Mutual Authentication Protocols for RFID Special Schemes

Submitted by

Sarah Hani Abu Ghazalah

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2016

Declaration

I, Sarah Hani Abu Ghazalah, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed . . . . (Sarah Hani Abu Ghazalah)
Date:

My success is only by God

Abstract

Radio Frequency IDentification (RFID) is a wireless identification technology that uses radio waves to identify tagged objects. RFID systems provide low-cost tagging capabilities for many applications such as access control systems, transportation ticketing, and supply chain management. Providing security and preserving privacy for these systems is challenging. The tags utilised in such applications are low-cost tags with limited resources that cannot afford the use of conventional cryptographic primitives. Thus, low-cost RFID tags might be vulnerable to passive attacks, such as eavesdropping, and active attacks, such as tag cloning, impersonation, replay, data de-synchronization attacks, tag data leakage, forward secrecy invasion and location tracking.

There has been considerable research into the mutual authentication of passive RFID tags to combat passive and active attacks, and in this thesis we present analysis of the prior art, which led us to make five academic research contributions.

Security is increasingly important, especially for tagging of important objects, and there are growing concerns from users about their privacy. To this end, in this thesis, we studied RFID security and privacy in several schemes, such as in RFID-enabled supply chains, RFID cloud-based scheme, and multi-tag group reading schemes. We focused on how to improve and propose RFID mutual authentication protocols in such schemes that are practical and cost effective, and satisfy the security and privacy requirements.

Lastly, we provide a formal analysis of the proposed protocols using CasperFDR and Scyther tools, along with the implementation of the proposed protocols with their performance measures.

Acknowledgement

First of all, all thanks are due to God, this thesis could not have been completed without God blessings.

I would like to express my sincere gratitude to my supervisor Prof. Konstantinos Markantonakis for the immeasurable amount of support and guidance throughout my PhD. I remember in the first meeting with my supervisor, he told me that doing PhD is like a running contest, I should keep running without stopping. I found this utterly true. Also, I would like to extend my deepest gratitude to Prof. Keith Mayes for reviewing this thesis. I am particularly in debt for Dr. Raja Naeem Akram for his insightful comments on my thesis.

During my stay at Royal Holloway, I was privileged to meet great people. I would like to thank my colleagues in the Smart Card Centre: Assad Umar, Hafizah Mansor, Mehari Msgna, Lazaros Kyrillidis, Sheila Cobourne, Danushka Jayasinghe, Benoit Ducray, Rashedul Hassan, and Iakovos Gurulian.

I am very thankful to my sponsors, King Khaled University, and the Ministry of Education in Saudi Arabia for their financial support over the years.

Thank you Mom and Dad for always supporting me, praying for me and believing in me. You are my role models. I dedicate this thesis to my lovely baby Hamad, who brought joy to my life, and to my sisters and brother. Thanks for the endless prayers of my grandmother, may her soul rest in peace.

Last but not least, a special thanks go to my husband Yahya for being patient with me during my PhD. I would not be where I am today without your help and support. I owe my success to you.

Contents

List of Figures

List of Tables

List of Abbreviations

ACK	Acknowledgement
CRC	Cyclic Redundancy Check
CSP	Communicating Sequential Process
DoS	Denial of Service
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPC	Electronic Product Code
EPCC1Gen2	EPC Class-1 Generation-2
FDR	Failure-Divergence Refinement
GE	Gate Equivalent
GHz	Gigahertz
HF	High Frequency
IC	Integrated Circuit
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
KHz	Kilohertz
LF	Low Frequency
MAC	Message Authentication Code
MHz	Megahertz
NACK	Negative Acknowledgement
PC	Protocol Control
PCB	Printed Circuit Board
PRNG	Pseudo Random Number Generator
RAM	Random Access Memory
RF	Radio Frequency
RFID	Radio Frequency IDentification
UHF	Ultra High Frequency

Chapter 1

Introduction

Contents

In this chapter, we explain the motivation behind the thesis, followed by the main contributions it makes to the design of a secure RFID system. The chapter concludes by outlining the structure of the thesis, and listing publications achieved throughout the PhD study.

1.1 Motivation

Radio Frequency Identification (RFID) is a wireless identification technology that uses radio signals to transmit data. It is used for identifying objects such as products, animals and people. These objects are embedded with a small token known as an RFID tag. In this thesis, we focus on passive RFID tags, which can be defined as wireless transponders that do not have any power of their own and only respond to the electromagnetic fields generated by the nearby reader(s). Passive RFID tags are designed to be resource-limited in order to be used with low-cost items. An RFID tag stores a unique identifier and may optionally hold additional information about the object. RFID readers communicate wirelessly with the tags to identify, read, write or update the tag's data without requiring a line-of-sight and over greater distances than other identification technologies such as barcodes [111].

Currently, the dominant identification system is the barcode, although it requires a line-of-sight scan to identify objects belonging to the same type (homogeneous). RFID technology offers advantages that surpass barcode technology [111] as follows:

- Unique identification: Each tagged object can be identified uniquely, including objects from homogeneous types.
- No line-of-sight: Tagged objects can be scanned without any requirement for a line-of-sight or a physical connection, as RFID relies on radio frequencies to transmit information rather than light as in the barcode technology.
- Simultaneous scanning: Multiple RFID tags can be presented simultaneously for reading, whereas barcodes are presented sequentially; one object at a time.
- Rewritable memory: Some RFID tag's memories can be erased and re-written with new data via the reader.

Although RFID technology provides a promising solution, security is increasingly important, especially for tagging of important objects, and there are growing concerns from users about privacy, for the following reasons:

1. The wireless communication channel between the reader and the tag may be susceptible to eavesdropping and/or manipulation. Moreover, a unique tag's identification may be tracked by an intruder, and thus the privacy of the tag's holder could be compromised.
2. An RFID tag is typically designed to be low-cost for mass distribution. This can result in tags being designed with limited memory and computing capabilities,

which do not take advantage of heavy cryptographic techniques such as asymmetric encryption.

A securely designed RFID system should take into account three main attributes, which are integrity, availability, and confidentiality. The consequences of breaching an RFID system can be severe. An intruder could eavesdrop on the communication channel and modify the transmitted data thus affecting the *integrity* of data. Moreover, an intruder might block the transmitted message from reaching targets, or the messages might be lost during transmission and this will affect the *availability* feature. Also, the data might be compromised or tracked by an intruder, leading to a breach of *confidentiality*.

Cryptography seems an inevitable tool in designing a secure RFID system. From a theoretical point of view, traditional cryptography can be an ideal approach. However, some common best-practice cryptographic approaches require more memory and/or processing power than would be feasible for cost-effective RFID tags. Therefore, researchers considered lightweight cryptographic techniques [1].

There is considerable research into the lightweight mutual authentication of RFID tags to combat passive and active attacks, and in this thesis, we present analysis of the prior art, and focus on proposing RFID mutual authentication protocols that provide adequate levels of security and privacy for several RFID schemes. This has led us to make five academic research contributions, as shown in the next section.

1.2 Main Contributions

The main contributions of the thesis are as follows:

1. Discovering potential attacks in some of the prior related work and proposing countermeasures.

2. Proposing an RFID mutual authentication protocol for low-cost RFID tags.

3. Distributing and updating tags' shared secret key between the RFID-enabled supply chain entities.

4. Protecting and preserving RFID tags' data in the cloud-based RFID systems.

5. Proving that a group of legitimate RFID tags have been scanned simultaneously in grouping-proof-based systems.

1.3 Organisation

This thesis is divided into the following chapters:

- Chapter 2 presents the background of RFID technology, and identifies the main privacy and security features required in secure RFID systems.
- Chapter 3 presents a description of the formal mechanical analysis tools such as CasperFDR and Scyther that are used to check the secrecy and authenticity of the proposed protocols. This chapter also presents the lab setup, the implementation tools, and the techniques used in the performance measurement.
- Chapter 4 shows how design flaws can be exploited to perform a data desynchronisation on one of the widely-cited RFID mutual authentication protocols proposed by Boyeon Song.
- In chapter 5, we propose a new lightweight RFID mutual authentication protocol, which builds on the strength of existing proposals and overcomes Song's protocol and other proposals' weaknesses.
- Chapter 6 discusses the use of secret sharing strategies for managing the key distribution and recovery in RFID-enabled supply chains. We point out the weaknesses found in two of the proposed solutions, and propose our enhanced scheme.
- In chapter 7, we review and enhance a recent proposed protocol regarding the security and privacy of RFID tag's data that resides in the cloud; assuming that the cloud is not trusted.
- In chapter 8, we focus on a particular RFID application called a grouping-proof, where an entity such as a reader generates a proof of simultaneous presence of two or more tagged items. We propose an offline two rounds RFID grouping-proof protocol that provides immunity against passive and active attacks on RFID protocols, and improves the current work performance.
- Chapter 9 offers concluding remarks and discusses future work.

1.4 Publications

Parts of the material presented in this thesis have been previously published in refereed conferences:

1. S. Abughazalah, K. Markantonakis, and K. Mayes. A vulnerability in the Song authentication protocol for low-cost RFID tags. In *The 28th IFIP Advances in Information and Communication Technology Security and Privacy Protection in Information Processing Systems (IFIP SEC 2013)*, editors, L. Janczewski, H. Wolfe, and S. Shenoi, volume 405, pages 102-110. Springer, 2013.

2. S. Abughazalah, K. Markantonakis, and K. Mayes. A formally verified mutual authentication protocol for low-cost RFID tags. In *International Journal of RFID Security and Cryptography*, 3(2):156-169, 2014.

3. S. Abughazalah, K. Markantonakis, and K. Mayes. Enhancing the key distribution model in the RFID-enabled supply chains. In *the 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2014)*, pages 871-878, IEEE, 2014.

4. S. Abughazalah, K. Markantonakis and K. Mayes, Secure mobile payment on NFC-enabled mobile phones formally analysed using CasperFDR. In *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2014)*, pages 422-431, IEEE, 2014.

5. S. Abughazalah, K. Markantonakis, and K. Mayes. Secure improved cloud-based RFID authentication protocol. In *The 9th International Workshop on Data Privacy Management (DPM 2015)*, editors, J. Garcia-Alfaro, J. Herrera-Joancomartí, E. Lupu, J. Posegga, A. Aldini, F. Martinelli, and N. Suri, volume 53, pages 147-164, Springer, 2015.

6. S. Abughazalah, K. Markantonakis, and K. Mayes. Two rounds RFID grouping-proof protocol, In *The 10th IEEE International Conference on RFID (IEEE RFID 2016)*, pages 193-206, IEEE, 2016.

Chapter 2

RFID Overview

Contents

In this chapter, we provide a brief discussion of the history of RFID technology. Then, we discuss RFID architecture with regard to the RFID components, the operating frequencies, the communication methods between a tag and a reader, and the anti-collision protocols. Subsequently, the regulations and standardisations governing RFID systems and the main applications of RFID technology are illustrated. The chapter concludes by outlining the concerns associated with the deployment of RFID in terms of security and privacy, and the common countermeasures.

2.1 RFID History

The roots of RFID technology can be traced back to 1901, when Guglielmo Marconi transmitted radio signals across the Atlantic to send messages. In 1934, "Radio Detection and Ranging" system, otherwise known as *Radar* was introduced, which uses radio waves to locate physical objects. Radar was used widely in the Second World War to detect incoming aircraft by sending pulses of radio energy and capturing the echoes generated from aircraft [22]. However, identifying the aircraft correctly was the main issue; they were not able to identify which aircraft belonged to whom. Thus, the British developed the first active Identification, Friend or Foe (IFF) system. They embedded a transmitter in each aircraft that allowed the radar operators and pilots to automatically distinguish friendly aircraft from enemy aircraft via the radio frequency (RF) signals [22]. The first public description of passive communication used in RFID technology was published in 1948 by Harry Stockman in "Communication by Means of Reflected Power" [23].

In 1970, wireless sensors developed by Los Alamos National Laboratory were used to track nuclear materials and to identify trucks on roads, bridges and tunnels [129]. In the same year, Los Alamos National Laboratory developed the first low frequency passive RFID transponders to track cows. A transponder encapsulated in glass could be injected under a cow's skin to check the dosages of hormones and medication given to them [129]. In 1973, passive RFID transponders was deployed to allow users to unlock a door without a key. The nearby reader powers the card to check the identity sent by the card to unlock the door. Also, in 1973, a one-dimensional (or linear) barcode design was invented by encoding the product's and brand's information onto a physical object that could be scanned by a reader [24].

Over time, IBM developed an ultra high frequency (UHF) RFID tags, which offered a longer read range, and faster data transfer. IBM used these new inventions in Wal-Mart but not for a long time. In 1999, UHF RFID tags obtained more attention when the Uniform Code Council, EAN International, Procter & Gamble and Gillette established the Auto-ID Center at Massachusetts Institute of Technology to study RFID and invent new RFID technologies [25]. Adoption of RFID grew steadily over the following years, and passive UHF tags started to be used in the supply chain to track products through their life cycle. These tags only store the tag's serial number for identification purposes, and stored in a database [129].

