B4E51	97E6D6FC, 8072E812	No
4	CF964D1B, B89D2299	7825DFCB, 9D7A9053 70BC93E4, DA6B4E51	CF964D1B, B89D2299	No
5	E06B1A86, 97863DA6	7825DFCB, C217C338 70BC93E4, DA6B4E51	E06B1A86, 97863DA6	No
6	9F5649FB, 374A23EC	4E9221E0, C217C338 70BC93E4, DA6B4E51	9F5649FB, 374A23EC	No
7	3A2C6137, 2888A673	792F11CF, C217C338 70BC93E4, DA6B4E51	3A2C6137, 2888A673	No

The prototyping results show that the tag is able to reply with an encrypted message in every new session when being queried. Furthermore, one sub-key in its key set is also updated. The replay attacker tried to impersonate a genuine tag by replaying the previous tag output, but it is not the correct response the reader expects. Therefore, the replay attacker cannot pass the authentication process. The prototyping of the replay attack model verifies that our novel protocol is safe against replay attacks.

5.5 The Denial-of-Service Attack Model

In the denial-of-service attack model, the communication channel between a reader and its tag can be blocked by an attacker. The configuration of this model is prototyped as shown in Figure 5.7. In the implementation of this prototype, the reader module first queries the tag module and then receives the reply from it. When the key update message arrives, the denial-of-service attacker tries to block it in attempt to desynchronize the tag and the reader. If the reader and the tag can reestablish normal operation, then it verifies our protocol is robust and safe against denial-of-service attacks.

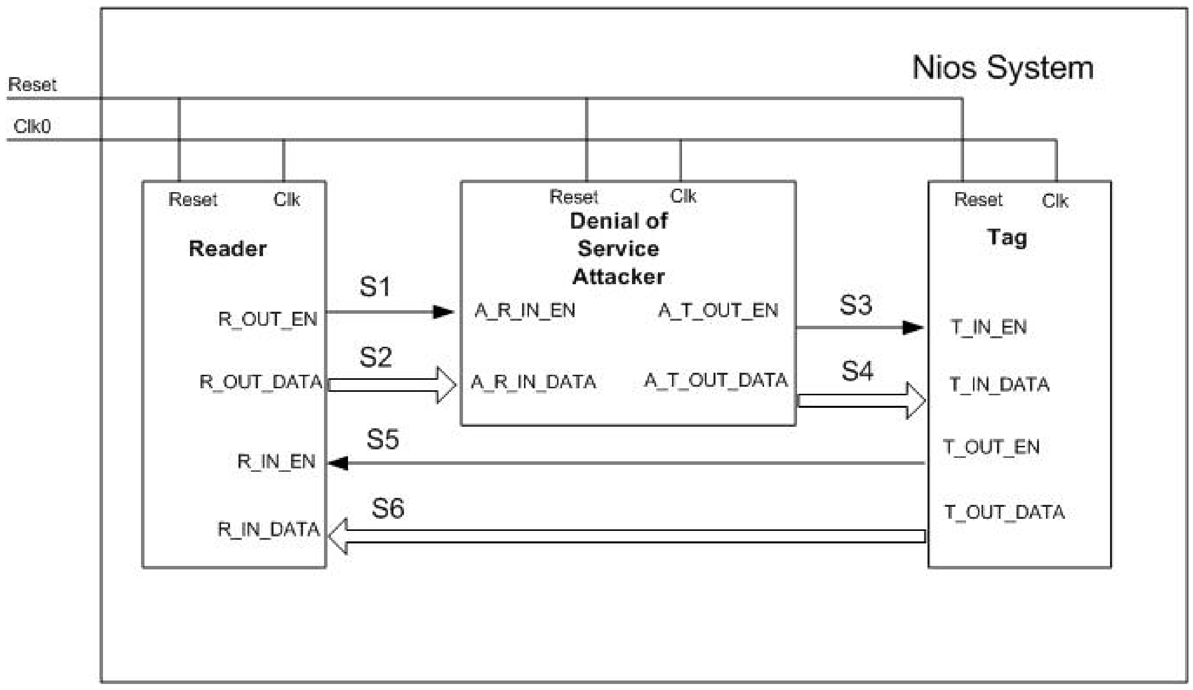
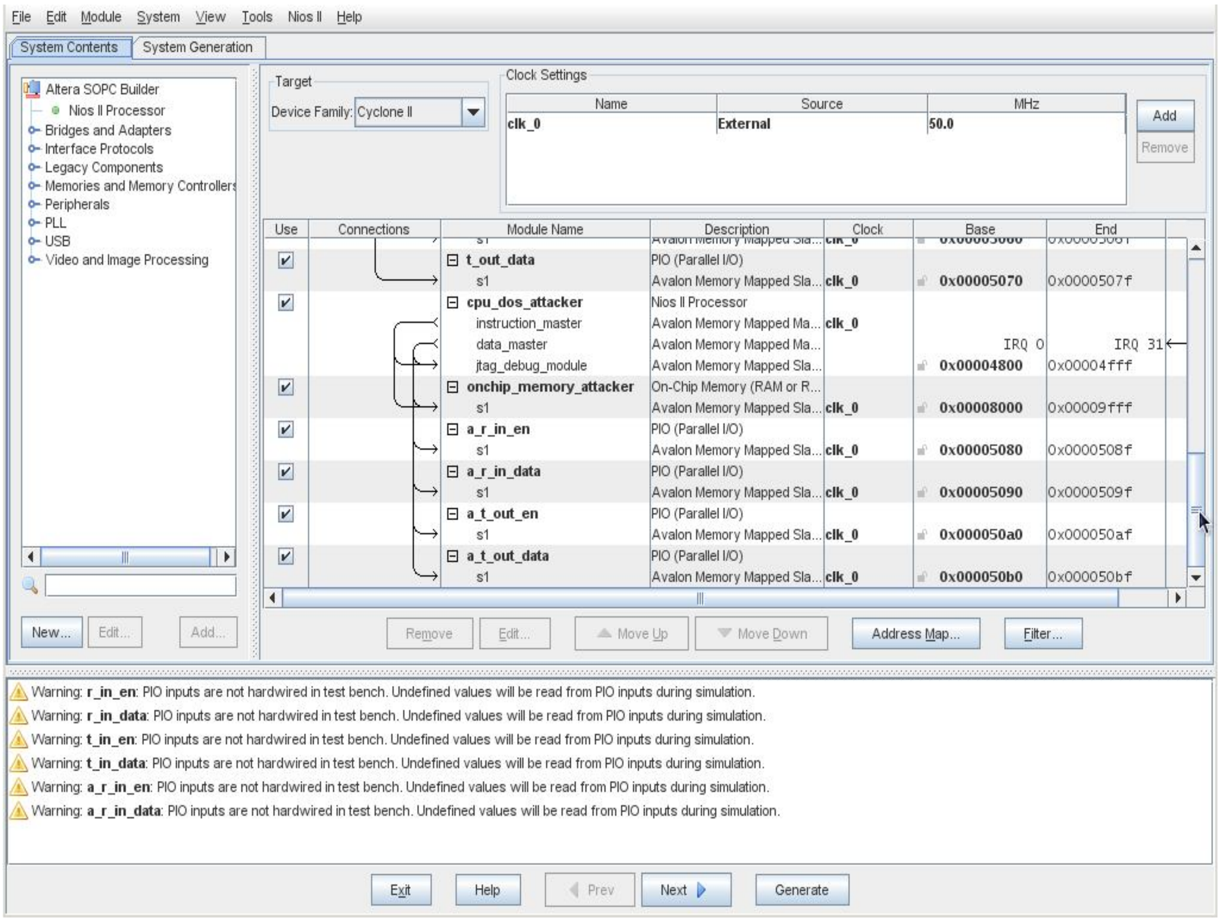


Figure 5.7 Denial-of-Service Attack Model Implementation using Nios-II System

The reader, the attacker and the tag modules are implemented as individual Nios-II processors. Using the same technique in previous models, the processors are linked to each other through parallel ports. Figure 5.8 shows the Nios-II processor for the attacker module configured using SOPC Builder tool environment.



Target

Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Connections	Module Name	Description	Clock	Base	End
✓		t_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Sla...	clk_0	0x00005070	0x0000507f
✓		cpu_dos_attacker	Nios II Processor			
		instruction_master	Avalon Memory Mapped Ma...	clk_0		
		data_master	Avalon Memory Mapped Ma...		IRQ 0	IRQ 31
		jtag_debug_module	Avalon Memory Mapped Sla...		0x00004800	0x00004fff
✓		onchip_memory_attacker	On-Chip Memory (RAM or R...			
		s1	Avalon Memory Mapped Sla...	clk_0	0x00008000	0x00009fff
✓		a_r_in_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Sla...	clk_0	0x00005080	0x0000508f
✓		a_r_in_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Sla...	clk_0	0x00005090	0x0000509f
✓		a_t_out_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Sla...	clk_0	0x000050a0	0x000050af
✓		a_t_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Sla...	clk_0	0x000050b0	0x000050bf

Warning: **r_in_en**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **r_in_data**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **t_in_en**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **t_in_data**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **a_r_in_en**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **a_r_in_data**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 5.8 Attacker Module for the Denial-of-Service Attack Model

During the protocol verification, we recorded the messages exchanged for one hour with a new reading operation triggered every ten seconds. We did not observe a case that an attack could be successful. The results for the first seven sessions are summarized in Table 5.4.

Table 5.4 Denial-of-Service Attack Messages Exchanged

Session Number	Tag Response Message To Query	Tag Key Update Message	Tag Key Update Message Reader Re-send	Tag Key Update ACK Message	Effectiveness of Attack
1	B5718393, 0FE7A64B	8EA3D37C, 3F5198CA	8EA3D37C, 3F5198CA	2B6EDC58, 1B49A4D8	No
2	AF4D7FD9, DDF60A22	BE0B6D17, 27D8F698	BE0B6D17, 27D8F698	A7C5352E, CB059EBD	No
3	97E6D6FC, 8072E812	70F67232, FAA7DB1A	70F67232, FAA7DB1A	110D71D7, 99059299	No
4	CF964D1B, B89D2299	EE83D92B, 2D52F3F9	EE83D92B, 2D52F3F9	DC693F94 E0076EFE	No
5	E06B1A86, 97863DA6	C6C69B41, 94A7E2F1	C6C69B41, 94A7E2F1	37BDFDA6, FBE840A1	No
6	9F5649FB, 374A23EC	F2D1EB14, EDEE7EC1	F2D1EB14, EDEE7EC1	41D18FB4, 0B60BEBD	No
7	3A2C6137, 2888A673	B88CCB65, 1F511A01	B88CCB65, 1F511A01	92238AA7, B0FE05E7	No

It shows that the tag key update operation is successful in every session. The key update message can be blocked by the denial-of-service attacker periodically in the test. In the timeout period, if the reader does not receive the key update acknowledgement, it will re-send the message and the tag is able to receive it. Finally, the key update acknowledgement arrives indicating that the tag's key set is synchronized with the reader.

5.6 The Man-in-the-Middle Attack Model

In the man-in-the-middle attack model, an attacker stays between a reader and its tags and acts like a reader to the tag or appears to be a tag to the reader. Figure 5.9 shows the Nios-II system implemented for the prototype of this model. In this test, the reader first queries the tag and receives its reply. When the key update message is sent out from the reader, the man-in-the-middle attacker attempts to modify it. If the tag is able to detect the message is not legitimate, it verifies that our protocol is safe against man-in-the-middle attacks.

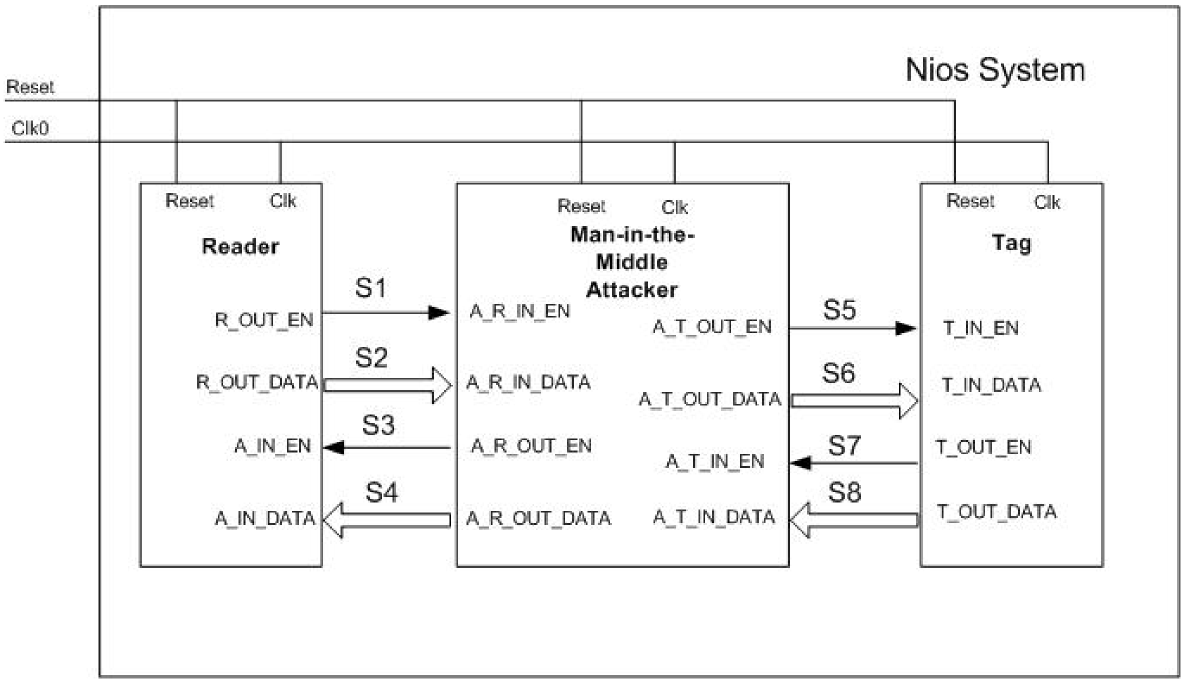
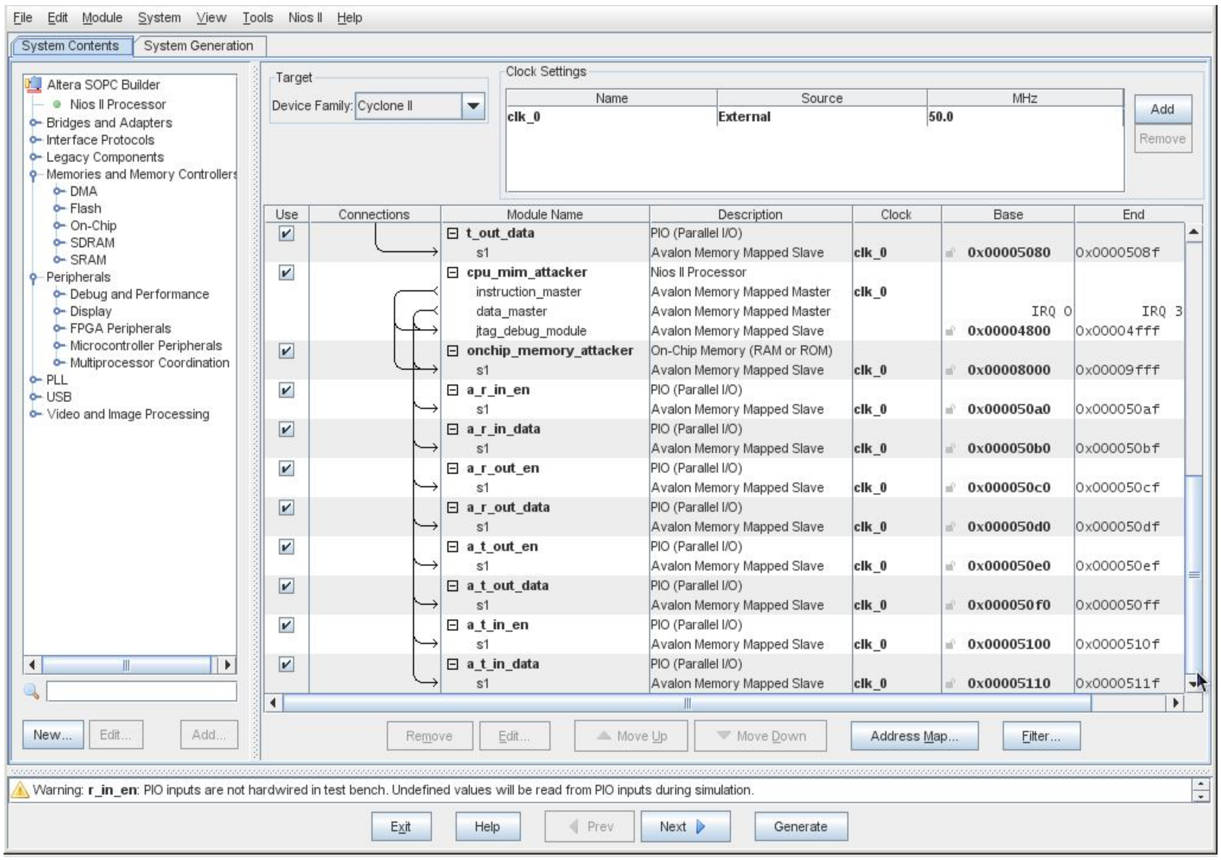


Figure 5.9 Man-in-the-Middle Attack Model Implementation using Nios-II System

The reader, attacker and tag modules are implemented as individual Nios-II processors. Figure 5.10 shows the attacker module of this model designed in the SOPC Builder environment.



Target

Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Connections	Module Name	Description	Clock	Base	End
✓		⊟ t_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005080	0x0000508f
✓		⊟ cpu_mim_attacker	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	clk_0		
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 3
		jtag_debug_module	Avalon Memory Mapped Slave		0x00004800	0x00004fff
✓		⊟ onchip_memory_attacker	On-Chip Memory (RAM or ROM)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x00009fff
✓		⊟ a_r_in_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050a0	0x000050af
✓		⊟ a_r_in_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050b0	0x000050bf
✓		⊟ a_r_out_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050c0	0x000050cf
✓		⊟ a_r_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050d0	0x000050df
✓		⊟ a_t_out_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050e0	0x000050ef
✓		⊟ a_t_out_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x000050f0	0x000050ff
✓		⊟ a_t_in_en	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005100	0x0000510f
✓		⊟ a_t_in_data	PIO (Parallel I/O)			
		s1	Avalon Memory Mapped Slave	clk_0	0x00005110	0x0000511f

Warning: **r_in_en**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 5.10 Attacker Module for the Man-in-the-Middle Attack Model

In the experimental test, we recorded the messages exchanged for one hour with a new reading operation triggered every ten seconds. We did not observe any case that a successful attack could happen. A sample of the test results from session 28 to 34 is summarized in Table 5.5.

Table 5.5 Man-in-the-Middle Attack Messages Exchanged

Session Number	Tag Response Message To Query	Tag Key Update Message Reader Send	Tag Key Update Message Reader Re-send	Tag Key Update ACK Message	Effectiveness of Attack
28	BE0F3DF6, DD11E327	7C6F3423, DA1EE993	7C6F3423, DA1EE993	51B7093F, 4E4B5E76	No
29	2962986F, 2B17C3FA	F6D59CA9, F92AF59B	F6D59CA9, F92AF59B	685BF097, DBECF678	No
30	D7F46FAE, F0FE0ADF	63C4CFEF, C12C399C	63C4CFEF, C12C399C	A8A9C526, 4F35367B	No
31	6ACF5229, 4D6B3B82	20951352, B3E721FD	20951352, B3E721FD	2CAB2088, 597F2810	No
32	71861D2C, 12174846	0CEF9372, BA89253D	0CEF9372, BA89253D	EF222F41, 1FE998C4	No
33	A8F5EE0, 89E2D8DF	71E95F13, 16D89577	71E95F13, 16D89577	80198675 1A0FA400	No
34	D9BD752A, 888FBF27	FC907F17, B7F84EC8	FC907F17, B7F84EC8	88781C4F, 64990BE8	No

The results of model prototyping show that a successful tag key update is realized in every session. The man-in-the-middle attacker tries to modify the key update message. However, when the tag decrypts the key update message, it finds the field of the last 30 bits of the tag ID does not match its own. As a result, the key update operation does not take place based on the false message. In the timeout period, since the reader cannot receive the key update acknowledgement, it re-sends the key update message. The key update acknowledgement in the last step is finally received and the reading process is finished successfully.

Chapter 6

Performance and Security Analysis

In the following sections, we analyze and compare our protocol with Toiruul and Lee's protocol and Kim et al.'s protocol [34, 38]. We select them for comparison because, like our work, they both use symmetric key algorithms in their protocols. We apply the same approaches as we make analysis on our authentication scheme. SystemC based modeling is also used to evaluate the security level for different threat models and FPGA prototyping environment is applied to analyze the performance level of these protocols and compare with ours.

6.1 Security Level Analysis

We first analyze the security level of Toiruul and Lee's protocol by building the same model as shown in Figure 4.1. In this simulation, there is no attack module applied in the model. The values of secrets are initialized as follows:

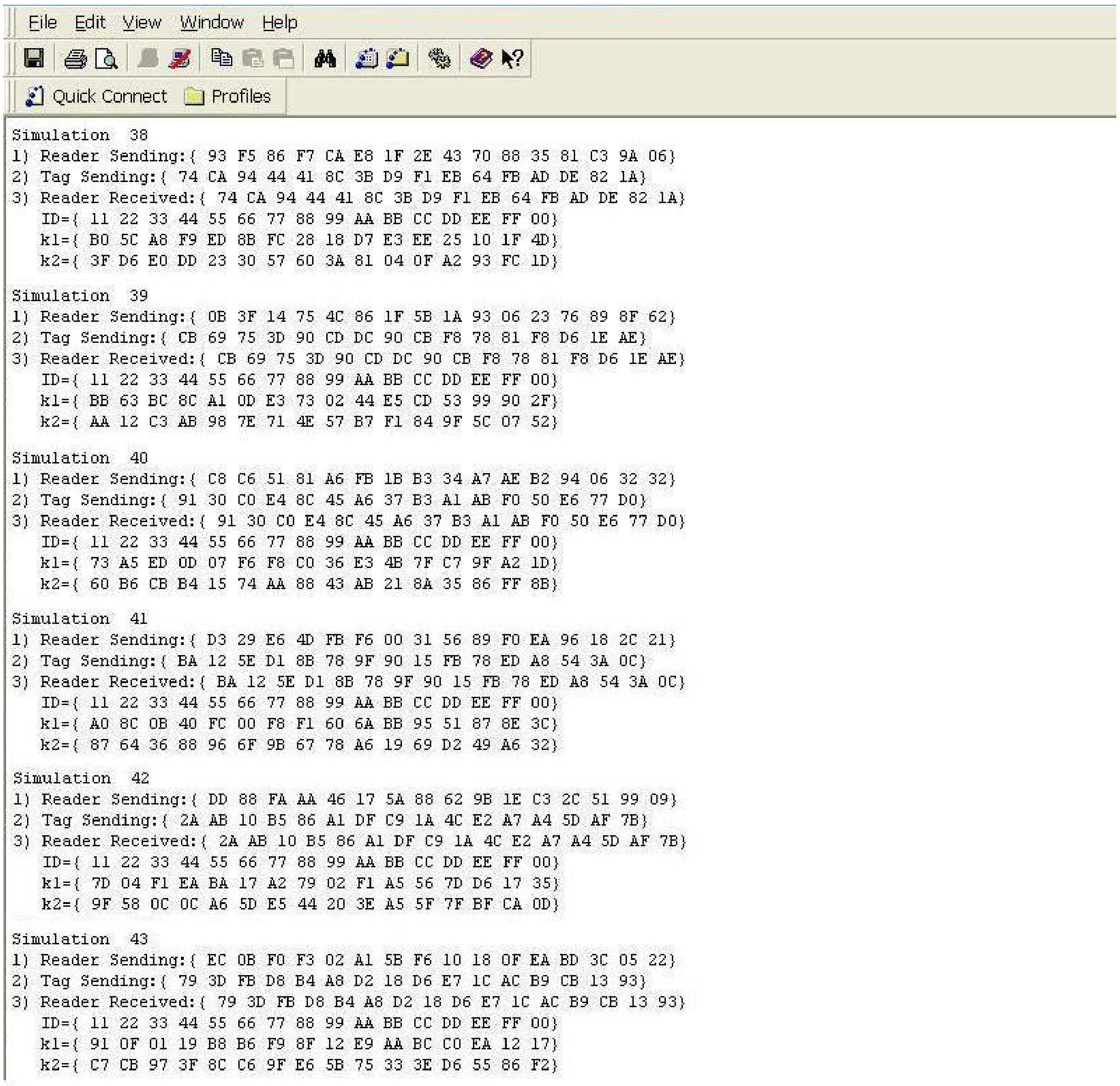
ID:{0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff,0x00}

k: {0xe3,0x25,0x87,0x34,0x19,0x34,0x90,0x90,0x56,0x13,0xdf,0x56,0x94,0x61,0x4b,0x1a}

k1:{0x23,0x19,0x83,0x84,0x91,0xdc,0xd8,0x3e,0x09,0x59,0x3c,0xb2,0xb6,0x46,0x39,0x92}

k2:{0xc1,0x50,0xd7,0xc9,0xb2,0xc8,0xb2,0x89,0xa2,0x79,0x04,0x06,0x82,0x56,0x90,0x23}

In every new session, the secrets are changed with the same process as described in the section 2.4. Protocol simulation results of from session number 38 to 43 are shown in Figure 6.1.