Between 1999 and 2003, the U.S. Department of Defence and 100 RFID global companies supported the use of RFID technology and funded the Auto-ID Center to develop this technology. Auto-ID Center opened research labs in Australia, United

Kingdom, Switzerland, USA, Japan and China to bring together RFID manufacturers, researchers, and users to develop standards, perform research, and share information [129]. Together with the EPCglobal community, both are creating the standards and assembling the building blocks needed to create an “Internet of things”.

From 2003 till today, RFID technology rapid development is being challenged by security and privacy issues. Ongoing research is directed towards the security and privacy of RFID, and the design of tag’s supporting cryptography is being developed.

According to [26], in 2015, the RFID market was worth \$10.1 billion, compared to \$9.5 billion in 2014 and \$8.8 billion in 2013. It is predicted that the RFID market will be worth \$13.2 billion by 2020. This implies that RFID technology will be a promising technology in the near future.

2.2 RFID Architecture

In this section, a brief background to the architecture of RFID systems is presented in terms of RFID components, operating frequencies, reader and tag communication methods, and reader and tag anti-collision schemes.

2.2.1 RFID Components

An RFID system consists of three main components; an RFID tag, an RFID reader, and a back-end database (server).

1. RFID tag: An RFID tag is an identification object that consists of integrated circuitry (IC) and antenna. The integrated circuit is for computation and storage purposes, and it is attached to an antenna that provides communication between the reader and the tag. It is also known as a transponder. An RFID tag can be classified based on the source of power into three categories as follows:

    - Active tag: An Active tag is a wireless transponder embedded with a battery and a transmitter enabling the tag to run the chip’s circuitry and to broadcast a signal to a reader. This tag is expensive and can be read from 100 metres or more.
    - Passive tag: A Passive tag is a wireless transponder that does not have any power of its own and only responds to the electromagnetic fields generated by the nearby reader(s). Passive RFID tag is designed to be simple and resource-limited in order to be used with low-cost items. This type of tag will be the focus of this thesis.

- Semi-passive tag: A Semi-passive tag uses a battery to run the chip's circuitry, but depends on the reader's signals for communication purposes. This ensures that a semi-passive tag is only active when queried by a reader.

2. RFID reader: An RFID reader has an RF module, a control unit, and a coupling element (antenna) to sense the presence of RFID tags and communicate with them (discussed more deeply in Section 2.2.3). It is also known as a transceiver or interrogator. RFID readers have different frequencies, and may offer a wide range of functionality. Generally, RFID readers have more capabilities than tags with regard to internal storage and processing power, so complex cryptographic computations may be carried out by RFID readers. An RFID reader is connected to a back-end server through a secure communication channel. Currently, many applications rely on fixed reading devices, but may also be integrated into hand-held mobile devices [30, 31].

3. Back-end database (server): A server is a database that contains records associated with the RFID tags it manages. These records may contain product information, tracking logs, etc. The server has very high storage and computing capabilities to be able to manage hundreds or thousands of RFID tags.

   A typical passive RFID system is shown in Fig. 2.1, and works as follows:

   (a) The reader emits an RF signal via its antenna to power the tag.
   (b) The tag sends a message to the reader via the RF signal.
   (c) The reader forwards the tag's message to the server for further processing.
   (d) The server processes the tag's data and sends back a reply to the reader.
   (e) The reader forwards the server's reply to the tag. In some scenarios the tag and the server have to update their values to be used in the next session.

2.2.2 Operating Frequency

The electromagnetic spectrum within which RFID systems typically operate is commonly divided into low frequency (LF), high frequency (HF), ultra high frequency (UHF), and microwave. Different frequencies are widely deployed in RFID systems, ranging from 120 KHz to 5.8 GHz [111]. Table 2.1 lists standard frequencies and their respective read distances and standards.

- LF RFID tags: LF RFID tags typically operate in the 120-140 KHz range and have short read-ranges of 10-20 centimetres. LF signals are able to travel through

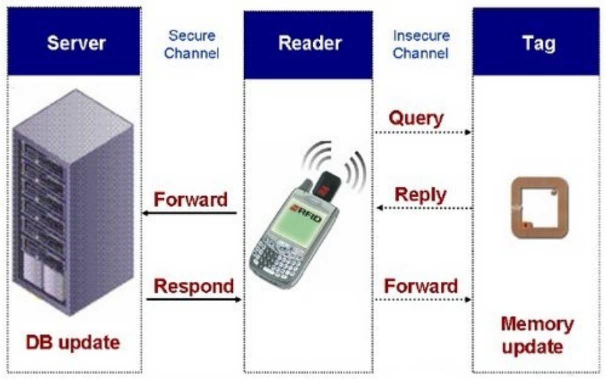


Figure 2.1: Typical passive RFID system scenario

water, metals or dirt, so they are often used in pet identification or laundry management tags. They are also used in short read-range applications such as access control and car immobilization.

- HF RFID tags: HF RFID tags operate in the 13.56 MHz frequency. HF tags offer a higher data distance up to 1 metre, but in practice the distance is 4-6cm as in the NFC technology, bankcards, Oyster cards etc. HF RFID tags do not perform as the LF RFID tags in proximity to metals and liquids. HF tags are mostly used in smart cards, library books, and transport ticketing systems.
- UHF RFID tags: UHF tags operate in the 868-928 MHz range. UHF tags are most commonly used for item tracking and supply-chain management applications due to the long read distance they offer and for their low-cost. These tags are more sensitive to metals and liquids. This thesis focusses on this type of tags.
- Microwave: Microwave tags operate at 2.45-5.8 GHz and has a read range of up to approximately 30 metres. Applications of microwave tags are highway toll collection and vehicle fleet identification. This type of tag consumes more energy and is more costly than the lower frequency tags.

2.2.3 RFID Communication Methods

Passive RFID tags obtain their power by harvesting energy from the electromagnetic field of the reader's signal. RFID tags communicate with the reader in two methods; inductive coupling or passive backscatter as follows [29]:

- Inductive coupling: LF and HF tags use inductive coupling. The reader generates a current magnetic field. Then, when the tag is placed close enough to the reader,

Table 2.1: RFID tag's frequency, reading distance and supported standards

Frequency Range	Frequency	Read Distance	Standard
Low Frequency (LF)	120-140 KHz	10-20 cm	ISO 18000-2 [27]
High Frequency (HF)	13.56 MHz	up to 1 meters	ISO 18000-3 [27] ISO 14443 [32] ISO 15693 [34]
Ultra High Frequency (UHF)	868-928 MHz	$<$ 10 meters	ISO 18000-6 [27] EPCGlobal [35] EPCGlobal [28]
Microwave	2.45 - 5.8 GHz	$>$ 10 meters	ISO 18000-4 [27]

the field from the reader coil will couple to the coil of the tag, charge the on-tag capacitor and generate a voltage that powers the tag circuitry. The reader modulates its magnetic field amplitude according to the data to be sent to the tag. A tag on the other hand, transmits its ID by turning on and off its load resistor in accordance with the digital data to be sent; this is known as *load modulation*. The reader detects these amplitude variations, and demodulates the transmitted message. The reading distance between the reader's antenna and the tag's antenna should be fairly small because of the limited range of the magnetic field as shown in Table 2.1. This method is shown in Fig. 2.2

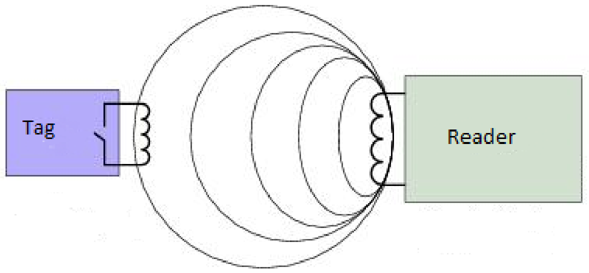


Figure 2.2: Inductive coupling

- Passive backscatter: This method is shown in Fig. 2.3. UHF and microwave tags use passive backscatter. A reader's antenna generates continuous electromagnetic waves, and this will develop a potential difference at the tag's antenna and energises the tag circuit. The tag's antenna is tuned to receive these waves and demodulates it into patterns of ones and zeros that form the commands for the operations to be performed. Then, the tag changes the amplitude of the electromagnetic waves reflected by the tag's antenna in accordance with the tag's messages, and this is called *backscattering*. Because the tag does not have power

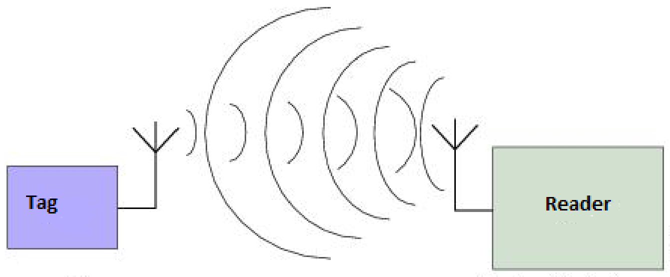


Figure 2.3: Passive backscatter

to transmit the data, it uses variable impedance with a transistor to send a wave back [33].

2.2.4 Anti-collision

Since a shared wireless channel is used between the reader and the tag, signal collision may occur. Collision in RFID is divided into tag collision and reader collision. Anti-collision protocols for the tag and reader are presented below:

- Tag anti-collision: When multiple tags are energised by the RFID reader simultaneously, tag collision may occur as the tags reflect their respective signals back to the reader at the same time. Such issue arises in some applications where multiple tags need to be read within the same RF field. As a result, the reader is unable to differentiate these signals. To minimize collisions, RFID tags must use an anti-collision protocol. The most widely used anti-collision protocol in RFID systems is the Aloha scheme [111, 29].

  In basic Aloha scheme, after the tag is energised, it sends its ID to the reader and waits for an acknowledgement (ACK) from the reader. If it receives a negative acknowledgement (NACK), meaning a collision has occurred, it resets and re-sends its ID again. Systems based on the basic Aloha scheme still suffer from the collision problem as all the tags send their ID randomly (no time allocation). Subsequently, several enhanced Aloha protocols have been proposed including Slotted Aloha and Framed Slotted Aloha.

  In a Slotted Aloha scheme, the tags transmit their messages at defined, synchronous time slots, which are controlled by the reader. A slot is a discrete time interval sufficiently long to allow the tag to transmit its ID. In this scheme, a tag must choose one of the slots randomly and transmit data within a single slot. If there is a collision and the tag does not receive an acknowledgement, it retransmits after a random delay. This means the collisions only occur at the start of each slot.

To further improve the Slotted Aloha, a Framed Slotted Aloha has been introduced. The time slots are grouped in a frame; in each read cycle there are multiple frames with the same number of slots. Each tag chooses a slot in a frame and sends its data once within a frame. If the tag does not receive an acknowledgement, it waits for the next read cycle and selects a slot in the new frame. The read cycle is repeated until no collision occurs. Framed Slotted Aloha can be classified into Basic Framed Slotted Aloha for fixed frame size and Dynamic Framed Slotted Aloha for variable frame size, where the number of tags increases during the protocol execution.

An Aloha scheme is a probabilistic algorithm, where the tags respond at randomly generated times, or within a slotted/framed time interval. Another scheme uses a deterministic algorithm in which the reader sorts through the tags based on their unique ID in a tree-based data structure. The reader searches the tree nodes of all possible ID numbers, and the presence of a response gives the reader an indication as to where to search next [36]. Tree-based structure may have longer identification delays but lower tag starvation problems, where a tag might wait for unlimited periods of time time to be identified [19].

- Reader anti-collision: Reader collision happens when two or more readers communicate on the same frequency at the same time (reader-to-reader frequency interference), or when a nearby reader attempts to query the same tag simultaneously (multiple reader-to-tag interference) [19]. Because the readers have more memory space and higher computing capabilities than tags, they can detect collision easily. A commonly used anti-collision protocol proposed in [37] allocates different frequency bands or times to neighbouring readers.

2.3 UHF RFID Standards

RFID standards describe the physical and the link layers, and the requirements for the air interface, anti-collision mechanisms, communication protocols and security functions. The two most relevant UHF RFID standards are the ISO 18000-6 standard [27] and the EPC Class-1 Generation-2 standard [35], hereinafter denoted as (EPCC1Gen2). ISO 18000-6 specifies the standard for the following tags:

- LF RFID tags

- HF RFID tags
- UHF RFID tags
- Microwave RFID tags

Two other ISO standards, ISO/IEC 14443 [32] and ISO/IEC 15693 [34] are related to smart card and proximity card interfaces operating in the HF range.

In November, 2013, a new version of the EPCC1Gen2 standard was released [28]. The new standard added new cryptographic functions that can be optionally performed. The additional security commands are *Authenticate, AuthComm, SecureComm, KeyUpdate, TagPrivilege.* The tag shall generate a 16-bit random number as in [35], and has the same probabilities of the random number generator (RNG) in the EPCC1Gen2 standard as discussed in the next section.

Some of the related work, and the implementation tools used in this thesis support the EPCC1Gen2 standard [35], hence, the next section will discuss some of the main specifications of the EPCC1Gen2 standard.

2.3.1 EPCglobal Standard EPC Class-1 Gen-2 Specifications

The EPCC1Gen2 standard aims to standardise and promote UHF RFID tags. In EPC Class-0, the tags are read-only devices, while in EPC Class-1, the tags' data are one-time programmable, but in practice, commercially available UHF tags can be erased and re-written numerous times. The standard specifies the following:

- Physical layer: The tags do not have any power source, they can respond only when they are powered by the reader. The communication between the reader and tag is half duplex, which means that the reader talks and the tag waits and listens for any incoming data, or vice versa, but not simultaneously.
- Tag memory Tag memory is logically divided into four banks as follows [35]:

    1. Reserved memory : Reserved memory stores a 32-bit kill password and a 32-bit access password. Once a tag receives the kill password, it is permanently disabled. Tags with a non-zero access password have to receive the correct access password before transitioning to the *secured state.*
    2. EPC memory: This area of memory contains a 96-bit unique identity of the tag, known as Electronic Product Code (EPC) value, a 16-bit cyclic redundancy check (CRC) value, and a protocol control (PC), where PC contains physical layer information.

3. TID memory: This memory bank contains an 8-bit ISO/IEC 15693 class identifier, and additional information that identifies the custom commands and/or optional features that a tag supports.
4. User memory: This memory bank is for user-specific data storage.

- Cryptographic operations

  The EPCC1Gen2 standard supports a 16-bit CRC to detect any modification of the backscattered messages [35]. The tag computes its 16-bit CRC over the values of PC and EPC, and maps the computed CRC-16 into the EPC memory.

  Moreover, tags generate 16-bit random number (RN-16) to be included in the transmitted messages and in handling password commands [35]. The designed RNG should meet the following probabilities: the probability that any two or more tags simultaneously generate the same sequence of RN-16 shall be less than 0.1%, and the generated random number shall not be predictable with a probability greater than 0.025%.

- Tag identification layer: For the reader to communicate with the tags, it performs three basic operations [35]:

  1. Select: This process is for selecting the tags the reader wants to communicate with; it is similar to selecting records from a database.
  2. Inventory. This process is performed in one session at a time to identify the tag. The tag responds with a 16-CRC value and its unique EPC value.
  3. Access: This operation is for interacting with the tag by reading and writing data in the tag's memory. Tags have to be identified before access.

- Slot counter: The EPCC1Gen2 is based on the Framed Slotted Aloha discussed in section 2.2.4. The number of time slots is set by the reader as Q, where Q is an integer ranging from 0 to 15, and is sent via the *Query* or *QueryAdjust* command. When the tag receives such a command, it extracts a value x between 0 and 2^Q-1 from its RNG, and loads x into its slot counter. If the number in the slot counter is zero, the tag is transferred into the *reply* state, otherwise to the *arbitrate* state.

- Tag states: RFID tags transit into different states during a session as follows [35]:

  – Ready state: The tag remains in the *ready* state after being energised. When it receives a *Query* command from the reader, it extracts a value x between 0 and 2^Q-1 from its RNG, and loads x into its slot counter. If the result is non-zero it transits to the *arbitrary* state, or to the *reply* state if the number is zero.

- Arbitrate state. A tag in an arbitrate state decrements its slot counter every time it receives a *QueryRep* command. When the slot counter reaches zero, it transits to the *reply* state and backscatters a 16-bit random number (RN16).
- Reply state: A tag sends an RN16, once it enters the reply state. If the tag receives a valid acknowledgement (ACK), it transits to the *acknowledge* state and sends the EPC value. Otherwise, the tag transits to the arbitrate state if it receives NACK.
- Acknowledge state. A tag in this state can transit to any state except the *killed* state. If the access password is non-zero the tag transits to the *open* state, otherwise it transits to the *secured* state.
- Open state: After receiving a Req_RN command, the tag backscatters a new RN16 that both reader and tag use in subsequence messages. If the tag receives the correct access password from the reader, it transits to the *secured* state.
- Secured state. Upon receiving a Req_RN command, the tag backscatters a new RN16 that both reader and tag use in future messages. Tags in a *secured* state may transit to any state except the *open* or the *acknowledge* states.
- Killed state: If the tag receives the correct *kill password* in either the *open* state or *secured* state, it will be disabled permanently.

2.4 RFID Applications

RFID systems are being adopted in a wide variety of fields including [38, 39, 24]:

1. Tracking and identification:
   - Pets with implanted tags
   - Product life cycles in supply-chain management
   - Checkout in retail shops
   - Asset tracking
   - Tagged tickets in public transport
   - Smart appliances in homes

    - Smart posters
    - E-passports

2. Access control:
    - Car ignition keys
    - Building/premises contactless proximity cards

3. Anti-Counterfeiting:
    - Casino tokens
    - Banknotes
    - Luxury goods

4. Automated payment
    - Contactless bank cards
    - Automotive toll payment
    - Electronic fare management systems

2.5 RFID Privacy and Security

Although RFID technology has several advantages over other identification technologies, security and privacy are among the main concerns that need to be tackled [12]. Because the tag sends its data to any nearby readers without alerting the tag's owner, the tag's data could be disclosed and/or tracked. Furthermore, the wireless channel between the reader and tag is vulnerable to two kinds of attacks; passive and active. A passive attack occurs when an adversary eavesdrops on the communication session between an authorised reader and tag. An active attack occurs when the adversary can impersonate, replay, modify, inject and/or block the messages between the reader and tag. In this thesis, passive and active attacks are referred as active attacks for simplicity.

In this section, we illustrate the main attributes that a secure RFID system requires, along with the associated attacks and threats.

2.5.1 RFID Privacy and Security Attributes

The main attributes that secure systems including RFID should take into account are [40]:

- Integrity/Authentication: The system should provide a mutual entity authentication, where the communication should take place between legitimate entities, and provide assurance to the reader or server about the identity of the tag and vice versa. This attribute forms a subset of the *integrity* attribute as both attributes' goals are to ensure that the data cannot be modified in transit by any malicious entity.
- Confidentiality/Privacy: Privacy fits into the *confidentiality* dimension. Privacy involves protecting the tag's data from being revealed to any malicious entity, and preventing the tag's location from being tracked.
- Availability: When requested, all entities should be present to provide other authorised parties with the information they need.

2.5.2 Attacks on RFID Systems

The nature of the wireless communication between the reader and tag enables an intruder to interfere and accomplish a variety of *active attacks* as shown in Fig. 2.4 and highlighted below [19, 17]:

1. The main attacks affecting *authentication* are:
    - **Impersonation attacks:** The attacker impersonates a legitimate entity by using the leaked sensitive data, or performing a replay/spoofing attack.
    - **Replay/Spoofing attacks:** The attacker eavesdrops on the communication between the reader and tag, then maintains the transmitted messages in order to impersonate a legitimate entity in the next session and replay these messages.
2. The main attacks affecting *privacy* are:
    - **Tag data leakage:** An RFID tag is associated to one unique identity, and this could lead to serious privacy concerns. Thus, the tag's sensitive data should not be revealed when transmitting in order to preserve the privacy of the tag's holder and to prevent tag impersonation attacks.
    - **Traceability:** Traceability is referred to as the ability of the adversary to decide with probability if messages from different sessions belong to the same tag or not. This can be done without knowing the tag's data; for example, the adversary might observe the level of signals, using the same tag's responses, or by linking the responses. To prevent attackers from tracking the

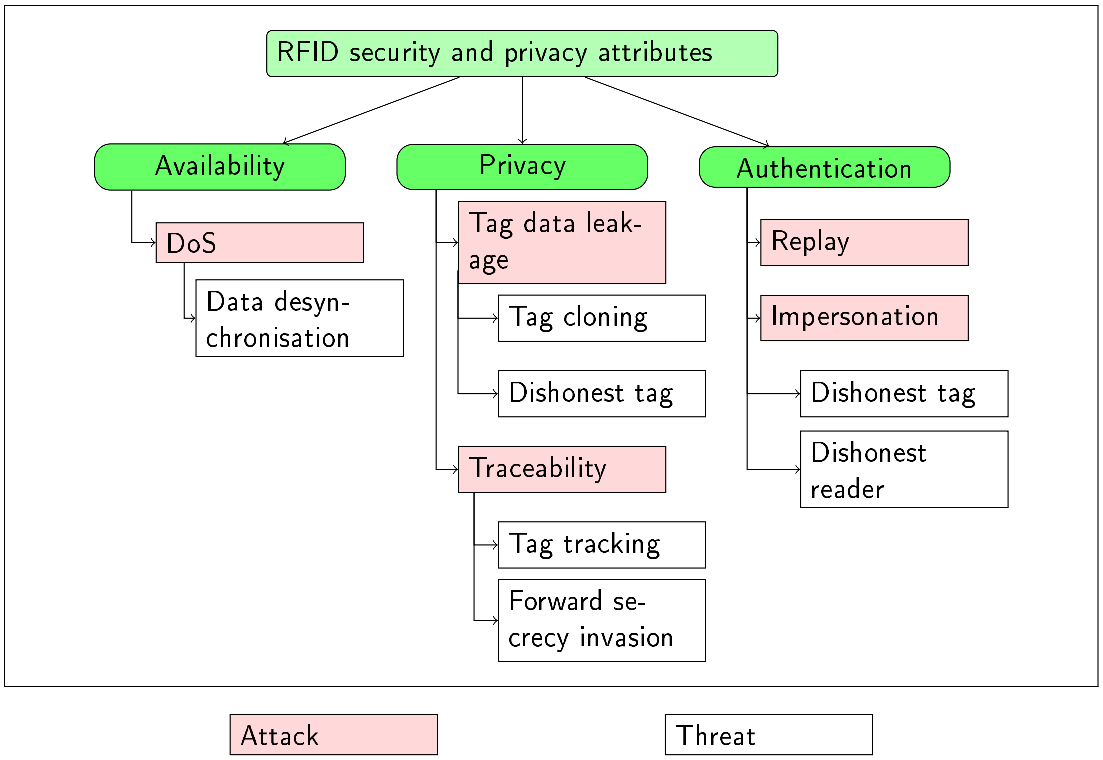


Figure 2.4: RFID attacks and threats

tag's location, the tag's replies should not be static or linkable to previous messages. Moreover, given that the attacker has eavesdropped on the session between the reader and tag and all the information stored in the involved tags has been revealed at time *t'*, the attacker should not be able to correlate any readings of the same tag at a time $t \leq t'$.

3. The main attack affecting *availability* is:

   - **Denial-of Service attacks (DoS):** The attacker might block the tag's or reader's messages from reaching the target, causing the tag or the reader to assume a state in which it can no longer be valid. Hence, the attacker's goal is to prevent the system from operating as intended, either temporarily or permanently. Another technique that the attacker can exploit, is to send a stream of messages to the RFID tags or the reader, flooding it with invalid messages.

2.5.3 RFID Main Threats

In this section, the principal threats that RFID systems might encounter are classified according to the authentication, privacy and availability attributes mentioned above, each of which is discussed below:

(1) RFID authentication threats: The main attacks related to hindering the authentication process are *impersonation*, and *replay attacks*. Such attacks share the following threats:

- **Dishonest tag:** An intruder can impersonate a legitimate tag to claim to be in the neighbourhood of the reader, cause the reader/server to authenticate the intruder instead of a legitimate tag, and/or to cause the server to update its value, causing a desynchronisation in the next session(s).
- **Dishonest reader/server:** Similarly, an intruder might impersonate a reader to claim to be in the neighbourhood of the tag, cause the tag to authenticate the intruder instead of a legitimate reader/server, or cause the tag to update its value causing a desynchronisation in the next session(s).

(2) RFID privacy threats: Privacy is a significant concern when using RFID technology. The wireless communications channel between a tag and a reader can disclose information about a tag, including its unique identity. In fact, the tag's unique ID (EPC) remains fixed most of the time, which would allow an attacker to establish an association between the tag's current response and its previous response(s). The two attacks associated to privacy are *tag data leakage* and *traceability*.

There are two possible threats related to *tag data leakage*:

- **Tag cloning:** An RFID system may be vulnerable to tag cloning, where the attacker uses a compromised tag's data to clone a tag and use it as if it was legitimate; for example, by removing the legitimate tag and replace it with a cloned tag, hence, an attacker can fool the system into believing the cloned tag is legitimate and present [110].
- **Dishonest tag:** An attacker may use a compromised tag's data to impersonate the tag, which could lead to unwanted results such as data desynchronisation, impersonating the reader/server in future sessions, or breaching the privacy of the tag's holder in terms of location and his/her past behaviour [19].

There are two possible threats related to *traceability*:

- **Forward secrecy invasion:** An RFID tag is generally not a tamper/attack-resistant device, so it can be physically compromised to reveal secret information stored in the memory of the tag. Forward secrecy guarantees that all the previous transactions that happened before the tag's secret was revealed remain unlinkable. In other words, in forward secrecy invasion, given all the stored data of a compromised tag at time t, the attacker is able to identify the tag at time $t_0 \leq t$. The past transactions of a tag may allow tracking of the tag owner's past locations and behaviour.
- **Tag location tracking:** If the data being sent from the tag to the reader is static or linked to data sent previously, the tag holder's location can be tracked without his/her knowledge.