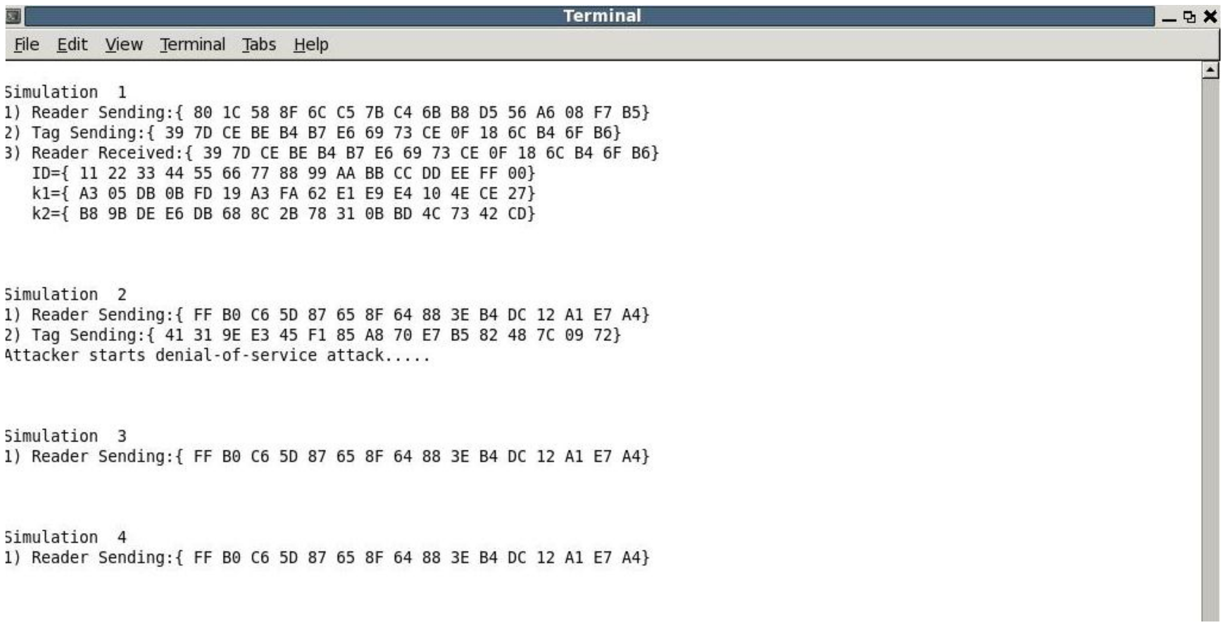


```
Simulation  38
1) Reader Sending:{ 93 F5 86 F7 CA E8 1F 2E 43 70 88 35 81 C3 9A 06}
2) Tag Sending:{ 74 CA 94 44 41 8C 3B D9 F1 EB 64 FB AD DE 82 1A}
3) Reader Received:{ 74 CA 94 44 41 8C 3B D9 F1 EB 64 FB AD DE 82 1A}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ B0 5C A8 F9 ED 8B FC 28 18 D7 E3 EE 25 10 1F 4D}
   k2={ 3F D6 E0 DD 23 30 57 60 3A 81 04 0F A2 93 FC 1D}

Simulation  39
1) Reader Sending:{ 0B 3F 14 75 4C 86 1F 5B 1A 93 06 23 76 89 8F 62}
2) Tag Sending:{ CB 69 75 3D 90 CD DC 90 CB F8 78 81 F8 D6 1E AE}
3) Reader Received:{ CB 69 75 3D 90 CD DC 90 CB F8 78 81 F8 D6 1E AE}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ BB 63 BC 8C A1 0D E3 73 02 44 E5 CD 53 99 90 2F}
   k2={ AA 12 C3 AB 98 7E 71 4E 57 B7 F1 84 9F 5C 07 52}

Simulation  40
1) Reader Sending:{ C8 C6 51 81 A6 FB 1B B3 34 A7 AE B2 94 06 32 32}
2) Tag Sending:{ 91 30 C0 E4 8C 45 A6 37 B3 A1 AB F0 50 E6 77 D0}
3) Reader Received:{ 91 30 C0 E4 8C 45 A6 37 B3 A1 AB F0 50 E6 77 D0}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ 73 A5 ED 0D 07 F6 F8 C0 36 E3 4B 7F C7 9F A2 1D}
   k2={ 60 B6 CB B4 15 74 AA 88 43 AB 21 8A 35 86 FF 8B}

Simulation  41
1) Reader Sending:{ D3 29 E6 4D FB F6 00 31 56 89 F0 EA 96 18 2C 21}
2) Tag Sending:{ BA 12 5E D1 8B 78 9F 90 15 FB 78 ED A8 54 3A 0C}
3) Reader Received:{ BA 12 5E D1 8B 78 9F 90 15 FB 78 ED A8 54 3A 0C}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ A0 8C 0B 40 FC 00 F8 F1 60 6A BB 95 51 87 8E 3C}
   k2={ 87 64 36 88 96 6F 9B 67 78 A6 19 69 D2 49 A6 32}

Simulation  42
1) Reader Sending:{ DD 88 FA AA 46 17 5A 88 62 9B 1E C3 2C 51 99 09}
2) Tag Sending:{ 2A AB 10 B5 86 A1 DF C9 1A 4C E2 A7 A4 5D AF 7B}
3) Reader Received:{ 2A AB 10 B5 86 A1 DF C9 1A 4C E2 A7 A4 5D AF 7B}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ 7D 04 F1 EA BA 17 A2 79 02 F1 A5 56 7D D6 17 35}
   k2={ 9F 58 0C 0C A6 5D E5 44 20 3E A5 5F 7F BF CA 0D}

Simulation  43
1) Reader Sending:{ EC 0B F0 F3 02 A1 5B F6 10 18 0F EA BD 3C 05 22}
2) Tag Sending:{ 79 3D FB D8 B4 A8 D2 18 D6 E7 1C AC B9 CB 13 93}
3) Reader Received:{ 79 3D FB D8 B4 A8 D2 18 D6 E7 1C AC B9 CB 13 93}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ 91 0F 01 19 B8 B6 F9 8F 12 E9 AA BC C0 EA 12 17}
   k2={ C7 CB 97 3F 8C C6 9F E6 5B 75 33 3E D6 55 86 F2}
```

Figure 6.1 Toiruul and Lee's protocol Simulation

Once a tag's current secrets are disclosed, an attacker does not need to intercept the messages in the next sessions to keep track of how the secrets are updated, because this process is fixed in the second stage of the protocol. Therefore, the future secrets can be deduced and the subsequent tag's responses can be computed. Thus, the protocol does not provide backward security.

However, it ensures forward security because it is hard to compute k1 and k2 value in a previous session even if the current secrets are known. To analyze the protocol behavior in denial-of-service attacks, we construct the model using the same configuration as shown in Figure 4.11. The model is simulated and the results are presented in Figure 6.2.

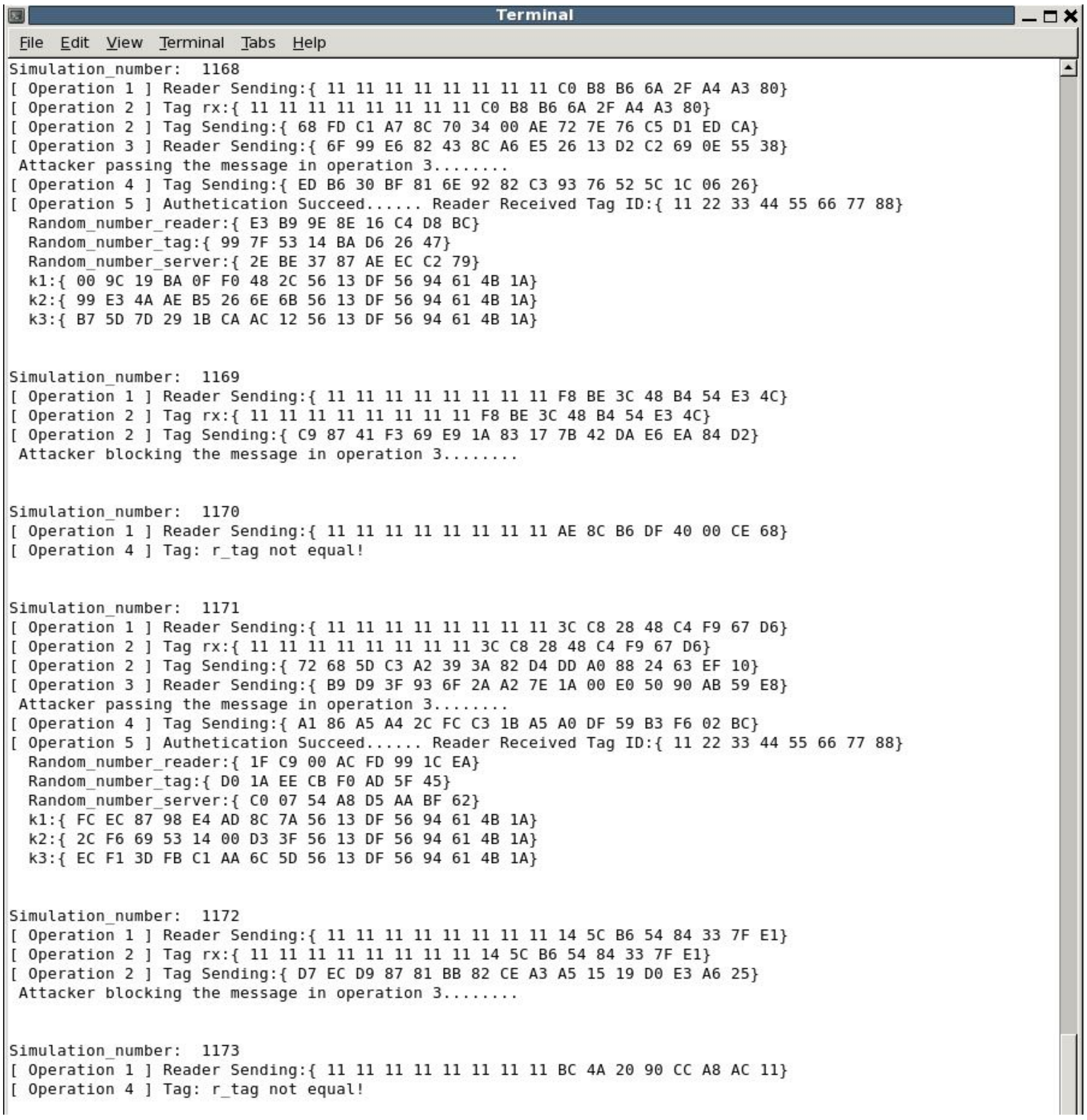


```
Simulation  1
1) Reader Sending:{ 80 1C 58 8F 6C C5 7B C4 6B B8 D5 56 A6 08 F7 B5}
2) Tag Sending:{ 39 7D CE BE B4 B7 E6 69 73 CE 0F 18 6C B4 6F B6}
3) Reader Received:{ 39 7D CE BE B4 B7 E6 69 73 CE 0F 18 6C B4 6F B6}
   ID={ 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00}
   k1={ A3 05 DB 0B FD 19 A3 FA 62 E1 E9 E4 10 4E CE 27}
   k2={ B8 9B DE E6 DB 68 8C 2B 78 31 0B BD 4C 73 42 CD}


Simulation  2
1) Reader Sending:{ FF B0 C6 5D 87 65 8F 64 88 3E B4 DC 12 A1 E7 A4}
2) Tag Sending:{ 41 31 9E E3 45 F1 85 A8 70 E7 B5 82 48 7C 09 72}
Attacker starts denial-of-service attack.....


Simulation  3
1) Reader Sending:{ FF B0 C6 5D 87 65 8F 64 88 3E B4 DC 12 A1 E7 A4}


Simulation  4
1) Reader Sending:{ FF B0 C6 5D 87 65 8F 64 88 3E B4 DC 12 A1 E7 A4}
```

Figure 6.2 Simulation of Denial-of-Service Attack in Toiruul and Lee's Protocol

The first simulation is a normal session where the system is not attacked. When the attacker starts the denial-of-service attack at the second session, the attacker attempts to block the tag reply message and the reader cannot receive the tag reply. When the reader tries to start reading operation again it cannot identify the tag because the tag's secrets are already updated but the reader's secrets are not. We also simulated all the other attack models using SystemC modeling, but we did not observe any other successful attacks. Thus, we can conclude that Toiruul and Lee's protocol [34] is not secure against denial-of-service attacks and it is also vulnerable in backwards security.

To evaluate the security level of Kim et al.'s protocol [38], we simulate the protocol in all the attack models in SystemC modeling and examine the results. The protocol simulation results of the denial-of-service attack are shown in Figure 6.3.



```
Simulation_number:  1168
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 C0 B8 B6 6A 2F A4 A3 80}
[ Operation 2 ] Tag rx:{ 11 11 11 11 11 11 11 11 C0 B8 B6 6A 2F A4 A3 80}
[ Operation 2 ] Tag Sending:{ 68 FD C1 A7 8C 70 34 00 AE 72 7E 76 C5 D1 ED CA}
[ Operation 3 ] Reader Sending:{ 6F 99 E6 82 43 8C A6 E5 26 13 D2 C2 69 0E 55 38}
 Attacker passing the message in operation 3........
[ Operation 4 ] Tag Sending:{ ED B6 30 BF 81 6E 92 82 C3 93 76 52 5C 1C 06 26}
[ Operation 5 ] Authetication Succeed...... Reader Received Tag ID:{ 11 22 33 44 55 66 77 88}
  Random_number_reader:{ E3 B9 9E 8E 16 C4 D8 BC}
  Random_number_tag:{ 99 7F 53 14 BA D6 26 47}
  Random_number_server:{ 2E BE 37 87 AE EC C2 79}
  k1:{ 00 9C 19 BA 0F F0 48 2C 56 13 DF 56 94 61 4B 1A}
  k2:{ 99 E3 4A AE B5 26 6E 6B 56 13 DF 56 94 61 4B 1A}
  k3:{ B7 5D 7D 29 1B CA AC 12 56 13 DF 56 94 61 4B 1A}


Simulation_number:  1169
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 F8 BE 3C 48 B4 54 E3 4C}
[ Operation 2 ] Tag rx:{ 11 11 11 11 11 11 11 11 F8 BE 3C 48 B4 54 E3 4C}
[ Operation 2 ] Tag Sending:{ C9 87 41 F3 69 E9 1A 83 17 7B 42 DA E6 EA 84 D2}
 Attacker blocking the message in operation 3........


Simulation_number:  1170
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 AE 8C B6 DF 40 00 CE 68}
[ Operation 4 ] Tag: r_tag not equal!


Simulation_number:  1171
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 3C C8 28 48 C4 F9 67 D6}
[ Operation 2 ] Tag rx:{ 11 11 11 11 11 11 11 11 3C C8 28 48 C4 F9 67 D6}
[ Operation 2 ] Tag Sending:{ 72 68 5D C3 A2 39 3A 82 D4 DD A0 88 24 63 EF 10}
[ Operation 3 ] Reader Sending:{ B9 D9 3F 93 6F 2A A2 7E 1A 00 E0 50 90 AB 59 E8}
 Attacker passing the message in operation 3........
[ Operation 4 ] Tag Sending:{ A1 86 A5 A4 2C FC C3 1B A5 A0 DF 59 B3 F6 02 BC}
[ Operation 5 ] Authetication Succeed...... Reader Received Tag ID:{ 11 22 33 44 55 66 77 88}
  Random_number_reader:{ 1F C9 00 AC FD 99 1C EA}
  Random_number_tag:{ D0 1A EE CB F0 AD 5F 45}
  Random_number_server:{ C0 07 54 A8 D5 AA BF 62}
  k1:{ FC EC 87 98 E4 AD 8C 7A 56 13 DF 56 94 61 4B 1A}
  k2:{ 2C F6 69 53 14 00 D3 3F 56 13 DF 56 94 61 4B 1A}
  k3:{ EC F1 3D FB C1 AA 6C 5D 56 13 DF 56 94 61 4B 1A}


Simulation_number:  1172
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 14 5C B6 54 84 33 7F E1}
[ Operation 2 ] Tag rx:{ 11 11 11 11 11 11 11 11 14 5C B6 54 84 33 7F E1}
[ Operation 2 ] Tag Sending:{ D7 EC D9 87 81 BB 82 CE A3 A5 15 19 D0 E3 A6 25}
 Attacker blocking the message in operation 3........


Simulation_number:  1173
[ Operation 1 ] Reader Sending:{ 11 11 11 11 11 11 11 11 BC 4A 20 90 CC A8 AC 11}
[ Operation 4 ] Tag: r_tag not equal!
```