(3) RFID availability threat: Availability is crucial since the core function of a secure RFID system is to mutually authenticate the tags and server/reader instantly; therefore, every entity should be available for authentication purposes. If a reader/server is no longer available, the tags will not be able to authenticate themselves, and hence abort the session and vice versa.

The main threat affecting *availability* is:

- **Data desynchronisation:** Data desynchronisation between the tag and reader may happen if both entities need to update their data after each successful identification, and can also occur in some protocols that provide countermeasures against replay attacks for example. If the data in the tag and in the server are desynchronised, the tag will not be able to authenticate the server and abort the session; hence, the availability feature will not be achieved. Data desynchronisation can either occur unintentionally when messages are lost during transmission due to system malfunction or communication errors, or can be implemented deliberately by an adversary on an RFID communication channel.

2.6 Common Countermeasures

In this section, we present some of the countermeasures for the attacks and threats mentioned in the previous section. Then, we discuss briefly the design of some of the cryptographic functions that can fit the RFID tags limited resources.

2.6.1 PUFs-Based Approach

One of the proposed solutions to combat tag cloning and replay attacks is the deployment of tamper-proof and unclonable functions on hardware; this refers to Physical Unclonable Functions (PUFs). PUFs can uniquely identify and authenticate each silicon chip by exploiting the physical characteristics of the silicon and the IC manufacturing process variations with a low cost mechanism [41]. PUFs are designed to be easy to evaluate but difficult to clone [42]. Therefore, PUFs has have become attractive for RFID ICs.

PUFs map a set of challenges to a set of unique responses based on the unique characteristics of a particular chip, or on a user-defined input; for example, the variation of delays in wires and gates in the chip. These delays in turn depend on highly unpredictable factors, such as manufacturing variations and noise [43].

In [42], the authors proposed using a PUF as a secure key derivation mechanism. Moreover, the authors proposed applying elliptic curve cryptography to the PUFs to authenticate a tag offline. Multiple challenges are given to the PUFs to generate several fingerprints with some auxiliary data. Then, the issuer signs the challenges, fingerprints and auxiliary data with his/her secret key, and embeds the signed data on the tag's chip. In the authentication step, the reader receives challenges and auxiliary data from the tag, and it challenges the tag's PUF with one of the challenges. When the reader receives the tag's fingerprint it checks its authenticity from the signed data. The authors in [42] claimed that the attacker needs to embed a fake physical structure on the product in order to produce correct fingerprints to the challenges. Also, the attacker cannot forge the challenges, fingerprints and auxiliary data as he/she does not know the issuer's secret key used in the signature. However, according to [43], this scheme is still expensive for low-cost RFID tags as it uses a public-key cryptography.

The authors in [43] proposed a privacy-preserving RFID protocol based on PUFs. Simply, when a reader interrogates a tag, the tag responds with its ID and updates its ID using PUFs (p(ID)). The database stores the sequence ID, p(ID),$p^{(2)}$(ID), ... , $p^{(k)}$(ID) for each tag, and sends a tuple of all possible tag IDs to the reader to find a match with the received tag's identity. This implies that the reader has to eliminate all the previous values within the tuple to prevent replay attacks. However, in this scheme, the tag can only be authenticated k times, which enables an attacker to perform a DoS attack [44]. Several other cryptographic primitives based on PUFs to achieve a mutual authentication between tag and server have been proposed; for example as in [9, 10, 46].

PUFs have also been used to solve server scalability issues in RFIDs [47, 48, 10, 49]. In these schemes, the reader stores the tag's ID, which is updated in each session using PUFs; thus the attacker cannot guess or clone the tag's ID in future sessions, and the reader/server can retrieve the tag's record in O(1).

In real-world applications, Devadas et al. [41] fabricated a PUF-enabled unclonable RFID tag and showed that a PUF circuit can be integrated into a passive RFID tag for authentication purposes. Furthermore, Holcomb et al. [11], proposed SRAM-PUFs, which use SRAM cells in RFID chips as a PUF mechanism.

2.6.2 Human-Computer (HB) Protocols

In 2005, Juels et al. [7] adapted the concept of human-to-computer authentication protocols proposed by Hopper and Blum (HB protocol) [8] for use on low-cost RFID tags. This type of protocol is based on the computational hardness of the *Learning Parity with Noise Problem (LPNP)*, and uses dot products of binary vectors and a random noise bit. The HB-style protocols do not depend on common cryptographic methods but on the correctness of the tag's replies in several rounds.

In the HB protocol [8], in q rounds, the reader sends a challenge a to the tag, and the tag sends the binary product of $a.x$ xored with *noise* (v), where x is a secret key shared between the tag and reader. This protocol can resist passive attacks but not active attacks. As a result, Juels et al. [7] proposed an enhanced version of the HB protocol (called HB$^+$) by including a new secret value y, which serves as another secret between the tag and reader. The tag calculates $z{=}r_A.x \oplus r_B.y \oplus v$, and sends z to the reader, where r_A and r_B are binary vectors generated by the tag and reader respectively. However, Gilbert et al.[6] showed that this protocol is vulnerable to man-in-the-middle attacks.

In response to Gilbert et al.'s [6] attack on HB$^+$, Bringer et al. [50] proposed an HB^{++} protocol to avoid man-in-the-middle-attacks. The protocol involves renewing the secrets in each session and correlating the tag's and reader's challenges. However, according to Piramuthu [51], HB^{++} is still not immune to attacks from an adversary that pretends to be an authentic reader. Piramuthu [51] suggested updating the value of ρ every time z is computed, instead of updating it at the beginning of each round to prevent an attacker from revealing the secret data. Also, to make the protocol more lightweight, Piramuthu suggested omitting z, x, y, v for the protocol execution.

Munilla et al. [52] proposed another HP protocol called the HP-MP protocol. In this protocol, the reader sends a challenge a to the tag, and the tag then computes $z{=}\ a.x \oplus v$. The tag then looks for a binary vector b, where $b.x{=}z$, and sends b to the reader. The reader checks whether $b.x = a.x$ and authenticates the tag. This protocol has been shown to be insecure against passive attacks [5], and is also vulnerable to replay attacks [13]. Since then a range of HB proposals closely related to HB-MP have been proposed to overcome the attacks discussed in [6, 51].

A new area, where distance bounding protocols and HB family protocols can be integrated to thwart relay and replay attacks, has been proposed in [53]. Pagnin et al. [53], suggested combating man-in-the-middle attacks on HB protocols by modifying the communication channel of the receiver architecture using a distance bounding protocol. This scheme is known as a hybrid HB$^+$DB protocol. In HB$^+$DB, the reader and tag share three secret values, namely (x, y, z). The protocol is divided into three phases. The first phase is the initialisation of the HB protocol; the tag exchanges a challenge s with the reader, and computes c_i=*(b.y)* $\oplus$ v, where b=*PRNG*$_z$*(s)* is computed by both entities. In the second phase, the distance bounding protocol is initiated by the reader, which starts a timer and sends a challenge a to the tag. On the tag, the message $r_i = a.x \oplus c_i$ is computed and sent to the reader. When the reader receives the tag's response it stops the clock and verifies that the tag's response is correct, and that the tag is close enough to the reader.

2.6.3 Lightweight Cryptographic Functions

RFID tags are deployed in many applications that require security and privacy. In order to satisfy these needs, cryptographic algorithms such as block ciphers can be an ideal solution. However, the problem in providing some of the commonly used cryptographic techniques on these devices is the extremely constrained environment. The cryptographic primitives have to be of low memory, have minimal power consumption, and operate at sufficient speed. Hence, researchers have attempted to design lightweight functions that suit RFID tags resource constraints.

Starting with a block cipher scheme to calculate message digests, Yoshida et al. [54], proposed a compression function called the MAME block cipher, specifically for limited-resources hardware. The function takes a 256-bit message block, produces a 256-bit output and requires 8100 gate equivalent (GE). However, according to [55], the work in [54] is still demanding for RFID tags.

Another well-known block cipher scheme called PRESENT has been proposed [56]. The original form of PRESENT is a block cipher that accepts 64-bit input and 80-bit or 128-bit keys, with 1570 GE. Several PRESENT block cipher improvements have been proposed to minimise the required GE for RFID tags, including work by [57, 58]. KATAN [59] is another example of a block cipher that can be fitted into constrained devices and consumes less GE than PRESENT. KATAN is composed of three block ciphers of 32, 48, or 64-bit block size, requiring 802 GE, 927 GE and 1054 GE respectively. The key size for all variations is 80-bit and the number of rounds is 254. A smaller GE block cipher has been proposed in [60]. The authors in [60] designed Piccolo, an ultra-lightweight block cipher, which is suitable for extremely constrained environments such as RFID tags. Piccolo is composed of a 64-bit block cipher, 80 or 128-bit keys, and requires 683 or 758 GE respectively. The authors claimed that Piccolo is the smallest of the current lightweight block ciphers discussed in the literature.

Bertoni et al. designed a sponge-construction hash function. Since then several hash functions have been proposed. O'Neill [61] designed an 8-bit low-cost-SHA-1 function requiring 5527 GE, which saves 1200 GE as compared to other SHA-1 hash functions. According to [55], this hash function [61, 54] is still demanding for RFID tags. The authors in [55] proposed a QUARK lightweight hash function for 64 and 112 bit security, needing 1379 GE and 2296 GE respectively. Their work was inspired by the ultra-lightweight block cipher PRESENT [56]. Two other lightweight hash functions based on QUARK have been proposed: SPONGENT [62] and PHOTON [63]. Regarding SHA-3, Keccak is a family of hash functions that are based on the sponge construction, which can perform nearly all symmetric cryptographic functions and can fit constrained devices [4]. It uses one of seven permutations named Keccak-f [b], where b $\in$ {25, 50, 100, 200, 400, 800 or 1600} is the width of the permutation. The sponge construction state's width is also determined by the width of the specified permutation [4]. The largest permutation is Keccak-f [1600]. In 2013, Kavun et al. presented a lightweight implementation of the SHA-3 hash function [157] with 20790 GE and a 64-bit output for Keccak-f [1600]. However, Pessl and Hutter [64], showed that Kavun et al.'s implementation of Keccak-f [1600] does not fulfil RFID tag demands. They designed a hardware implementation of SHA-3 Keccak-f [1600] that is smaller than Keccak-f [1600], SHA-1 and SHA-2 GE, provides 128-bit security and requires 5500 GE.

2.7 Summary

In this chapter, we provided a background information about RFID technology, starting from its origins, then illustrating the main RFID architecture with regard to the RFID components, the operating frequencies, the communication methods, and the anti-collision protocols. This was followed by a description of the standards supported by other related work. Then, we outlined the current deployments of RFID technology. Finally, we discussed the main security and privacy requirements of RFID systems and associated attacks and threats, and concluded with some common countermeasures to such attacks. The next chapter presents a description of the tools used in formal analysis and performance measurement.

Chapter 3

Formal Analysis and Implementation Tools

Contents

In this chapter, we present the tools used in the formal analysis of the proposed protocols and in the implementation process. We provide a description of CasperFDR and Scyhter formal analysis tools. Then, we highlight the tools used in the implementation of the proposed protocols, and the performance measurement techniques.

3.1 Introduction

A considerable number of RFID security protocols have appeared in the academic literature, claiming to be secure in the presence of a malicious agent, called an intruder, who is assumed to have complete control over the communications network. Unfortunately, a large proportion of the RFID protocols fail to meet security and privacy requirements as they have been shown to be vulnerable to attacks [65, 66, 68, 69, 70, 112, 71, 72, 73, 74].

Therefore, for the sake of completeness, we subjected our proposed protocols to formal analysis tools such as CasperFDR [104] and Scyther[75].

In addition, to measure the performance of the proposed protocols in a restricted embedded device, such as the RFID tag, we implemented the protocols in Chapter 5, 6, 7, and 8. After the execution of the protocol, we measured the used memory space, the protocol's computing time, the power consumption, and the time required to send and receive messages between the tag and the reader (communication time cost).

3.2 Formal Analysis Tools

Cryptographic protocols are widely adopted as significant components in meeting security requirements. To implement a system, it must be demonstrated that the protocol satisfies the fundamental security requirements. Formal verification tools provide a good way to validate this challenge, and considerable developments have been made in their development over the last 30 years [103]. CasperFDR and Scyther analyse protocols under the assumption of perfect cryptography. Perfect cryptography means using cryptographic functions in the encryption process, where the adversary learns nothing from the encrypted messages unless he/she knows the decryption key.

These tools check that the communication channel between the sender and receiver achieves *secrecy* and *authentication* based on the protocol's specifications [76], which meet the objectives of our protocols. Secrecy means that the exchanged secret data cannot be accessed by an intruder. Authentication means that every party can authenticate the party with whom they are executing the protocol. We chose these tools to add more value to our protocols. Also, they have proved their capabilities in finding vulnerabilities in many protocols, such as in [77, 78, 15, 3, 79, 80, 81]. In addition, Scyther, for example, assists in protocol analysis by providing classes of attacks, in an unbounded number of sessions, as opposed to the single attack traces provided by other formal analysis tools.

In this section, we provide a description of the tools used to formally analyse the proposed protocols in Chapter 5, 6, 7, and 8.

3.2.1 CasperFDR

Security protocols were analysed using process algebras communicating sequential processes (CSP) [20] and its model checker failures-divergence refinement (FDR) [21]. Briefly, they work as follows:

- Each agent taking part in the protocol is modelled as a CSP process, including the intruder.
- The compiled protocol specification is tested against the specified security properties, such as "correctly achieves authentication", or "ensures secrecy"; FDR searches the state space to investigate whether any insecure traces can occur.
- If FDR finds that the security properties are not achieved, then it returns a trace of the system that does not satisfy the security properties; this trace corresponds to an attack upon the protocol.

Although CSP and FDR have proved successful in finding attacks in a number of protocols [2, 14], the task of producing a CSP description of a system is very time-consuming and error-prone. As a result, Lowe proposed CasperFDR a tool for simplifying this process [104]. The CasperFDR tool takes an abstract description of the protocol, together with its security requirements, and produces a CSP code checked and verified by FDR. The protocol is analysed in the context of the Dolev-Yao model [67], where the intruder has full control over the communication such that the intruder may intercept, analyse, modify messages, and/or send any message he/she composes to other agents, pretending to come from a legitimate agent. CasperFDR specifies the cryptographic primitives as a *black-box approach*, which means CasperFDR does not know which mathematical objects are used, only their properties. It supports symmetric and asymmetric encryption including hash functions.

CasperFDR checks the authenticity of the transmitted data by examining the associated events, namely *Running* and *Commit*. The *Running* and *Commit* events are attached to security and authentication specifications [82]. When the sender sends a message to the receiver, the receiver performs the *Running* event, which means that it starts running the protocol apparently with the sender. The sender performs the *Commit* event when it receives the receiver's reply, which means that the sender has finished a run of the protocol with the receiver. Regarding checking the security requirements, CasperFDR checks the secrecy specifications via an event called *Claim_ Secret*, which is performed by both parties. When the sender receives the receiver's message, it performs *Claim_ Secret* to ensure that the data are kept secret.

The input file in CasperFDR includes the following sections, which are divided into protocol definition and system definition:

1. Protocol definition: This part defines the generic operation of the protocol. It consists of:

    - Protocol description: This part defines the sequence of sending the data, and the contents of the exchanged messages. For encryption, for example, the expression is $A \rightarrow B$: *{m}{k}*, where agent A sends a message m encrypted with a secret key k to agent B.
    - Free variables: The types of variables and functions that are used in the protocol are defined in this section. Some of the predefined types used in our analysis are (*Agent, Server, Data, Nonce, TimeStamp*, and *HashFunction*). We also use *InversKeys*, which returns keys that are inverses of each other.
    - Processes: Each agent running in the system is represented by a CSP process parametrised by some arguments, and the parameters following the keyword "knows" define the knowledge that the agent is expected to have at the beginning of the protocol run. For example: *INITIATOR(A,na) knows SK(A), PK(A), PK*, means that agent A has one nonce na as an argument and he/she knows its secret key *SK(A), public key PK(A)* and the other communication partner's public key *PK*.
    - Specifications: This section defines the protocol's goals, such as *Secret* and *Agreement*. *Secret* implies that the data should be kept secret between two partners, for example, *Secret(A, na, [B])* can be paraphrased as: agent A thinks that *(na)* is a secret that can be known to only itself and agent B. *Agreement* specifies the authentication requirement, for example, *Agreement(A,B,[na,nb])* specifies that agent A is correctly authenticated to agent B , and the two agents agree on the data values na and nb.

2. System definition: The system definition includes the components required to be checked by the analyser FDR. It consists of:

    - Type definition (actual variables): During the analysis of the protocol using FDR, it uses the variables defined in this section. It defines the variables of the actual system and they are the same as the variables defined in the Free variables section. An intruder is also defined here as an agent that participates during the protocol execution.

- Functions: Defines the functions used by the agent(s). They are of type *symbolic*, which means that CasperFDR produces its own values as the result of this function.
- System definition: This section illustrates which agent should be present and checked by FDR, and it should match the specifications in the Processes section.
- Intruder: The identity of the intruder and the values he/she knows during the protocol execution are defined here. For example:
  Intruder = Mallory
  IntruderKnowledge = {Alice, Bob, Mallory, nm, PK, SK(Mallory)}, means the intruder *Mallory* knows the agents *Alice* and *Bob*, his/her nonce *nm*, his public key and Alice and Bob's public keys *PK*, and his secret key *SK(Mallory)*.

3.2.2 Scyther

Scyther [75] is a formal analysis tool that analyses protocols under the assumption of perfect cryptography. Scyther checks the secrecy and authenticity of the transmitted data. Unlike other formal tools, Scyther can verify protocols for an unbounded number of sessions; it establishes that the security properties hold for all possible behaviours of a protocol in the presence of a Dolev-Yao style intruder. In case that Scyther cannot not establish unbounded verification, it establishes a form of bounded verification [83]. Scyther accepts the protocol description as an input, outputs a summary report, and displays a graph if there is an attack on the protocol. The domain analysis is as follows [83]:

- Protocol specification: Scyther specifies the security protocol as an *abstract* syntax, where the protocol run is specified as a sequence list of *send* and *receive* events. Initial knowledge such as variables, constants and functions are declared at the beginning of the protocol specification.

- Agent model: Any entity that participates in the protocol execution is specified as a *role* in a *closed world assumption*, which means that only *honest* role (agent) who shows no behaviour other than the behaviour described in the protocol specification participate in the protocol execution. A *role* can execute multiple runs of the protocol.

- Threat model: Scyther uses a Dolev-Yao adversary model [67].

- Cryptographic primitives: Scyther deploys a *perfect cryptography assumption.* Also, similar to CasperFDR, Scyther specifies the cryptographic primitives as a *black-box approach.*

- Intruder initial knowledge: In Scyther the intruder initial knowledge is the names of roles, their public keys, global variables, nonces and the intruder's secret keys. For a Dolev-Yao model (network model), the intruder rules are defined as {*deflect; inject*}, where in wireless communication the intruder's rules are defined as {*eavesdrop;jam;inject*}.

- Role terms: There are four basic terms in Scyther: *Var*, denoting variables that are used to store received messages; *Fresh*, denoting values that are freshly generated for each role; and *Role*, denoting roles (agents).

- Events: In Scyther, there are two events *send* and *recv*. For example, *send (R, T , rt)* denotes the sending of a message *rt* by the role *R*, intended for the role *T*. Likewise, *recv (R, T , rt)* denotes the receipt of message *rt* by *T*, apparently sent by *R*.

- Predefined types: The predefined types are *agent, Function, Nonce, TimeStamp* and *Ticket*. Ticket is used to substitute any unknown terms. It is also possible to define a new user type by using the *usertype* command.

- Security requirements: Scyther defines the objectives of the security protocol as *secrecy* and two forms of *authentication*; data authentication and entity authentication. Secrecy means the secret data should remain secret even if the communication channel is compromised. *Secrecy* is specified as *Claim(R,secret,rt)*, which means the data *rt* should remain secret for the role *R*. *Authentication* is achieved in Scyther using three forms, including *Aliveness, Synchronisation (Nisynch)* and *Agreement (Niagree)*, which are defined below:

    1. Aliveness: If B runs a protocol with A and it is successfully completed by role B, then role A has previously been running the protocol. Aliveness is specified as *claim(A, Alive)*, where *A* is the role executing this event.
    2. Nisynch (non-injective synchronisation): Means the two events (send and recv) must be executed in the expected order as specified in the specification. Synchronisation combines aliveness and entity authentication. In other words, synchronisation confirms that all the received messages were indeed sent by the partner agent, the sent messages have indeed been received by the receiver, and the actual message occurred exactly as specified by the protocol description. Synchronisation is specified as *claim(A, Nisynch).*
    3. Niagree (non-injective agreement): While synchronisation focuses on the behaviour of the exchanged messages, agreement focuses on the contents of the

exchanged messages. It means that A and B agree that both roles are alive and agree to the values of the variables after the execution of the protocol. Agreement is specified as *claim(A, Niagree)*.

3.3 Implementation Tools and Performance Measurement Techniques

In this section, we outline the tools used in the implementation of the protocols discussed in Chapters 5 , 6, 7, and 8. Then, we discuss the procedures we followed in programming the tag's firmware and the main issues associated with the implementation. Finally, we list the performance measurement techniques.

3.3.1 Lab Set-up

The implementation of our proposed protocols involved an EPCC1Gen2-compliant RFID tag [84], an EPCC1Gen2-compliant RFID reader [85], an AVR ICE JTAG programmer as shown in Fig. 3.1, and one laptop (host computer). The used tools support the EPCC1Gen2 standard discussed in Section 2.3.1.

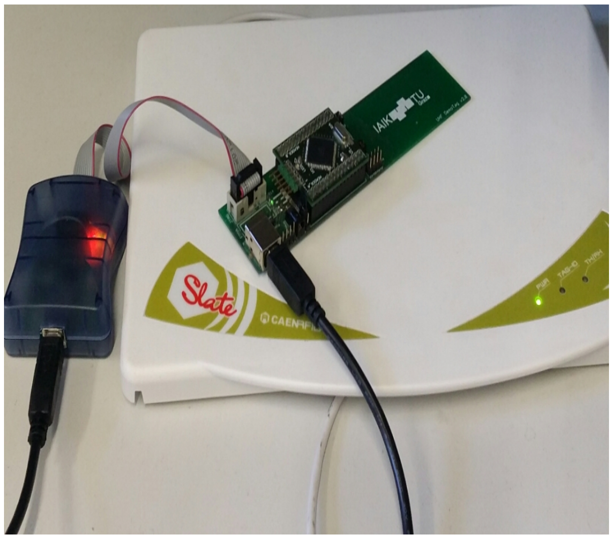


Figure 3.1: The CAEN Slate reader, the AVR JTAG ICE programmer, and the RFID UHF DemoTag

For the tag side, we used a programmable, semi-passive battery-powered RFID tag called *DemoTag* developed by IAIK TU Graz [84] shown in Fig. 3.2. The *DemoTag* is used to emulate a real RFID EPCC1Gen2 tag and is designed to add some custom commands to the EPCC1Gen2 standard. The tag's PCB board consists of a UHF antenna,

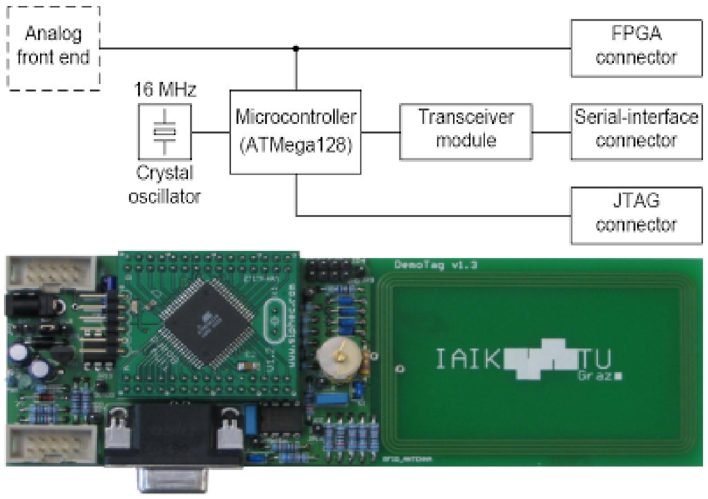


Figure 3.2: DemoTag structure [84]

a power supply, a USB port to communicate with the host computer, a JTAG connector, an analogue front end and a programmable Atmel ATMega128 microcontroller (F_CPU = 16 MHz).

The original *DemoTag* firmware provides an implementation of the EPCC1Gen2 protocol, which is implemented as a firmware library. The firmware running on the *DemoTag* is written in C code using Crossworks for AVR IDE from Rowley Associates [86]. The *DemoTag* has 128 KB of flash memory, 4 KB of RAM, and 4 KB of non-volatile EEPROM memory. The firmware is stored in the flash memory, while data such as the tag's ID and messages are stored in the EEPROM. The tag stores one word (16-bit) per memory bank.

We modified the original tag's firmware by adding more functions to conform with our protocol, including functions for calculating the hash on the tag's data. The updated firmware was debugged then uploaded to the flash memory of the microcontroller in the form of .hex via the *AVR ICE JTAG programmer*.

For the reader side, we used the *Slate (model R1260I) desktop reader* developed by CAEN [85]; it is the reader supported by the *DemoTag*. *Slate desktop reader* is an UHF RFID EPCC1Gen2-compliant reader with integrated antenna. It is embedded with an EPCC1Gen2 reader firmware, which is controlled by the host computer via a USB link. For programming the reader to read and write data into the tag's memory, and the generation of the reader's random number and messages, we used Microsoft Visual Studio C#.