Figure 6.3 Simulation of Denial-of-Service Attack in Kim et al.'s Protocol

It is found that if the denial-of-service attack happens in the third step, the tag is kept in a state that it is waiting for the next message to come from the reader. The reader does not receive any response from the tag and starts the query from the beginning again. At this point the tag could not response with the right message the reader is waiting for. Although the reader must start the reading operation from the beginning, it cannot identify the tag correctly after the attack. Therefore, we can conclude that this protocol is weak against denial-of-service attacks. We also implemented other security attack models, but no other successful attacks are observed. If the current secrets of a tag are disclosed, the tag's previous and subsequent responses are predictable because its secrets are not updated. Although random number generators are used, the values of Rtag, Rreader and Rserver can be computed by intercepting the messages between the reader and the tag. Therefore, it is also weak in both forward security and backward security. We summarize the security level comparison of our proposed protocol with Toiruul and Lee's and Kim et al.'s protocols as shown in Table 6.1.

Table 6.1 Security Level Comparison

	Forward Security	**Backward Security**	**Replay Attack Resistance**	**Cloning Attack Resistance**	**Denial-of -Service Attack Resistance**	**Man-in-the-Middle Attack Resistance**	**Active Scanning Attack Resistance**
Our Proposal	Strong	Strong	Strong	Strong	Strong	Strong	Strong
Toiruul and Lee 's Protocol [34]	Strong	Weak	Strong	Strong	Weak	Strong	Strong
Kim et al.'s Protocol [38]	Weak	Weak	Strong	Strong	Weak	Strong	Strong

6.2 Performance Analysis

To evaluate the performance of a protocol, its efficiency must be measured using different parameters. If two protocols provide the same level of security, the protocol using less resource is obviously more efficient. In this section, we present the performance analysis of Toiruul and Lee's and Kim et al.'s protocols [34, 38] and then compare our proposal with them. The main parameters we will use in this analysis include code size (tag, reader/server), communication cost, execution time, scalability and the type of cipher used.

6.2.1 Code Size

We have implemented our protocol, Toiruul and Lee's and Kim et al.'s protocols in Altera Nios-II DE2 development environment. Using the same methodology as shown in chapter 5, a multiprocessor application consisting of only a reader module and a tag module is implemented as a Nios-II system in the same FPGA device. The reader and tag codes for each protocol are applied to the same Nios-II multiprocessor platform to evaluate their efficiency. To facilitate prototyping, the System-on-a-Programmable-Chip (SOPC) Builder tool is used to generate the multiprocessor system as shown in Figure 6.4. The reader and the tag modules are implemented as 32-bit RISC Nios-II processors (economic type) and the top level configuration is designed using VHDL hardware description language. The communication between reader and tag processors are also realized using parallel input/output (PIO) ports. As illustrated in Figure 6.4, the **reader_out_en**, **tag_in_en**, **tag_out_en** and **reader_in_en** signals are 8-bit PIO ports and the **reader_out_data**, **tag_in_data**, **tag_out_data** and **reader_in_data** signals are 32-bit PIO ports. The signals of **reader_out_en**, **reader_out_data**, **tag_out_en** and **tag_out_data** are output type only while the signals of **tag_in_en**, **tag_in_data**, **reader_in_en** and **reader_in_data** are of input type.

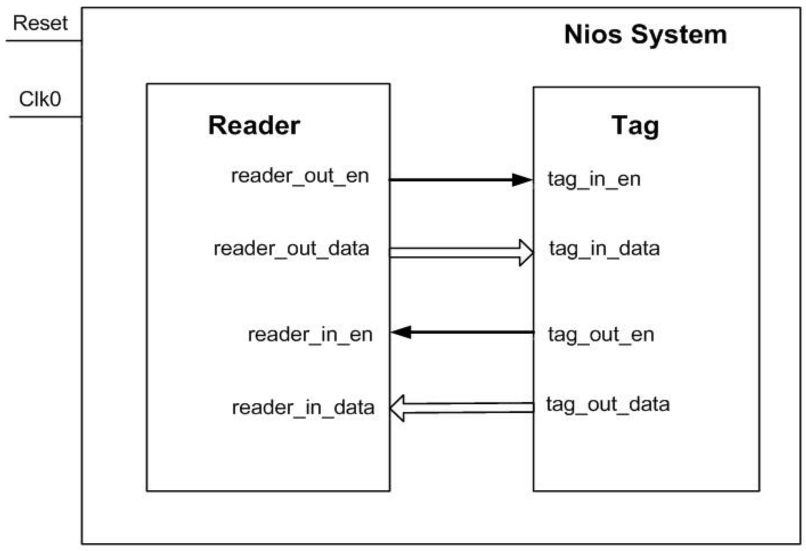


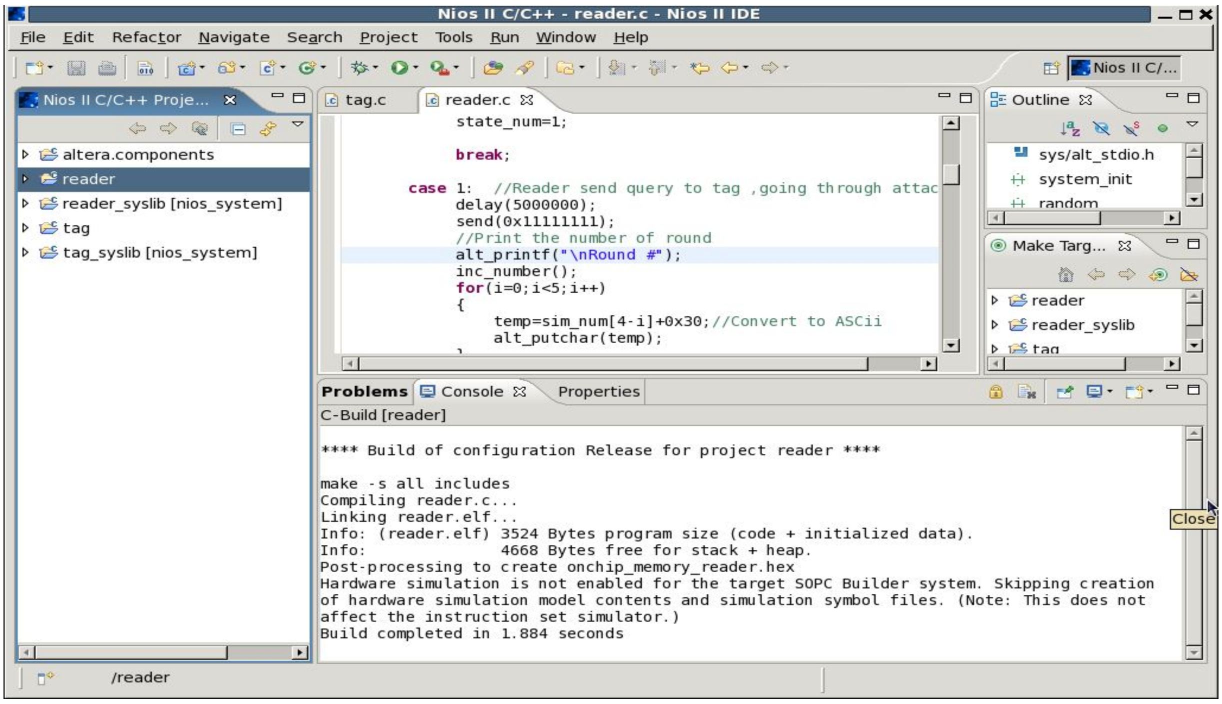
Figure 6.4 Nios-II System with Reader and Tag CPUs

JTAG UART is added to download codes to processors and to serve as a user interface. On-chip memory (RAM) is included to store code and data. After the Nios-II system is generated completely by the SOPC builder, we only need to add connections between PIO ports using VHDL hardware description language. The system architecture is compiled and uploaded into the FPGA on the DE2 board using the programmer utility available in Quartus II design environment.

Implemented as C application, the reader and the tag's protocol source codes use only a reduced subset of device driver and a small C library so that the size of the code is optimized. The communication between Nios-II CPUs is implemented as following C-like pseudo code:

```
Define  DELAY_CONSTANT   T
Send_Function (Value) {
    Data_Out_Port = Value;
    Delay(T);
    Set_Data_Out_En;
    Delay(T);
    Clear_Data_Out_En;
    Clear_Data_Out_Port;
    Exit;
}
Receive_Function (Value) {
   While(Timeout = False) {
        If ( Detected Rising Edge) {
         Receive_Data = Data_In_Port;
         Exit;
      }
   }
   Set_Fail_To_Receive_Flag;
   Exit;
}
```

A screen snapshot of the reader and tag codes after being compiled successfully are displayed in Figures 6.5 and 6.6 respectively.

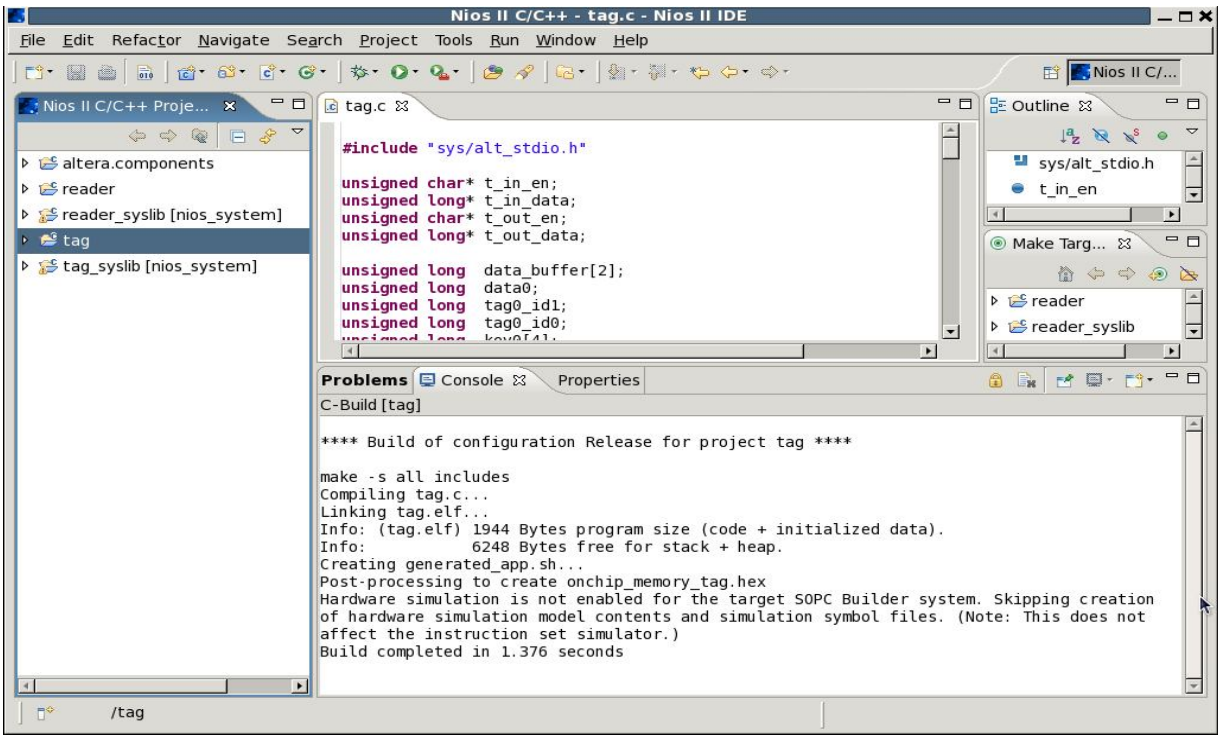


```
              state_num=1;

              break;

          case 1:  //Reader send query to tag ,going through attac
              delay(5000000);
              send(0x11111111);
              //Print the number of round
              alt_printf("\nRound #");
              inc_number();
              for(i=0;i<5;i++)
              {
                  temp=sim_num[4-i]+0x30;//Convert to ASCii
                  alt_putchar(temp);
```

C-Build [reader]

```
**** Build of configuration Release for project reader ****

make -s all includes
Compiling reader.c...
Linking reader.elf...
Info: (reader.elf) 3524 Bytes program size (code + initialized data).
Info:              4668 Bytes free for stack + heap.
Post-processing to create onchip_memory_reader.hex
Hardware simulation is not enabled for the target SOPC Builder system. Skipping creation
of hardware simulation model contents and simulation symbol files. (Note: This does not
affect the instruction set simulator.)
Build completed in 1.884 seconds
```

Figure 6.5 Compilation of the Reader Code



```
#include "sys/alt_stdio.h"

unsigned char* t_in_en;
unsigned long* t_in_data;
unsigned char* t_out_en;
unsigned long* t_out_data;

unsigned long  data_buffer[2];
unsigned long  data0;
unsigned long  tag0_id1;
unsigned long  tag0_id0;
```

```
**** Build of configuration Release for project tag ****

make -s all includes
Compiling tag.c...
Linking tag.elf...
Info: (tag.elf) 1944 Bytes program size (code + initialized data).
Info:           6248 Bytes free for stack + heap.
Creating generated_app.sh...
Post-processing to create onchip_memory_tag.hex
Hardware simulation is not enabled for the target SOPC Builder system. Skipping creation
of hardware simulation model contents and simulation symbol files. (Note: This does not
affect the instruction set simulator.)
Build completed in 1.376 seconds
```

Figure 6.6 Compilation of the Tag Code

We observe that the reader code size is 3524 bytes and the tag code size is 1944 bytes. The same method is applied to determine the code size of the other two protocols. As for Toiruul and Lee's protocol, the tag code size is 5700 bytes while the reader code size is 7836 bytes. For Kim et al.'s protocol, the tag code size is 6440 bytes while the reader code size is 9544 bytes. The reader and the tag code size for each protocol are summarized as shown in Table 6.2.

Table 6.2 Code Size Comparison

	Our Proposal		**Toiruul and Lee's Protocol [34]**		**Kim et al.'s Protocol [38]**	
	Reader	**Tag**	**Reader**	**Tag**	**Reader**	**Tag**
Code Size	3523	1944	7836	5700	9544	6400

6.2.2 Communication Cost

In order to evaluate the communication cost of each protocol, we can assume that a reader needs to authenticate “n” number of tags. In our protocol, when a reader query tags, it needs to send out only one query message for all the tags to respond, therefore the communication cost of the first step is only 64 bits. If “n” tags are responding, then “n” tags’ response consist of 64n bits. Therefore, the communication cost in the second step is 64n bits. Similarly, it is concluded that the communication cost for step 3 and step 4 are all 64n bits each.

As for the Toiruul and Lee’s protocol communication cost, we used the same method to analyze. When a reader queries “n” tags, it has to initiate “n” messages in step one. However, since the output of encryption algorithm AES is 128 bits, the communication cost of the first step is 128n bits. In the second step, n tags send back 128n bits to the reader, making the total communication cost to be 256n bits.

The Kim et al.’s protocol consists of four steps in over-the-air communication channel. If a reader queries n tags, the communication cost in each step is 128n bits due to the message size of AES encryption algorithm. The comparison of communication cost is summarized in Table 6.3. It is demonstrated that our scheme is also more efficient than the other two schemes in terms of communication cost, especially when reading multiple tags.

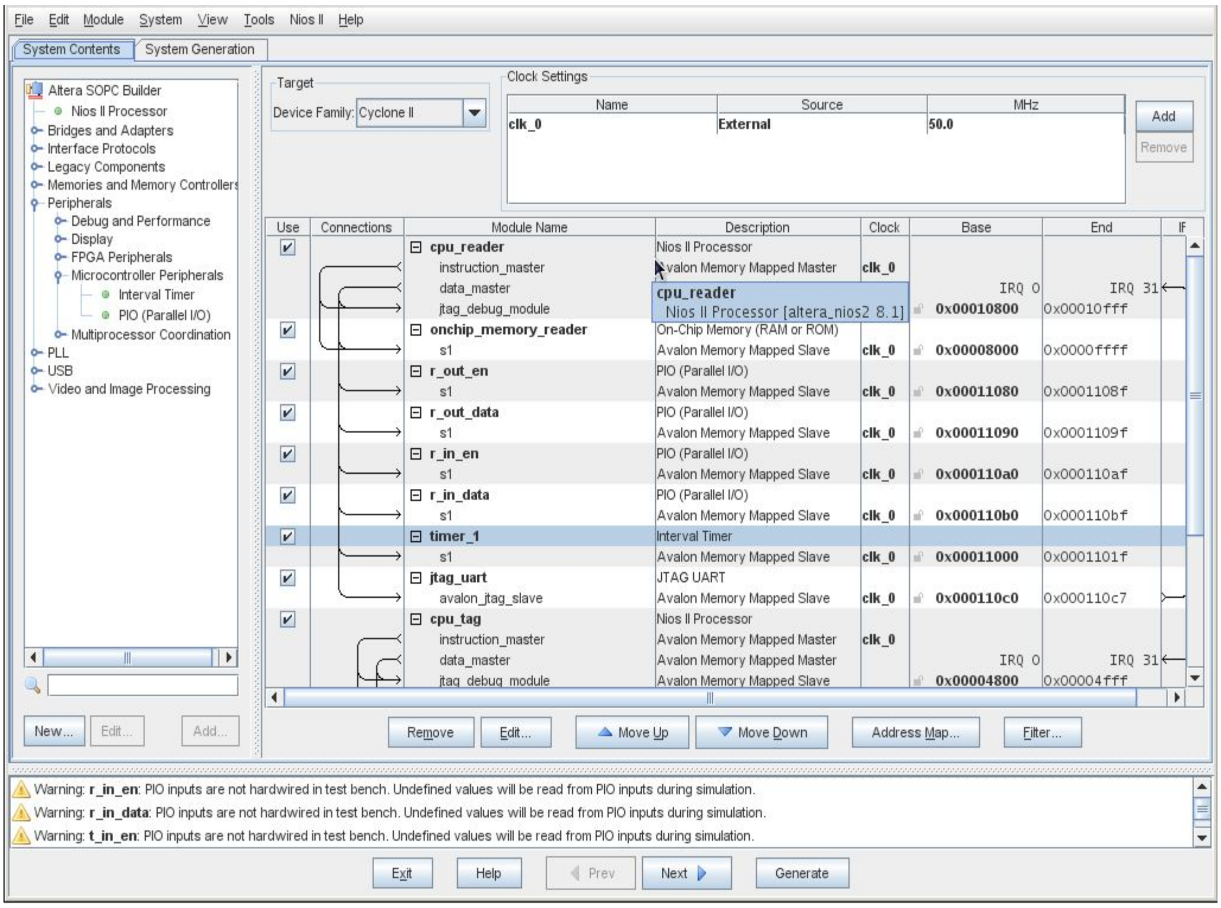
Table 6.3 Communication Cost Comparison

	Our Proposal	**Toiruul and Lee’s Protocol [34]**	**Kim et al.’s Protocol [38]**
Communication Cost	64 + 3 x 64n = 64 +192n bits	2 x 128n = 256n bits	4 x 128n = 512n bits

6.2.3 Number of Clock Cycles

Generally speaking, the number of clock cycles required for executing a protocol is a critical measurement because it is proportional to power consumption when the clock frequency remains unchanged.

Measuring the number of clock cycles can be achieved using a timestamp timer peripheral. First of all, a timestamp timer, timer_1 is added into the Nios-II system for the reader module in our protocol. The screen snapshot of the reader system in an SOPC Builder is shown in Figure 6.7.



Device Family: Cyclone II

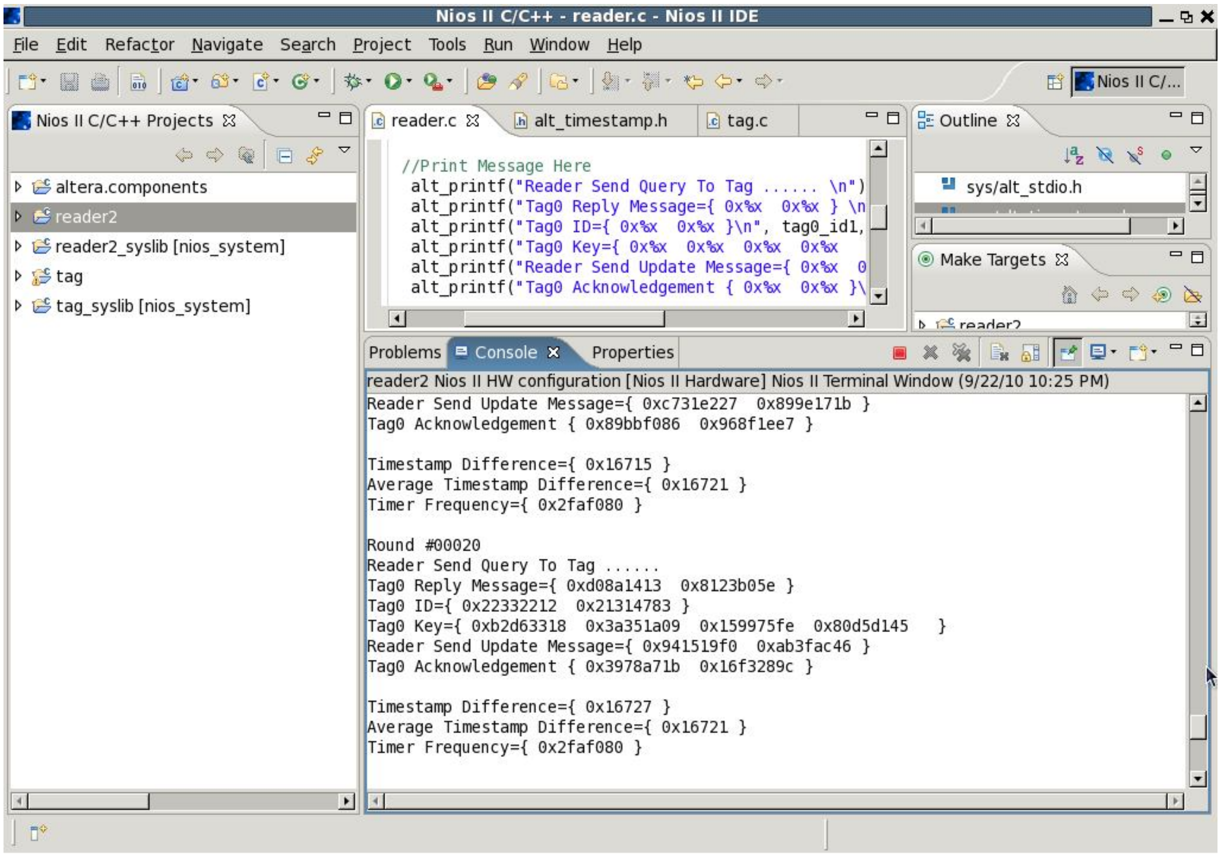
Name	Source	MHz
clk_0	External	50.0

Use	Module Name	Description	Clock	Base	End
✓	cpu_reader	Nios II Processor			
	instruction_master	Avalon Memory Mapped Master	clk_0		
	data_master	cpu_reader		IRQ 0	IRQ 31
	jtag_debug_module	Nios II Processor [altera_nios2 8.1]		0x00010800	0x00010fff
✓	onchip_memory_reader	On-Chip Memory (RAM or ROM)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x0000ffff
✓	r_out_en	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00011080	0x0001108f
✓	r_out_data	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x00011090	0x0001109f
✓	r_in_en	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x000110a0	0x000110af
✓	r_in_data	PIO (Parallel I/O)			
	s1	Avalon Memory Mapped Slave	clk_0	0x000110b0	0x000110bf
✓	timer_1	Interval Timer			
	s1	Avalon Memory Mapped Slave	clk_0	0x00011000	0x0001101f
✓	jtag_uart	JTAG UART			
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x000110c0	0x000110c7
✓	cpu_tag	Nios II Processor			
	instruction_master	Avalon Memory Mapped Master	clk_0		
	data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31
	jtag_debug_module	Avalon Memory Mapped Slave		0x00004800	0x00004fff

```
Warning: r_in_en: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: r_in_data: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: t_in_en: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
```

Figure 6.7 Reader System with Timestamp Timer

It is selected as a full feature option timer and is specified as timestamp timer in the project system library. The first timestamp is obtained at the beginning of the protocol execution. After the operation is finished, the second timestamp is recorded. The execution time of each session is measured by calculating the difference of the two timestamps. For better accuracy, the timestamp difference is measured 20 times and the average value is calculated for each evaluation. Since the communication between the reader and the tag also takes time, we have to measure the timestamp difference with different delay constant T. The delay constant is initially defined as 60 and the timestamp difference is acquired as shown in Figure 6.8.



```
//Print Message Here
 alt_printf("Reader Send Query To Tag ...... \n")
 alt_printf("Tag0 Reply Message={ 0x%x  0x%x } \n
 alt_printf("Tag0 ID={ 0x%x  0x%x }\n", tag0_id1,
 alt_printf("Tag0 Key={ 0x%x  0x%x  0x%x  0x%x
 alt_printf("Reader Send Update Message={ 0x%x  0
 alt_printf("Tag0 Acknowledgement { 0x%x  0x%x }\
```

reader2 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/22/10 10:25 PM)

```
Reader Send Update Message={ 0xc731e227  0x899e171b }
Tag0 Acknowledgement { 0x89bbf086  0x968f1ee7 }

Timestamp Difference={ 0x16715 }
Average Timestamp Difference={ 0x16721 }
Timer Frequency={ 0x2faf080 }

Round #00020
Reader Send Query To Tag ......
Tag0 Reply Message={ 0xd08a1413  0x8123b05e }
Tag0 ID={ 0x22332212  0x21314783 }
Tag0 Key={ 0xb2d63318  0x3a351a09  0x159975fe  0x80d5d145   }
Reader Send Update Message={ 0x941519f0  0xab3fac46 }
Tag0 Acknowledgement { 0x3978a71b  0x16f3289c }

Timestamp Difference={ 0x16727 }
Average Timestamp Difference={ 0x16721 }
Timer Frequency={ 0x2faf080 }
```

Figure 6.8 Timestamp Difference Measured in our Protocol

It may be noticed that the timer frequency is 0x2faf080 in hexadecimal value, which is equivalent to 50MHz in decimal number. The average timestamp difference is 0x16721 and is equal to 91937 clock cycles. Since each clock cycle is 0.02μs, it is computed that the average timestamp difference is 1.838 milliseconds.