3.3.2 Implementation Process

The tag is programmed using C language. The tag computes the required messages, and stores them in the user memory bank discussed in Section 2.3.1 within the EEPROM memory. The tag stores the data using the tag's firmware command *syscall_ writeWord* to be read on demand. Finally, it updates the data if required.

To generate a random number in the tag, we used the existing PRNG function that is included in the original firmware. For the hash function, we used an SHA-2 (SHA-256) included in *Crypto-avr-lib SHA 256* library [87]. This library provides special implementations of cryptographic functions in C, which respect the microcontroller's limited resources.

We used C# to program the reader application. The reader's library *CAENRFIDLib* is imported for communicating with the tag, reading the tag's memory and writing data into the tag's memory. The two commands used in our protocols were the *WriteTagData_ EPC_ C1G2* method to write data in the specified memory bank, and the *ReadTagData_ EPC_ C1G2* method to read data from the tag's specified memory bank.

The *CAENRFIDLib* library does not provide methods for generating random numbers and calculating hash functions. Therefore, to generate random numbers, we used a .NET Framework method called *rngCsp* for this purpose. To calculate the hash on the tag's data we imported the *Crypto-avr-lib SHA-256* library. The *Crypto-avr-lib SHA-256* is the same hash function used in the tag.

We faced a communication overhead between the reader and the tag during the implementation process. The reader could not write messages in the tag's memory in a single write command, as the EPCC1Gen2-compliant tag is programmed to write only one word (16-bit) of data in a single write command. As a result, in some of the protocols' implementations we had to send more than 40 write commands to transmit the reader's messages, and this sometimes led to communication loss between the reader and the tag, and also affected the performance dramatically.

3.3.3 Performance Measurement Techniques

The aim of this section, is to discuss the techniques we deployed in measuring the performance of the proposed protocols on a resource-restricted device such as an RFID tag. The main measurements are:

1. The tag's memory space: The cost of storing the tag's data, the reader's messages, the tag's messages and the tag's firmware.

2. The communication cost between the reader and the tag: Communication cost refers to the time cost for a tag to read/write data in its memory and the time cost for a reader to read/write data in the tag's memory.

3. The tag's computing time cost: The cost of computing each message independently, and the total cost of running the whole protocol on the tag.

4. The tag's power consumption.

To measure memory cost, after debugging the firmware, *Crossworks for AVR IDE* provided us with the total cost of storing the tag's data, random numbers and tag/reader exchanged messages in the EEPROM and flash memory.

Moreover, since the SHA-2 code runs on Atmel ATMega128 microcontroller, there are no real GE, the amount of RAM and flash memory, which are used by software implementations take the place of GE. *Crossworks for AVR IDE* shows that the size of the deployed SHA-2 functions within the flash memory is 1.4 KB.

To measure the communication costs in the tag, the Atmega128 microcontroller inside the DemoTag is embedded with a *Timer1/counter*. The *Timer1* is a 16-bit register that is capable of counting from 0 to 65535 transitions. We used *TCCR1B* as a timer control register, and a counter *TCNT1*, which counts the internal System Clock ticks. In the reader side, to measure the communication cost, the .NET Framework *stopwatch* function was deployed at the beginning and the end of reading and writing functions, then we calculated the elapsed time using a timing function *stopwatch.ElapsedMilliseconds*.

To measure the execution time for computing each message (computing time cost) and/or to measure the time to read/write data (communication cost) in the tag's memory, we started the timer *TCCR1B* and the counter *TCNT1* as (*TCCR1B* |= *((1 « CS10) | (0 « CS11))*) with no Prescaler (Timer Clock = System Clock), and *TCNT1=0* respectively. Then, at the end of the message's function execution we stopped the timer by using the following command (*TCCR1B* |= *((0 « CS10) | (0 « CS11))*) and printed the counter value. Each value in the counter represents a transition (one clock pulse), so to calculate the time period in milliseconds (ms) for an F_CPU = 16 MHz, we used the following equation:

$$Time = 1/Frequency = 1/16MHz = 0,0000000625sec \tag{3.1}$$

Thus each transition only took 0.0000000625 sec. For example, if the counter value is 65535, the time is 0,0000000625 * 65535 = 0,0040959375 sec ≡ 4 ms.

Finally, for measuring the power consumption of the *DemoTag*, we used a digital multimeter [88] to measure the Direct Current (DC) Voltage, DC Current, and Resistance. The *DemoTag* power supply is provided by a USB port, which provides an 5V DC voltage. The resistance provided in the *DemoTag* is 1470 Ohms. The average power consumption in all the proposed protocols is about 17 mW @ 1 MHz clock frequency.

3.4 Summary

In this chapter, we provided a background information about the tools used in the formal analysis of our protocols. CasperFDR and Scyther are deployed to confirm that secret data remains private and the participating entities are mutually authenticated. The tools deployed in the performance measurement and the techniques for measuring the tag's memory, communication, computing costs and power consumption were also demonstrated in this chapter. In the next chapter, we will study one of the widely-cited RFID protocols, demonstrate its vulnerability to DoS attacks, and provide some recommendations for combating such attacks.

Chapter 4

Data Desynchronisation on the Song RFID Mutual Authentication Protocol

Contents

This chapter reviews how protocol's design flaws can be exploited to perform a data desynchronisation on one of the widely cited RFID mutual authentication protocols proposed by Boyeon Song. We firstly give a general description of the data desynchronisation that might occur in RFID mutual authentication protocols. Subsequently, we explain in detail the Song protocol, and demonstrate how data in the Song protocol can be desynchronised. Finally, we propose a countermeasure that overcomes the Song's protocol vulnerability.

4.1 Introduction

In this chapter, we highlight weaknesses in an existing lightweight RFID mutual authentication protocol proposed by Song et al. in [89] (referred to hereafter as Song protocol). In [39], Song proposed an enhanced version of her protocol presented in [89]. In both versions, we discovered that the tag's and server's data can be desynchronised although they provide a synchronisation process.

Insecure communication between reader and tag is inherently vulnerable to interception, modification, fabrication and replay attacks as described in Section 2.5.2. In addition to these attacks is data desynchronisation, which is one of the major problems encountered in designing a secure RFID system. If data has to be updated synchronously after authentication is achieved between tag and server, an attacker can cause a data desynchronisation by blocking the exchanged message(s) from reaching the target, meaning that the receiver will not update the data, and hence, authentication will not be achieved in subsequent transaction(s).

Song proposed an RFID mutual authentication protocol that aims to achieve privacy and security as follows:

- The protocol achieves *privacy* by using a challenge-response scheme. The tag generates a cryptographic nonce to send a different response for every reader query, so the tag's transmitted message(s) cannot be traced or linked. Moreover, the tag and server update their values after each successful session.
- The tag calculates a keyed-hash function (Message Authentication Code (MAC)) on the tag's secret data to meet the *integrity* attribute.
- To combat data desynchronisation, the server database stores both the most recent previous data (old) and the current data (new) for each tag, hence achieving the *availability* feature.
- The tag's hashed identifier is a result of applying the hash function on its unique ID, assigned and calculated by the server, hence achieving the *confidentiality* feature.

Song claimed that the proposed protocol resists data desynchronisation by storing the old and new values of the tag's data in the server, thus when an attacker blocks the transmitted message(s), the server still can use the most recent old values to resynchronise with the tag in the next transaction. However, in this chapter, we will show how an attacker may apply data desynchronisation on the Song protocol, without compromising the internal data stored in the tag. In section 4.2, we will explain how data transfer between tag and server can be desynchronised in general.

4.2 Data Desynchronisation Description

Data desynchronisation between a tag and a server may occur unintentionally when messages are lost during transmission due to system malfunction or communication error. In addition, according to [17], data desynchronisation can be implemented deliberately on RFID communication channels via, for example, using one of the following options:

- Blocking the messages from reaching the targets. The attacker may use a "blocker tags" [16] or an "RFID Guardian" [90] to perform a deliberate DoS. These two schemes were potentially designed to achieve privacy for RFID tags.
- Active jamming, by transmitting a continuous signal to the tag to prevent the tag from communicating with the reader.
- An adversary may take advantage of the tag's limited resources and send a stream of random messages to it, so it will be flooded with random messages, and will abort the session without updating the data.

Storing and updating the (new) value to represent the current value, and the (old) value to represent the previous value can partially prevent data desynchronisation. In other words, even if the server updates tag's data after a successful authentication, and stores the recent old value, an adversary can easily cause synchronisation failure by intercepting and blocking messages between the server and the tag in *two or more consecutive sessions*, resulting in mismatched values. In Section 4.4, we demonstrate how this attack can be applied on the Song protocol.

4.3 Review of the Song Protocol

This section presents the Song protocol in detail. Notation used in this paper is defined in Table 4.1. The Song protocol consists of two processes: the initialisation process, and the authentication process, which are summarised below:

- Initialisation Process:

  This stage only occurs during manufacturing when the manufacturer assigns the initial values in the server and tag. The initialisation process is summarised below:

Table 4.1: A summary of notation

Notation	Description
h	A hash function, h : $\{0, 1\}^* \rightarrow \{0, 1\}^L$, where L is a bit-length of a tag identifier
f_k	A keyed hash function, $f_k : \{0, 1\}^* \times \{0, 1\}^L \rightarrow \{0, 1\}^L$
N	The number of tags
T_i	The i^{th} tag ($1 \leq i \leq N$)
D_i	The detailed information associated with tag T_i
s_i	A string of L bits assigned to i^{th} tag T_i
t_i	T_i's identifier of L bits, which equals $h(s_i)$
x_{new}	The new (refreshed) value of x
x_{old}	The most recent value of x
r	A random string of L bits
ε	Error message
$\oplus$	An XoR operator
$\parallel$	A concatenation operator
$\leftarrow$	A substitution operator
$x \gg k$	A Right circular shift operator, which rotates all bits of x to the right by k bits, as if the right and left ends of x were joined
$x \ll k$	A Left circular shift operator, which rotates all bits of x to the left by k bits, as if the left and right ends of x were joined
$\in_R$	The random choice operator, which randomly selects an element from a finite set using a uniform probability distribution

- An initiator (e.g. the tag manufacturer) assigns a string s_i of L bits to each tag T_i, computes $t_i = h(s_i)$, and stores t_i in the tag, where L should be large enough so that an exhaustive search to find the L-bit values t_i and s_i is computationally infeasible.
- The initiator stores the entries $[(s_i, t_i)_{new}, (s_i, t_i)_{old}]$ for every tag that it manages in the server. Initially $(s_i, t_i)_{new}$ is assigned with the initial values of s_i and t_i, and $(s_i, t_i)_{old}$ is set to null.

• Authentication Process:

The authentication process is shown in Fig. 4.1 as presented in [39], and summarised below:

1. Reader: A reader generates a random bit-string $r1 \in_R \{0, 1\}^L$, and sends it to the tag T_i.
2. Tag: The tag T_i generates a random bit-string $r2 \in_R \{0, 1\}^L$ as a temporary secret for the session, and computes $M1 = t_i \oplus r2$ and $M2 = f_{ti}(r1 \parallel r2)$, then sends M1 and M2 to the reader.

Server		Reader		T_i
			$\xrightarrow{1-r1}$	

				2-Generates r2
				Computes:
				M1 = $t_i \oplus$ r2
				M2 = f_{ti} (r1 $\|$ r2)

			$\xleftarrow{2-r1,M1,M2}$	
	$\xleftarrow{3-r1,M1,M2}$			
4-Re-computes M1 and M2 Computes: M3 = $s_i \oplus f_{ti}$ (r2 $\|$ r1)				
	$\xrightarrow{4-M3}$		$\xrightarrow{5-M3}$	
Updates: $s_i(_{old}) \leftarrow s_i(_{new})$ $s_i(_{new}) \leftarrow (s_i \ll L/4) \oplus (t_i \gg L/4) \oplus r1 \oplus r2$ $t_i(_{old}) \leftarrow t_i(_{new})$ $t_i(_{new}) \leftarrow h(s_i(_{new}))$				6-Re-computes M3 Updates: $t_i \leftarrow h((s_i \ll L/4) \oplus (t_i \gg L/4) \oplus r1 \oplus r2)$

Figure 4.1: The authentication process of the Song protocol

3. Reader: The reader transmits r1, M1, and M2 to the server.
4. Server:
   (a) The server searches its database using M1, M2 and r1 as follows.
      i. It chooses t_i from amongst the values $t_i(_{new})$ or $t_i(_{old})$ stored in the database.
      ii. It computes M2'=f_{ti}(r1 $\|$ (M1 $\oplus$ t_i)).
      iii. If M2'== M2, then it has identified and authenticated T_i. It then goes to step (b). Otherwise, it returns to step (i). If no match is found, the server sends ε to the reader and stops the session.

   (b) The server computes r2 = M1 $\oplus$ t_i, and M3 = $s_i \oplus$ f_{ti} (r2 $\|$ r1), and sends M3 to the reader.