Since communication between reader processor and tag processor takes certain amount of time, we can only get the protocol execution clock cycle by calculating the ideal execution time when T approaches zero. In fact, if T is set below five in the experiment, the protocol cannot run successfully. Table 6.4 shows the timestamp difference in hexadecimal format with different T values defined in the source code. The execution time is also converted and shown in the bracket beside.

Table 6.4 Timestamp Difference Measured in Our Protocol

Delay Constant T	Timestamp Difference
60	0x16721 (1.838 ms)
50	0x14dda (1.709 ms)
40	0x13482 (1.580 ms)
30	0x11b34 (1.450 ms)
20	0x101d1 (1.320 ms)
10	0xe884 (1.190 ms)

A chart is plotted as shown in Figure 6.9 to show the relation between the timestamp difference and different delay constant T values. As it is expected, the pattern is linear.

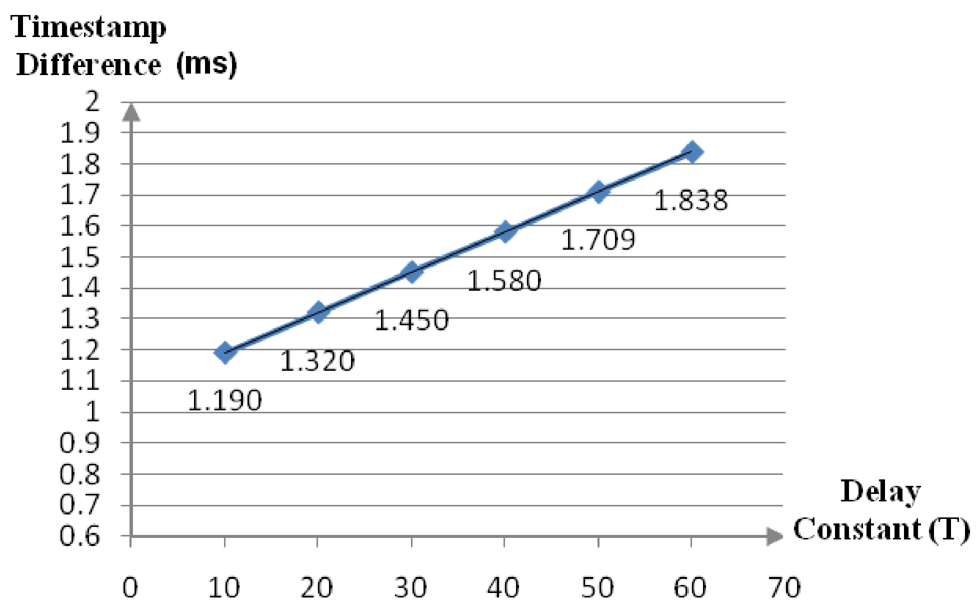
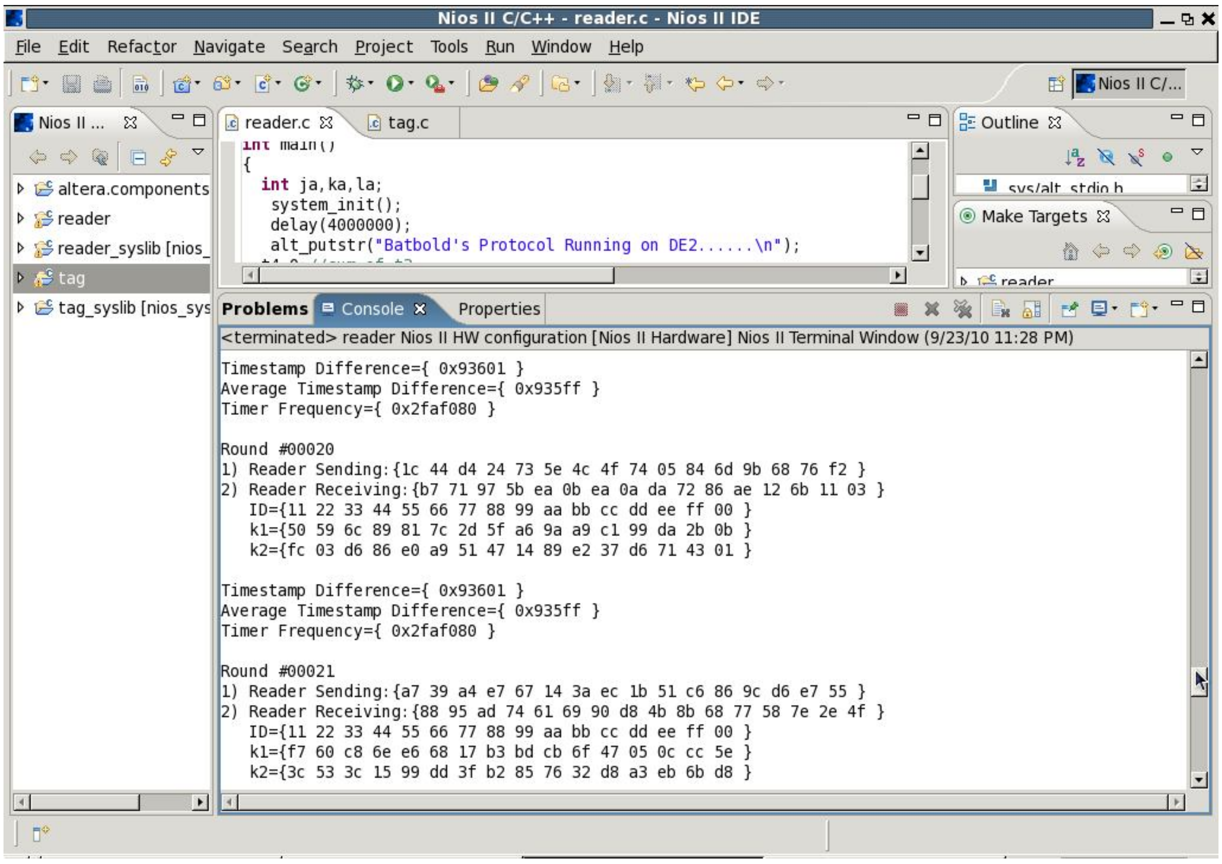


Figure 6.9 Timestamp Difference in Our Protocol

It shows that as T decreases, the timestamp difference also decreases linearly and the value drops by 0.13ms whenever T is reduced by 10. We can conclude that in an ideal condition where the communication delay is zero, the execution time of our protocol is 1.060 ms or about 53000 clock cycles.

We apply the same method to analyze the number of clock cycles for Toiruul and Lee's Protocol. Initially, the delay constant is defined as 60 and the average timestamp difference is shown in Figure 6.10. Then the timestamp difference using different delay constants is obtained as shown in Table 6.5.



```
int main()
{
  int ja,ka,la;
   system_init();
   delay(4000000);
   alt_putstr("Batbold's Protocol Running on DE2......\n");
```

```
<terminated> reader Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/23/10 11:28 PM)
Timestamp Difference={ 0x93601 }
Average Timestamp Difference={ 0x935ff }
Timer Frequency={ 0x2faf080 }

Round #00020
1) Reader Sending:{1c 44 d4 24 73 5e 4c 4f 74 05 84 6d 9b 68 76 f2 }
2) Reader Receiving:{b7 71 97 5b ea 0b ea 0a da 72 86 ae 12 6b 11 03 }
   ID={11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff 00 }
   k1={50 59 6c 89 81 7c 2d 5f a6 9a a9 c1 99 da 2b 0b }
   k2={fc 03 d6 86 e0 a9 51 47 14 89 e2 37 d6 71 43 01 }

Timestamp Difference={ 0x93601 }
Average Timestamp Difference={ 0x935ff }
Timer Frequency={ 0x2faf080 }

Round #00021
1) Reader Sending:{a7 39 a4 e7 67 14 3a ec 1b 51 c6 86 9c d6 e7 55 }
2) Reader Receiving:{88 95 ad 74 61 69 90 d8 4b 8b 68 77 58 7e 2e 4f }
   ID={11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff 00 }
   k1={f7 60 c8 6e e6 68 17 b3 bd cb 6f 47 05 0c cc 5e }
   k2={3c 53 3c 15 99 dd 3f b2 85 76 32 d8 a3 eb 6b d8 }
```

Figure 6.10 Timestamp Difference Measured in Toiruul and Lee's Protocol

Table 6.5 Timestamp Difference Measured in Toiruul and Lee's Protocol

Delay Constant T	Timestamp Difference
60	0x935ff (12.073 ms)
50	0x91f83 (11.958 ms)
40	0x90909 (11.843 ms)
30	0x8f289 (11.728 ms)
20	0x8dc07 (11.612 ms)
10	0x8c587 (11.497 ms)

A chart is plotted as presented in Figure 6.11 to show the relation between the timestamp difference and different delay constant T values.

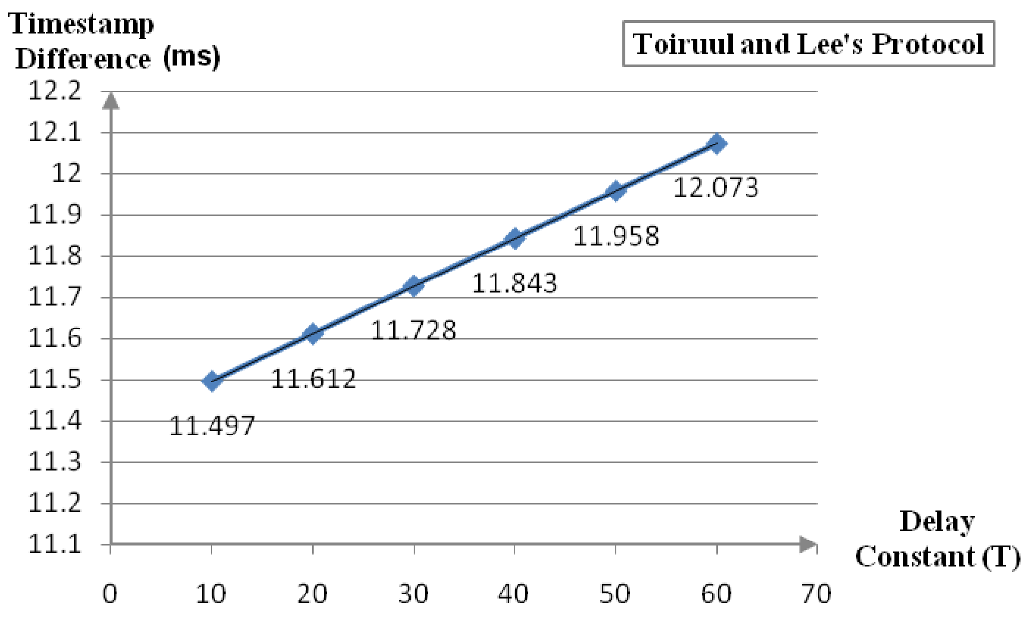
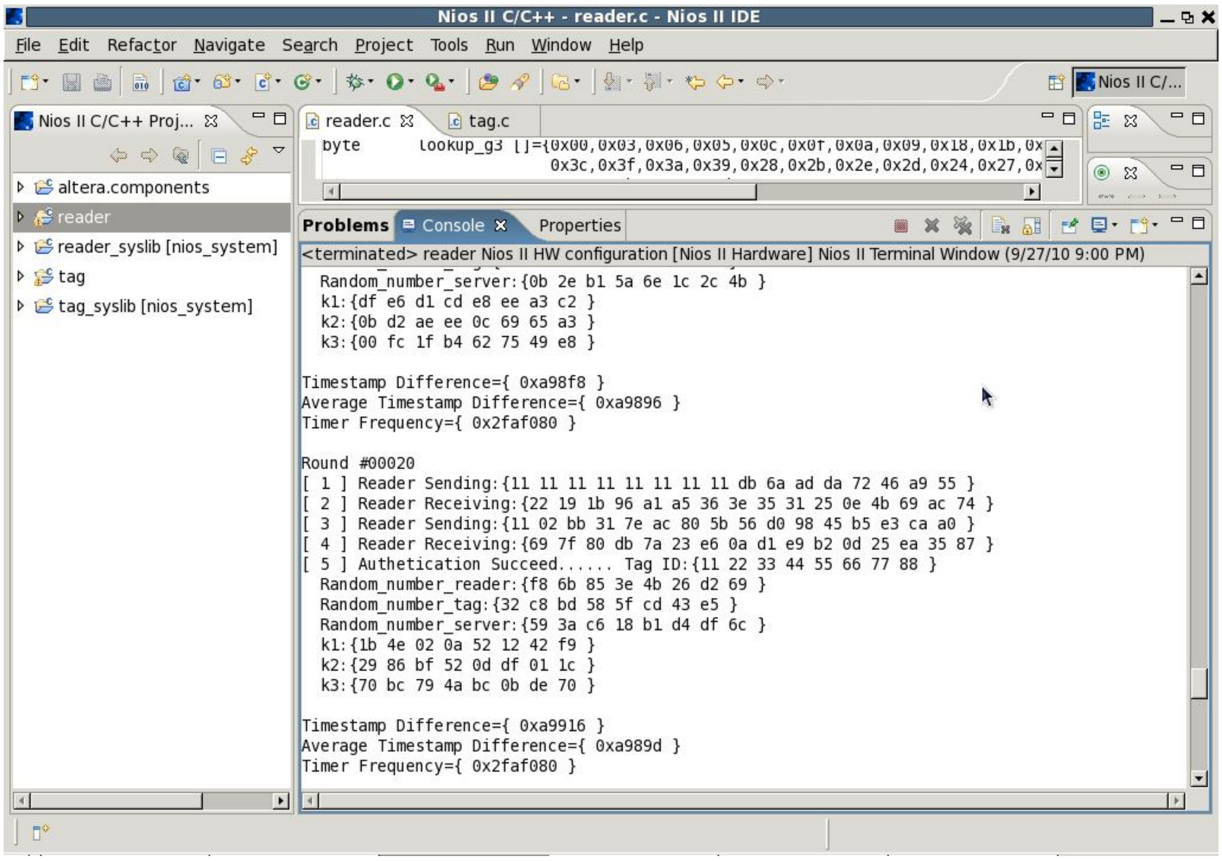


Figure 6.11 Timestamp Difference in Toiruul and Lee's Protocol

It appears that as T decreases, the timestamp difference also decreases linearly. The average value drops by 0.115ms whenever T is reduced by 10. It can be observed that if T approaches zero, the timestamp difference converges to 11.382ms. Therefore, in an ideal condition where communication delay is zero, the execution time for Toiruul and Lee's protocol is 11.282 or about 569100 clock cycles.

We use the same method to examine the number of clock cycles for Kim et al.'s protocol. When the delay constant T is defined as 60, the average timestamp difference is shown in Figure 6.12. The timestamp difference with different delay constant values is also obtained as shown in Table 6.6.


Nios II C/C++ - reader.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/...
Nios II C/C++ Proj...
altera.components
reader
reader_syslib [nios_system]
tag
tag_syslib [nios_system]
reader.c
tag.c
Problems
Console
Properties

```
byte      lookup_g3 []={0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x
                        0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x
```

```
<terminated> reader Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (9/27/10 9:00 PM)
  Random_number_server:{0b 2e b1 5a 6e 1c 2c 4b }
  k1:{df e6 d1 cd e8 ee a3 c2 }
  k2:{0b d2 ae ee 0c 69 65 a3 }
  k3:{00 fc 1f b4 62 75 49 e8 }

Timestamp Difference={ 0xa98f8 }
Average Timestamp Difference={ 0xa9896 }
Timer Frequency={ 0x2faf080 }

Round #00020
[ 1 ] Reader Sending:{11 11 11 11 11 11 11 11 db 6a ad da 72 46 a9 55 }
[ 2 ] Reader Receiving:{22 19 1b 96 a1 a5 36 3e 35 31 25 0e 4b 69 ac 74 }
[ 3 ] Reader Sending:{11 02 bb 31 7e ac 80 5b 56 d0 98 45 b5 e3 ca a0 }
[ 4 ] Reader Receiving:{69 7f 80 db 7a 23 e6 0a d1 e9 b2 0d 25 ea 35 87 }
[ 5 ] Authetication Succeed...... Tag ID:{11 22 33 44 55 66 77 88 }
  Random_number_reader:{f8 6b 85 3e 4b 26 d2 69 }
  Random_number_tag:{32 c8 bd 58 5f cd 43 e5 }
  Random_number_server:{59 3a c6 18 b1 d4 df 6c }
  k1:{1b 4e 02 0a 52 12 42 f9 }
  k2:{29 86 bf 52 0d df 01 1c }
  k3:{70 bc 79 4a bc 0b de 70 }

Timestamp Difference={ 0xa9916 }
Average Timestamp Difference={ 0xa989d }
Timer Frequency={ 0x2faf080 }
```

Figure 6.12 Timestamp Difference Measured in Kim et al.'s Protocol

Table 6.6 Timestamp Difference Measured in Kim et al.'s Protocol

Delay Constant T	Timestamp Difference
60	0xa989d (13.889ms)
50	0xa6bae (13.659ms)
40	0xa3eab (13.428ms)
30	0xa11ad (13.198ms)
20	0x9e4b4 (12.967ms)
10	0x9b7af (12.737ms)

In the same way, the relation between the timestamp difference and different delay constant T values can be plotted in a chart as shown in Figure 6.13.

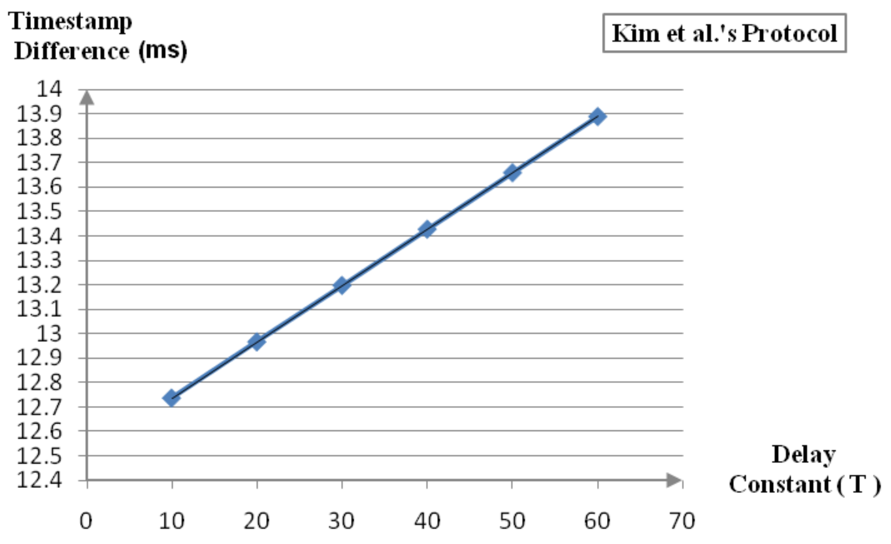


Figure 6.13 Timestamp Difference in Kim et al.'s Protocol

It can be observed that as T decreases, the timestamp difference also decreases linearly and the value drops by 0.23ms when T is reduced by 10. It is concluded that if T approaches zero, the timestamp difference converges to 12.507ms. Therefore, if there is no communication delay, Kim et al.'s protocol takes about 625350 clock cycles to execute. As discussed above, our protocol takes far less number of clock cycles than Toiruul and Lee's protocol and Kim et al.'s protocol. The comparison of each protocol's execution time and the number of clock cycles is summarized in Table 6.7.

Table 6.7 Number of Clock Cycles Comparison

	Our Proposal	**Toiruul and Lee's Protocol [34]**	**Kim et al.'s Protocol [38]**
Execution Time	1.060ms	11.382ms	12.507ms
Number of Clock Cycles	53000	569100	625350

The advantages of consuming less number of clock cycles include:

- Less power consumption
- Longer reading range
- Lower risk of being attacked

6.2.4 Cipher

The cipher used in a protocol is very important for its security as well as resource requirement. As a back-end server have more computational resource than a tag, the efficient implementation of a protocol on tags relies on lightweight ciphers. The right and suitable cipher must have a very small size in real implementation, consume less power and provide satisfactory level of security. We list the ciphers used in our protocol as compared with the other protocols as shown in Table 6.8.

Table 6.8 Cipher Usage Comparison

	Our Proposal	**Toiruul and Lee's Protocol [34]**	**Kim et al.'s Protocol [38]**
Back-end Server	XTEA	AES	AES
Tag	XTEA	AES	AES

Our proposal is based on XTEA cipher, while Toiruul and Lee's Protocol and Kim et al.'s protocol are based on AES cipher. As the platform for prototyping is Nios-II CPU based multiprocessor system and protocols are implemented as software, we cannot get the footprint of a cipher directly from compilation results of the source codes. However, both ciphers have been implemented and examined by the academic world extensively during recent years.

Feldhofer and Wolkerstorfer introduced an efficient implementation of AES algorithms in CMOS 0.35 μm process [36]. The results show that the Advanced Encryption Standard (AES-128) data path implementation achieves a current consumption of 8.5 μA at the frequency of 100 kHz. It consumes 1016 clock cycles to encrypt a 128-bit data block. It is also estimated that the hardware takes about 3595 gates.

Israsena and Wongnamkum compared the implementation of TEA algorithm with AES in CMOS 0.35 μm technology [48]. It shows that TEA encryption takes an equivalent gate count of 3872, but it consumes a much less current of 2.90 μA. XTEA algorithm was implemented in .35 μm CMOS technology and was compared with TEA in Israsena's another work [42]. It is estimated that the layout area of XTEA is 0.212 mm^2 and its maximum clock throughput is 47.5MHz. The TEA algorithm takes a similar area of 0.207 mm^2 and the maximum throughput is 53 MHz. These results demonstrate XTEA consumes less power than AES cipher in the same technology.

Jens-Peter Kaps's work made an in-depth study of ultra low power implementation of XTEA algorithm on FPGA [49]. His work introduced an efficient implementation of XTEA, called TinyXTEA-3 and it was experimented on XC3S50 Xilinx FPGA device [50]. The results show that XTEA-3 uses an area of 254 slices and takes 112 clock cycles. He has also compared with other related works about AES implementations on FPGA. The smallest AES implementation mentioned is the 8-bit AES by Good and Benaissa [51]. It achieved an area of 264 slices which is similar to XTEA-3, but it takes 3900 clock cycles. In another work by Chodowiec and Gaj, AES algorithm uses 112 clock cycles, which is the same as XTEA-3, however its area is 522 slices [52]. It is shown that XTEA is more efficient in ultra-low power and needs less resource than AES algorithm.

Therefore, our proposal selects XTEA as the best candidate of cipher because of its high efficiency proved on both CMOS and FPGA technologies.

As Kim et al.'s protocol is designed based on AES algorithm, using XTEA cipher instead of AES will make the key operations inefficient. As XTEA's message size is 64 bits, the size of Rserver, Rtag, Rreader, M, MOR and ID must be reduced to 32 bits to fit into it, which is too small for a security application. It also means that as the key size of K0 is 128 bits, the generated key K1, K2 and K3 will always have 96 bits unchanged. For the same reason, in Toiruul and Lee's protocol, the secrets of K1 and K2 will also always have 64 bits remained the same if XTEA cipher is applied. These results are not the intention of the original authors as the secrets are not sufficiently altered in every new session.

6.2.5 Scalability

In this subsection, we compare our scheme with Toiruul and Lee's Protocol and Kim et al.'s protocol in terms of scalability. In our protocol, if a reader initiates a reading operation, it just needs to issue a query message. The back-end server compares the tag reply with the list of expected reply from tags. The search time do not increase significantly with more tags added into the system. In Toiruul and Lee's Protocol, the reader relies on other anti-collision protocols to singularize a tag out of many. Therefore, it is considered to be scalable as well. In Kim et al.'s protocol, in order to initiate a reading operation, a reader has to know the tag's mask bit secret M in advance. If a reader does not know the value, a brute force search for known M values must be performed. As the population of tags grows, it will become very time consuming to identify a tag. Obviously, this scheme is not scalable. Table 6.9 summarizes the comparison of each protocol's scalability.

Table 6.9 Scalability Analysis

	Our Proposal	**Toiruul and Lee 's Protocol [34]**	**Kim et al.'s Protocol [38]**
Scalability	Yes	Yes	No

From the performance analysis shown above, our scheme has demonstrated to be more efficient in terms of code size, communication cost, number of clock cycles, cipher used and scalability. Achieving a high security level does not have to come with a high demand of resource.

Chapter 7

Conclusion and Future Works

As RFID technology attracts widespread attention from the public, its security issues need to be addressed properly. As "Moore's Law" continues to remain true and silicon chips are becoming more sophisticated with improved computation power and storage capability, stronger cryptographic primitives such as symmetric key encryption and decryption are no longer beyond the considerations for passive tags.

Our work focuses on the security and privacy aspects in the RFID applications. We have investigated a study of authentication methodologies and reviewed some cryptographic tools used in authentication protocols. Then, some related works on symmetric key cipher are studied and some proposed mutual authentication protocols used in RFID systems are reviewed.

To find a robust and efficient solution to RFID security issues, a novel RFID authentication key management protocol based on XTEA algorithm is proposed. Key update process in the protocol ensures high security level and mutual authentication is achieved. The simulation of the protocol and different attack models using SystemC based modeling proves that our scheme is safe against major attacks. Various attack models are prototyped on FPGA device to verify the robustness of our protocol. Unlike previous works, prototyping is realized efficiently on a FPGA multiprocessor platform. Overall, it is demonstrated that our scheme is safe against replay, active scanning, man-in-the-middle, denial-of-service and cloning attacks. It also provides forward security and backward security to the system.