   (c) If the server found the tag's record in the new or old values, it updates:

$$
\begin{aligned}
&s_i(_{old}) \leftarrow s_i(_{new})\\
&s_i(_{new}) \leftarrow (s_i \ll L/4) \oplus (t_i \gg L/4) \oplus r1 \oplus r2\\
&t_i(_{old}) \leftarrow t_i(_{new})\\
&t_i(_{new}) \leftarrow h(s_i(_{new}))
\end{aligned}
$$

5. Reader: The reader forwards M3 to the tag T_i.
6. Tag: The tag T_i computes s_i = M3 $\oplus$ f_{ti}(r2 $\|$ r1) and checks that h(s_i) = t_i. If the check fails, the tag keeps the current value of t_i unchanged. If the check succeeds, the tag authenticates the server, and sets:

$t_i \leftarrow h((s_i \ll L/4) \oplus (t_i \gg L/4) \oplus r1 \oplus r2)$

In the next section 4.4, we will analyse the Song protocol in terms of security.

4.4 Security Analysis of the Song Protocol

Although in the Song protocol, the server stores the new and old value of the tag's data and they are updated after each authenticated session, data desynchronisation can be performed without compromising the internal data stored in the tag. The Song protocol will fail if an attacker intercepts the communication in two consecutive sessions. If the server's message (M3) is blocked in sequential sessions, the server database will have no matching data to complete the authentication, causing the data between the server and tag to be desynchronised. To elaborate, for example, in the first access to the tag, the server's values (s_{old}, t_{old}) are set to null, while (s_{new}, t_{new}) values are set to specific values assigned by the server, where (t_{new}) is equal to the tag's value (t_i). If the authentication succeeds, then (t_{new}) and (t_i) will be updated to the same value and (s_{old}, t_{old}) will take the previous values of (s_{new}, t_{new}). As shown in Table 4.2, if the attacker blocks M3 from reaching the tag, the server has updated the tag's data, while the tag will not update (t_i). In this situation, the value (t_i) in the tag will match the value (t_{old}) in the server and mutual authentication can still be achieved. Then, we suppose that the attacker blocks M3 in the consecutive session; then the tag will also not update (t_i), while (s_{old}, t_{old}) in the database have been updated with values not associated with the tag's data (t_i). As a result, the tag's data will not match the server's data, causing data desynchronisation and authentication failure.

4.5 Revised Protocol

We propose an improvement to the Song protocol by changing the updating process discussed in Section 4.4. In the Song protocol, if authentication is achieved, the server's data will be updated even if the matching record is found in (s_{old}) and (t_{old}). In the revised protocol, we propose that in the event of an authentication failure, whether it is due to communication error or intentional interference by an adversary, both the server and the tag should not update their values. Also, if the data is found in (s_{old}) and (t_{old}), the server's values should remain fixed.

Not updating the data does not affect location tracking, as the tag's messages and server's message include fresh random numbers.

Table 4.2: Data desynchronisation on the Song protocol

Server		Tag
In the first session: If the server authenticates the tag successfully, the server updates its data $s_i^2(_{old}) \leftarrow s_i^1(_{new})$ $s_i^2(_{new}) \leftarrow (s_i^1 \ll L/4) \oplus (t_i^1 \gg L/4) \oplus r1 \oplus r2$ $t_i^2(_{old}) \leftarrow t_i^1(_{new})$ $t_i^2(_{new}) \leftarrow h(s_i^1(_{new}))$		
	$\xrightarrow{M3\{blocked\}}$	
		the tag will not update t_i^1
In the next session:		
		The tag uses the current value of t_i^1 in calculating M1 and M2
	$\xleftarrow{M1,M2,R1}$	
The tag's data t_i^1 will match the old server's data $t_i^2(_{old})$ and authentication is still achieved, then the server updates its data $s_i^3(_{old}) \leftarrow s_i^2(_{new})$ $s_i^3(_{new}) \leftarrow (s_i^2 \ll L/4) \oplus (t_i^2 \gg L/4) \oplus r1 \oplus r2$ $t_i^3(_{old}) \leftarrow t_i^2(_{new})$ $t_i^3(_{new}) \leftarrow h(s_i^2(_{new}))$		
	$\xrightarrow{M3\{blocked\}}$	
		The tag will not update t_i^1
In the next session:		
		The tag uses the current value of t_i^1 in calculating M1 and M2
	$\xleftarrow{M1,M2,R1}$	
The tags data t_i^1 will not match the old server data $t_i^3(_{old})$ and authentication will not be achieved		

4.6 Summary

In this chapter, we have highlighted a design flaw in the Song protocol. We found that this protocol is vulnerable to data desynchronisation only if an attacker blocks the transmitted message from reaching the tag in consecutive sessions. This attack affects the availability of the tag in the next transaction. We proposed a revised protocol, which combats the desynchronisation incident possible in the Song protocol by changing the data update mechanism. In the next chapter, we propose our own RFID mutual authentication protocol which withstands the Song protocol desynchronisation attack and other well-known RFID active attacks.

Chapter 5

Mutual Authentication Protocol for Low-Cost RFID Tags

Contents

In this chapter we propose a new lightweight RFID mutual authentication protocol, which builds on the strengths of existing schemes and overcomes their weaknesses. We then carry out a security analysis of our proposed protocol in terms of informal and formal analysis using CasperFDR and Scyther tools. Subsequently, we implement the proposed protocol and present the performance measurement.

5.1 Introduction

RFID is being used in many applications that require security and privacy, such as access control systems, and authentication of products in the supply chains [111]. Therefore, there is a need for improved security measures to protect against active attacks, such as those discussed in Section 2.5.2. Wireless communication between tag and reader may allow an attacker to eavesdrop on a session, modify the transmitted messages, and prevent some messages from reaching their target. Moreover, a malicious entity may obtain the tag's data and/or track the tag's holder or the tag's location [111]. As shown in the previous chapter 4, the Song protocol takes into account the data desynchronisation issues but it is still vulnerable to such incidents. Hence, in this chapter we attempt to overcome active RFID attacks, including vulnerability in the Song protocol.

Another area that affects the adoption of RFID systems is performance. A low-cost RFID tags cannot perform computationally intensive security cryptographic functions, as it offers tightly constrained computational power and storage capacity [64].

The protocol proposed in this chapter can fit into systems that require user privacy and security. The protocol uses lightweight functions, such as PRNG and hash functions that can be implemented into constrained devices, such as low-cost RFID tags. For example, the proposed protocol, can be used in access control to authorise people holding RFID tokens to access a building. Moreover, other system, such as in supply chains, where tagged products need to be identified and authenticated by nearby reader(s). Such systems require security to confirm that only authorised people and products are being authenticated, and at the same time preserve their privacy.

5.2 Related Work

In this section, we present the mutual authentication protocols proposed so far for solving the security and privacy concerns associated with the use of RFID systems. Some common best-practice cryptographic approaches require more memory and/or processing power than would be feasible for cost-effective RFIDs. Hence, lightweight cryptographic primitives have gained more attention. For example, using optimised PRNG and cryptographic hash functions would suffice, assuming that it requires significantly fewer resources than the public key and symmetric key approaches [36]. In the following section, some of the proposed lightweight solutions are explained more thoroughly.

5.2.1 Hash Function-Based Protocols

Considering hash function-based authentication protocols for RFID, in 2003, Weis et al. [93] (referred to hereafter as WP) introduced the first lightweight protocol based on one-way hash functions; their scheme is called *hash-based access control*. The main idea of this scheme is to lock the tag from offering any functionality until it receives the correct secret key. The tag stores the hash of the secret key as a *meta-ID*; i.e. *meta-ID=hash(Key)*. When the tag receives the correct secret key, it calculates the hash of the received secret key, and compares it with *meta-ID*, if a match is found, it unlocks itself. Since then, attacks on this protocol have received a large amount of attention, and hundreds of papers have been published indicating ways to combat such attacks.

In [94], (referred to hereafter as W2) the authors pointed out that Weis et al.'s protocol [93] is prone to location tracking, as the attacker can simply eavesdrop the previous session, track the tag since the value of meta-ID is fixed, and replay the tag's message in the next session. Hence, they proposed the first *randomised access control scheme*, which is based on pseudo-random functions (PRFs). The tag uses a PRF to generate a random number (R) and calculates the hash on its ID and R in order to obtain different responses, preventing attackers from tracking the tag's location. A server then identifies the tag by performing an exhaustive search of all the stored tags IDs, until it finds a match to the value received from the tag. To unlock a tag, the server sends the matched tag's ID to the tag. However, the tag's response can be intercepted, thus allowing the attacker to perform replay and tag impersonation attacks.

Forward secrecy was introduced by Ohkubo et al. [96] (referred to hereafter as OP). The Ohkubo et al. protocol involves updating the tag's data whenever a tag is queried using a cryptographic hash function, which is presumably one-way. Hence, forward secrecy is guaranteed as the tag stores the new value of the data that are not used in the computation of previous messages, and the attacker cannot obtain the old values from the curent stored data. The i^{th} tag T_i and the server store a secret (s). T_i updates (s) in every session by using a hash function (h). Then in the next transaction, T_i uses a second hash function (g) and sends $g(h(s_{i+1})$ to the server, where (s_{i+1}) is the updated secret to be used in the next transaction. However, according to [97], this scheme is still vulnerable to replay attacks, and they proposed an extended protocol by adding random numbers to the exchanged messages to avoid replay attacks.

Another approach is called a challenge-response scheme. In this scheme, the reader sends a challenge to the tag, which can be a cryptographic nonce, and the tag sends a response to the reader's challenge. The use of a challenge-response scheme was motivated by Molnar et al. [115] (referred to hereafter as MP). Molnar et al. proposed using cryptographic nonces generated by the reader and tag to protect the privacy of the tag's data. The tag and the server share a key (k). The server sends a random number (r1) as a challenge to the tag. The tag generates a random number (r2), and sends M1=ID $\oplus$ $f_k(0\| \text{ r1 } \| \text{ r2})$, where (f) is a PRF, to the server in order to be authenticated. Once the server successfully authenticates the tag, it sends M2=ID $\oplus$ $f_k(1 \| \text{ r1 } \| \text{ r2})$ to be authenticated by the tag. However, according to [39], this proposal is vulnerable to forward secrecy invasion, as the value of the secret key is fixed.

Dimitriou [110] (referred to hereafter as DP) suggested similar approach to the protocol proposed in [115] but with more features, such as updating tag's data after each successful run of the protocol. Dimitriou proposed an RFID mutual authentication protocol to enforce user privacy. This approach uses a challenge-response protocol, a hash function, and a keyed hash function. The tag only stores the tag's identifier (ID), which serves as a key to calculate the keyed hash function. The server stores the tag identifier (ID) and the hash of the tag identifier (HID), which is used as an index to retrieve the tag's data. When the server successfully authenticates the tag, it updates (ID) to g(ID), where (g) is a hash function, and sends M3 = $f_{ID_i}(\text{r2} \| \text{ r1})$ to the tag using the updated ID_i, where r2 and r1 are random numbers generated by the tag and reader respectively. The tag checks the received value of M3. If the check is successful, the tag updates (ID) to g(ID). Nevertheless, this protocol is still vulnerable to DoS attacks [39]; if the third message (M3) sent by the server is blocked, then the server will update the identifier (ID) while the tag keeps the old value of the identifier, resulting in a desynchronisation between the server and tag.

In [103] (referred to hereafter as HP), the authors proposed an RFID lightweight protocol that uses PRNG and three hash functions. The detail of this proposal is shown in Fig. 5.1. The server and the tag shares the tag's ID, the secret key ($SK_{ID,i}$), and a counter (i). The server and the tag generate random number (X and a) respectively. As shown in Fig. 5.1, if the attacker blocks the tag's messages in consecutive sessions, the tag will not be authenticated by the server and data desynchronisation between the tag and the server occurs.

5.2.2 Lightweight Function-Based Protocols

Lightweight algorithms were used in RFID protocols, taking into account the limitation of the RFID tags. Motivated for this reason, several proposals suggested using the PRNG, the CRC and simple triangular operations, such as XoR, And and OR.