We have compared the security and performance level of our protocol with Toiruul and Lee's and Kim et al.'s protocols. It is shown that both protocols are found to have weaknesses under denial-of-service attacks. We also analyzed the performance level in terms of code size, communication cost, execution time, scalability and the type of cipher used in the same FPGA multiprocessor platform. The comparison indicates that our scheme is more advantageous in terms of both security and performance levels.

For future work, it is necessary to implement the authentication protocol in real RFID systems to evaluate its performance. An attacker model should also be a part of the implementation in such a system so that it can launch different kinds of attacks. It will also be interesting to evaluate our scheme using WISP tag platform, because it will demonstrate the possibility for our scheme to be integrated into EPC tags in the future. The performance of power consumption and data rate can also be evaluated on tag level.

As silicon technology continues to evolve, the tag in the future will change significantly. The hardware and software co-design concept will play a key role in the efficient implementation of a RFID authentication protocol.

As discussed in our work, XTEA cipher has shown great potential in low power wireless security applications. Our scheme is proved to be helpful to create mutual authentication and trust between two entities. Therefore, some efficient applications in wireless sensor networks will be an interesting area to investigate in the future.

In our protocol, we used XTEA as cryptographic cipher because of its simplicity and high security level. As cryptographic technology keep evolving, new algorithms might come out in the future, therefore new encryptions can also be adapted into our protocol to improve its level of the security and performance.

Bibliography

[1] M. R. Rieback, B. Crispo and A. S. Tanenbaum, “The Evolution of RFID Security”, IEEE Pervasive Computing, Vol. 5, No. 1, pp. 62-69, January – March 2006

[2] M. W. Cardullo and W. L. Parks III., "Transponder Apparatus and System", United State Patent 3713148 [Online], Available at: http://patft.uspto.gov/, January 1973

[3] Elisabeth Ilie-Zudor, Zsolt Kemeny, Peter Egri and Laszlo Monostori, “The RFID Technology and its Current Applications”, in Proceedings of the 8th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises, pp. 29-36, Budapest, Hungary, September 2006

[4] Nick Long, “Frequency Bands for Short Range Devices”, in Proceedings of the IEE Seminar on Telemetry & Telematics, pp. 6/1-6/5, London, UK, April 2005

[5] R. Want, “An Introduction to RFID Technology”, IEEE Pervasive Computing, Vol. 5, No. 1, pp. 25-33, January-March 2006

[6] G. Barber and E. Tsibertzopoulos, “An Analysis of Using EPCglobal Class-1 Generation-2 RFID Technology for Wireless Asset Management”, in Proceedings of Military Communications Conference 2005, Vol. 1 , pp. 245–251, Atlantic City, New Jersey, USA, October 2005

[7] Ari Juels, “RFID Security and Privacy: A Research Survey”, IEEE Journal on Selected Areas in Communications, Vol. 24, No. 2, pp. 381-394, February 2006

[8] EPC Global, “Specification for RFID Air Interface, EPC Global (2005) EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz–960 MHz. Version 1.0.9” [Online], Available at: http://www.epcglobalinc.org

[9] M. Jantscher,.R. Ghosal, A. Grasso, and P. H. Cole, “A Security Primer, Networked RFID Systems and Lightweight Cryptography – Raising Barriers to Product Counterfeiting” , First Edition, Springer, 2008

[10] S.E. Schechter, “Quantitatively Differentiating System Security,” in Proceedings of the 1st Workshop on Economics and Information Security, Berkeley, California, USA, May 2002,

[11] S. E. Schechter, “Toward Econometric Models of the Security Risk From Remote Attacks”, IEEE Security & Privacy, Vol. 3, No. 1, pp. 40-44, January-February 2005

[12] M. Langheinrich and R. Marti, “Practical Minimalist Cryptography for RFID Privacy” IEEE Systems Journal, Vol. 1, NO. 2, December 2007

[13] M. O. Lehtonen, , F. Michahelles, and E. Fleisch, “Trust and Security in RFID-Based Product Authentication Systems”, IEEE Journal on Selected Areas in Communications, Vol. 24, No. 2, pp. 381-394, February 2006

[14] M. Lehtonen, T. Staake, F. Michahelles, and E. Fleisch, “From Identification to Authentication—A Review of RFID Product Authentication Techniques”, Presented at the Workshop on RFID Security, Graz, Austria, 2006.

[15] Z. Nochta, T. Staake, and E. Fleisch, “Product Specific Security Features Based on RFID Technology”, in Proceedings of International Symposium on Applications and the Internet Workshops , pp. 72–75, Phoenix, Arizona, USA, 2006

[16] A. J. Menezes, P. C. Oorschot and S. A. Vanstone “Handbook of Applied Cryptography”, CRC Press, 1997

[17] J. Deepakumara, H. M. Heys and R. Venkatesan, “FPGA Implementation of MD5 HASH Algorithm”, in Proceedings of Canadian Conference on Electrical and Computer Engineering 2001, Vol. 2, pp. 919 – 924, Toronto, Canada, 2001

[18] X. Wang and H. Yu, “How to Break MD5 and Other Hash Functions”, in Proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science, Vol. 3494, pp. 19-35, Springer, 2005

[19] X. Wang, Y. L. Yin and H. Yu, “Finding Collisions in the Full SHA-1”, in Proceedings of CRYPTO 2005, Lecture Notes in Computer Science, Vol. 3621, pp. 17-36, Springer, 2005

[20] National Institute of Standards and Technology, U.S. Department of Commerce, “Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family”, U.S. Office of Federal Register Notices, Vol. 72, No. 212, pp. 62212-62220, [Online], Available at: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, November 2007

[21] D. Naccache and P. Fremanteau, “Unforgeable Identification Device, Identification Device Reader and Method of Identification”, United States Patent 5434917 [Online], Available at: http://patft.uspto.gov/, July 1995

[22] D. C. Ranasinghe, D. Lim, S. Devadas, D. Abbott and P.H. Cole, “Random Numbers from Metastability and Thermal Noise”, IEE Electronic Letters, Vol. 41, No. 16, pp.13-14, 2005

[23] J. H. Anderson, “A PUF Design for Secure FPGA-Based Embedded Systems”, in Proceedings of the 15th Asia and South Pacific Design Automation Conference, pp.1-6, Taipei, January 2010

[24] J. Xie and X. Pan, “An Improved RC4 Stream Cipher”, in Proceedings International Conference on Computer Application and System Modeling, Vol. 7, pp. 156-159, Taiyuan, China, October 2010

[25] National Institute of Standards and Technology, “Advanced Encryption Standard”, Federal Information, Processing Standards Publication 197 [Online], Available at: http://csrc.nist.gov/ publications/fips/fips197/fips-197.pdf, November 2001

[26] W. E. Burr, “Selecting the Advanced Encryption Standard”, IEEE Security & Privacy, Vol. 1, No.2, pp. 43-52, March-April 2003

[27] R. M. Needham and D. J. Wheeler, “TEA, a Tiny Encryption Algorithm”, in Proceedings of the 2nd International Workshop on Fast Software Encryption, Lecture Notes in Computer Science, Vol. 1008, pp. 363-366, Springer, 1994

[28] J. Kelsey, B. Schneier and D. Wagner, “Related Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, New DES, RC2, and TEA”, in Proceedings of the 1st International Conference of Information and Communications Security, Lecture Notes in Computer Science, Vol. 1334, pp. 233-246, Springer, 1997

[29] R. M. Needham and D. J. Wheeler, "TEA Extensions", Technical Report, Computer Laboratory, University of Cambridge, England, October 1997

[30] J. Lu, “Related-key Rectangle Attack on 36 Rounds of the XTEA Block Cipher”, International Journal of Information Security, Vol. 8, No.1, pp. 1-11, Springer, February 2009

[31] S. A. Weis , S. E. Sarma , R. L. Rivest and D. W. Engels. “Security and Privacy Aspects of Low-cost Radio Frequency Identification Systems”, in Proceedings of the 1st International Conference of Security in Pervasive Computing, Lecture Notes in Computer Science, Vol. 2802, pp. 201-212, Springer, 2004

[32] M. Ohkubo, K. Suzuki and S. Kinoshita, “Cryptographic Approach to Privacy-Friendly Tags”, in Proceedings of RFID Privacy Workshop 2003, Massachusetts, USA, 2003

[33] X. Wang, X. Zhou and B. Sun, “An Improved Security Solution of RFID System”, in Proceedings International Conference on Wireless Communications, Networking and Mobile Computing, pp.2081-2084, Shanghai, China, September 2007

[34] B. Toiruul and K. Lee, “An Advanced Mutual-Authentication Algorithm Using AES for RFID Systems”, International Journal of Computer Science and Network Security, Vol. 6, No. 9, pp. 156-162, September 2006

[35] R. Juels, R. L. Rivest and M. Szydlo, “The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy”, in Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 103–111, Philadelphia, PA, USA , November 2001

[36] M. Feldhofer, S. Dominikus and J. Wolkerstorfer, “Strong Authentication for RFID System Using the AES Algorithm”, in Proceedings of the 6th International Workshop of Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science , Vol. 3156, pp. 357-370, Springer, 2004

[37] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, “AES Implementation on a Grain of Sand”, IEE Proceedings Information Security, Vol. 152, No. 1, pp. 13-20, October 2005

[38] K. Kim, K. Chung, J. Shin, H. Kang, S. Oh, C. Han and K. Ahn, “A Lightweight RFID Authentication Protocol using Step by Step Symmetric Key Change”, in Proceedings of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing, pp. 853 – 854, Chengdu, China, December 2009

[39] Open SystemC Initiative, “SystemC Version 2.0 User’s Guide” [Online], Available at: http://www. systemc.org

[40] X. Luo, K. Zheng, Y. Pan and Z. Wu, “Encryption Algorithms Comparisons for Wireless Networked Sensors”, in Proceedings IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, pp. 2081-2084, Hangzhou, China, October 2004

[41] H. Chae, D. J. Yeager, J. R. Smith and K. Fu, “Maximalist Cryptography and Computation on the WISP UHF RFID Tag”, in Proceedings Conference on RFID Security, Malaga, Spain, July 2007

[42] P. Israsena, "On XTEA-based Encryption/Authentication Core for Wireless Pervasive Communication", in Proceedings of International Symposium on Communications and Information Technologies 2006, pp. 59-62, Bangkok, Thailand, September, 2006

[43] Z. Luo, T. Chan, J. S. Li, E. Wong, W. Cheung, V. Ng and W. Fok, “Experimental Analysis of an RFID Security Protocol”, In Proceedings IEEE International Conference on e-Business Engineering, pp. 62-70, Shanghai, China, October 2006

[44] S. Gilbert, “RFID Security: Tiny Encryption Algorithm and Authentication Protocols”, Master Project, Ryerson University, Toronto, Canada, 2009

[45] L. Borghei, “FPGA-Based Smart RFID Tag With Robust Authentication Protocol”, Master Project, Ryerson University, Toronto, Canada, 2009

[46] C. Angerer, B. Knerr, M. Holzer, A. Adalan, and M. Rupp, “Flexible Simulation and Prototyping for RFID Designs”, in Proceedings of the 1st International EURASIP Workshop on RFID Technology, September 2007

[47] “Altera Development and Education Board”, [Online], Available at: http://www.altera.com/education/ univ/ materials/boards/unv-de2-board.html

[48] P. Israsena and S. Wongnamkum, “Hardware Implementation of a TEA Based Lightweight Encryption for RFID Security”, “RFID Security”, pp. 417–433, Springer, 2009

[49] Jens-Peter Kaps, “Chai-Tea, Cryptographic Hardware Implementations of xTEA”, in Proceedings of the 9th International Conference on Cryptology: Progress in Cryptology, Lecture Notes in Computer Science, Vol. 5365, pp. 363-375, Springer, 2008

[50] Spartan-3 FPGA Family Data Sheet, [Online], Available at: http://www.xilinx.com/ support/documentation/data_sheets/ds099.pdf

[51] T. Good and M. Benaissa, “AES on FPGA from the Fastest to the Smallest”, in Proceedings of the 7th International Workshop of Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, Vol. 3659, pp. 427–440, Springer, 2005

[52] P. Chodowiec and K. Gaj, “Very Compact FPGA Implementation of the AES Algorithm”, in Proceedings of Workshop on Cryptographic Hardware and Embedded Systems 2003, Lecture Notes in Computer Science, Vol. 2779, pp. 319–333, Springer, 2003

Glossary

AES	Advanced Encryption Standard
C1G2	Class-1 Generation-2
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Code
CTB	Cost to Break
DES	Data Encryption Standard
DSS	Data Security Standard
EPC	Electronic Product Code
FPGA	Field-Programmable Gate Array
HAC	Hash-based Access Control
IFF	Identification Friend or Foe
ISM	Industrial-Scientific-Medical
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
MD4	Message-Digest Algorithm 4
MD5	Message-Digest Algorithm 5
MDC	Modification Detection Codes
NIST	National Institute of Standards and Technology
NSA	National Security Agency
PIO	Parallel Input/Output
PRNG	Pseudo-random Number Generator
PUF	Physical Unclonable Function

RC4	Rivest Cipher 4
RF	Radio Frequency
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computing
RSA	Rivest-Shamir-Adelman
RTL	Register Transfer Level
SHA	Secure Hash Algorithm
SOPC	System on a Programmable Chip
SSL	Secure Socket Layer
TEA	Tiny Encryption Algorithm
TID	Tag Identification
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
UHF	Ultra High Frequency
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WISP	Wireless Identification and Sensing Platform
XOR	Exclusive OR
XTEA	eXtended Tiny Encryption Algorithm