Duc et al. [114] presented an RFID mutual authentication protocol conforming to the EPCCIG2 standard (referred to hereafter as DP2). This scheme uses simple cryptographic primitives, such as PRNG and CRC functions, as they are supported in the EPCC1G2 standard. The PRNG is used to update the secret key while the CRC detects

Server		Reader	Tag
X $\leftarrow \{0,1\}^t$			
	$\xrightarrow{X}$	$\xrightarrow{X}$	
			a $\leftarrow \{0,1\}^t$ B= $H_0(SK_{ID,i}$, ID, i, X, a)
	$\xleftarrow{a,B}$	$\xleftarrow{a,B}$	
Re-computes B' , If there is a match: set d_Y=1 else computes: B_{i-1}= $H_0(SK_{ID,i-1}$, ID, i-1, X, a) If there is a match: set d_Y=1 Otherwise set d_Y=0 If d_Y=1, and B'=B Computes Z=$H_1(SK_{ID,i}$, ID, i, X,a) If d_Y=1, and B'_{i-1}=B Computes Z=$H_1(SK_{ID,i-1}$,ID,i-1,X,a) if d_Y=0, rnd $\leftarrow \{0,1\}^*$ Computes Z=H_1(rnd) *Key update:* If B'=B Compute s=$H_2(SK_{ID,i}$, ID, i) Sets $SK_{ID,i}$ = s Sets $SK_{ID,i-1}$ = $SK_{ID,i}$ Sets i=i+1 else, no update			
	$\xrightarrow{Z}$	$\xrightarrow{Z}$	
			Computes Z'=$H_1(SK_{ID,i}$, ID, i, X, a) If Z'=Z Sets d_Z=1, otherwise, sets d_Z=0 If d_Z=1 Compute s=$H_2(SK_{ID,i}$, ID, i) Sets $SK_{ID,i}$ = s Sets i=i+1 else, no update

Figure 5.1: Hanatanil et al.'s RFID authentication protocol

any errors occurring during the transmission of the messages. The server and tag store a secret key, which is updated after a successful authentication. However, this scheme still has some weaknesses, including vulnerability to data desynchronisation, replay attacks before the next successful authentication, and forward secrecy invasion[95].

Chien et al. [95] introduced an improved version of Duc et al.'s RFID mutual authentication protocol [114] (referred to hereafter as CP). The proposed protocol requires the server and tag to generate random numbers to prevent replay attack. The tag keeps a static EPC, which represents the unique identity of the tag, and an access key (K) and authentication key (P), which are updated after each successful authentication. The server also maintains the same values as well as the *old* and *new* access and authentication keys to avoid DoS attacks. The protocol uses simple cryptographic primitives, such as a PRNG and a CRC. After the server authenticates the tag, it updates the data except for the EPC identifier which is static. However, according to Peris-Lopez et al. [98], this proposal permits location tracking, tag impersonation, server impersonation, and backward traceability, because of the linear properties of the CRC.

Another improved version of Chien et al.'s protocol was proposed by Yi et al. [99]. However, again due to the use of the CRC function, [101] found that this proposal is also vulnerable to data desynchronisation, tag and reader impersonation, and traceability attacks.

A lightweight RFID mutual authentication protocol was proposed in [102] using a Shrinking Algorithm (referred to hereafter as SG). In this protocol a shrinking algorithm generates different random keys. The tag generates an encryption key (CSGK1) using the shrinking algorithm that takes a shared secret between the tag and the database (K1) as an input, and computes M=CSGK1 $\oplus$ (ID $\|$ S), where (ID) is the tag's ID, and (S) is a random number generated by the reader. Once the database authenticates the tag, it generates another encryption key (CSGK2) using the shrinking algorithm, and computes ID$'$=CSGK2 $\oplus$ ID. Finally, the tag and database will update their values. The authors claim that their protocol reaches the synchronization between the tag and database by maintaining a list of current and previous tag's data in the database. However, we found that this protocol is vulnerable to a desynchronisation attack, as if the attacker blocks the database's message from reaching the tag in consecutive sessions.

Song et al. [89] proposed an RFID authentication protocol for low-cost tags (referred to hereafter as SP). This protocol uses MAC and PRNG. Each tag stores the hash of a secret (s) namely (t), and the server stores the old and new values of the secret (s_{new}, s_{old}), the hashed secret (t_{new}, t_{old}) and the tag's information (D). This scheme uses a challenge-response protocol and supports updating data when the mutual authentication is achieved. Cai et al. [113] demonstrated that Song et al.'s protocol does not provide protection against tag impersonation attacks. Moreover, Rizomiliotis et al. [73] found that an attacker can impersonate the server without accessing the internal data of a tag and launch DoS attacks. Also, in Chapter 4, we pointed out that the new version of this protocol in [39] is prone to data desynchronisation.

Yeh et al. [105] proposed an improved version of Chien et al.'s RFID authentication protocol conforming to the EPCC1G2 standard (referred to hereafter as YP). The data kept in the server and tag is the same as in Chien et al.'s protocol, except that their protocol uses an index (C) to avoid DoS attacks and database overloading. The initialisation and authentication phases do not use CRC functions. Only PNRG functions are used, thus blocking the bad linear properties of the CRC function. Although this protocol prevents DoS attacks, it is still vulnerable to forward secrecy invasion, tag impersonation attacks, and server impersonation attacks as pointed out by [117].

An improved version of Yeh et al.'s protocol was proposed by Yoon in [117] (referred to hereafter as YoonP). Their protocols' data and the initialisation process are identical to Yeh et al.'s protocol, but the authentication phase adds a secret session random number (r2) to the exchanged massage (M1) generated by the tag. Yoon claimed that the proposed protocol provides more security than that of Yeh et al.'s protocol; however, [118] demonstrated that eavesdropping on only one session of the protocol and O(2^{16}) PRNG function evaluation can reveal the tag's secret data, as the length of the data generated from PRNG is only 16-bit strings, which makes it easier for the attacker to do an exhaustive search to reveal the data; it is thus easy to launch traceability attacks, DoS attacks, tag impersonation and server impersonation attacks.

As shown above, many RFID protocols have attempted to protect against a wide range of attacks, including DoS attacks, impersonation attacks, location tracking, replay attacks and forward secrecy invasion; however, cryptanalysis on proposed protocols is still ongoing. Designing a secure RFID protocol is still a challenging task, and this may be due to the following reasons:

- EPCC1G2-compliant RFID authentication protocols: Designing an EPCC1G2-compliant protocol is challenging because the only security operations available in this standard are a 16-bit PRNG and a 16-bit CRC. These functions do not provide irreversibility as in the hash function [112, 74].
- Updating data: Some of the protocols require the tag and the server to update internal data after each successful session in order to prevent location tracking, replay attacks and forward secrecy invasion. However, they do not take into account that the attacker might block certain messages more than once in two consecutive sessions causing a desynchronisation of data [66, 105].
- The deployment of inexpensive operators: The use of bitwise, and/or bit shifts operators is compatible with the tag's computational capabilities but it can lead to various security vulnerabilities [119, 120, 70, 112, 72, 73, 74].

Hence, we propose a new lightweight RFID mutual authentication protocol discussed in the next section, which overcomes the weaknesses highlighted in this section and builds on their strengths to provide a secure RFID system.

5.3 A New Lightweight RFID Authentication Protocol

In this section, we explain the proposed protocol in detail.

5.3.1 Design Goals

When designing a lightweight RFID mutual authentication protocol, consideration should be given to the following:

- Mutual authentication: The protocol should provide a mutual entity authentication. The communication should take place between legitimate entities, such as a tag, a reader and a server, and provide assurance to the receiver (server) about the identity of the sender (tag) and vice versa.

- Privacy: The tag's data should remain secret and not be revealed to any malicious entity, thus providing anonymity to the tag. Another notion related to privacy is untraceability; if the data being sent from the tag to the reader is static or linked to data sent previously, the tag holder's location can be tracked without his/her knowledge. Finally, if the tag's memory is compromised, the attacker should not be able to trace past transactions. Therefore, the protocol should provide a mechanism to achieve forward secrecy.

- Security: Due to the wireless communication between the tag and reader, an active attacker can observe and manipulate the communication channel between reader and tags. In this study, we focus on three common techniques to violate the secrecy of the system, namely replay attacks, data desynchronisation incidents and tag/server impersonation attacks. To elaborate, the designed protocol should provide:

    1. Resistance to replay attacks: The adversary can eavesdrop on the communication between reader and tag, obtain the exchanged message(s), and resend it repeatedly. Therefore, the generated messages should be fresh to the protocol session to protect against replay attacks.
    2. Resistance to desynchronisation incidents: Messages can be lost or the adversary can modify the flow of messages and block messages from reaching their target, causing a desynchronisation between the two legitimate parties. Therefore, the server should store the old and new values of the tag in order to authenticate the desynchronised tag and reach synchronisation.
    3. Resistance to tag/server impersonation attacks: In this attack, the attacker sends a message to the server that claims to come from a legitimate tag, and this message fabrication enables the attacker to masquerade as a legitimate tag and vice versa. Hence, the responses should not be sent in clear.

- Performance: The tag's memory storage space, and the computing cost should be appropriate for the tag's limited computing capability, and the amount of data communicated should be minimised.

5.3.2 Assumptions

We present a lightweight RFID mutual authentication protocol, which operates under the following assumptions:

- The reader contacts the tag through a wireless channel, which is susceptible to active attacks.
- The communication channel between the reader and server is secure.
- The tag's data are stored in non-volatile memory, such as EEPROM or Flash memory, where they can be updated.
- All the operations in the tag are atomic i.e. either all of the operations or none are processed. If the attacker kills the electromagnetic field between the reader and tag or the tag simply walks away from the reader's signal, the tag will execute all the computations simultaneously or not at all.
- The proposed protocol supports a multiple readers scenario, all connected to a central server, so that a tag can be read in many different locations.

5.3.3 Protocol Design

The proposed protocol has the following main features:

- Tags are capable of computing XoR operation, generating a pseudo-random number and calculating hash functions.
- The proposed protocol uses random numbers in an attempt to prevent location tracking and replay attacks.
- The server stores both the old and the new values of the data in order to prevent desynchronisation incidents.
- After a successful authentication between the server and tag, both parties update their values to be used in the next transaction.
- The reader does not store or update any data related to the tags.

5.3.4 Threat Model

From an information security standpoint, RFID is a challenging platform. The limited computational power in RFID tags makes it infeasible to perform common best-practice cryptographic approaches, such as public key cryptography schemes.

Accordingly, we used a popular adversary model called the Dolev-Yao model [67], where the adversary has powerful resources to control the communication channel via performing the following:

- Eavesdropping
- Modifying messages
- Blocking messages from reaching targets
- Replaying previous messages
- Injecting new messages (forgery)
- Impersonating any entity

Defences against relay attacks, physical attacks, side channel attacks, and power analysis attacks are not within the scope of this thesis.

5.3.5 Notation

The notation used in the proposed protocol are presented in Table 5.1:

5.3.6 Protocol Description

The scheme consists of two processes: initialisation, and authentication.

- Initialisation Process: This stage only occurs during manufacturing when the manufacturer assigns the initial values in the server, and in the tag. The initialisation process is summarised below:
    - The server assigns random values for each tag it manages to (ID_{new}, K_{new}) in the server, and (ID_i, K_i) in the tag.
    - Initially, (ID_{old}, K_{old}) in the server is set to null.
- Authentication Process: The authentication process is shown in Fig. 5.2.

    1. Reader: The reader generates a random number R1 of L bits and sends it to the tag.

Table 5.1: A summary of notation

Notation	Description
T_i	The i^{th} tag of the RFID system, where $1 \leq i \leq n$
ID_{old}	The i^{th} tag old ID
ID_{new}	The i^{th} tag new ID
K_{old}	The i^{th} tag old secret key
K_{new}	The i^{th} tag new secret key
ID_i	The i^{th} tag ID
K_i	The i^{th} tag secret key
x	The value kept as either new or old to show whether the tag uses the old or new values of ID_i and K_i
R1	A pseudo-random number generated by the reader
R2	A pseudo-random number generated by the tag
H	A hash function, $h:\{0,1\}^* \leftarrow \{0,1\}^L$
$A \leftarrow B$	The value of A is updated to that of B
$\oplus$	An XoR operation
$\|$	A concatenation operation
j	The transaction number
i	The number of the tag in the system
n	The number of tags managed by the server

2. Tag: The tag generates a random number R2 of L bits, and computes two messages as follows:

   $M1 = H(K_i \| R1 \| R2)$

   $M2 = H(ID_i \oplus R2)$

3. Tag: The tag sends R2, M1 and M2 to the reader.

4. Reader: The reader sends R1, R2, M1, and M2 to the server.

5. Server: For all the stored IDs, the server computes $H(ID_i \oplus R2)$ until it finds a match with the received value of M2:

   – If there is a match in ID_{new}, then the server marks x=new. The server retrieves data (ID_{new}, K_{new}), and re-computes M1, i.e., $M'1=H(K_{new} \| R1 \| R2)$ to authenticate the tag.

   – If there is a match in ID_{old}, then the server marks x=old, retrieves the data (ID_{old}, K_{old}), and re-calculates $M'1=H(K_{old} \| R1 \| R2)$ to authenticate the tag.

6. Server: The server computes $M3=H(ID_x \| K_x \| R1 \| R2)$, and transmits it to the reader.

7. Server: The server updates the data as follows:

Server		Reader		T_i
		1- Generates R1		
			$\xrightarrow{1-R1}$	
				2- Generates R2 Computes $M1=H(K_i \| R1 \| R2)$ $M2= H(ID_i \oplus R2)$
			$\xleftarrow{3-R2,M1,M2}$	
	$\xleftarrow{4-R1,R2,M1,M2}$			
5- Searches for ID_i Marks x=new or old Re-computes $M'1=H(K_x \| R1 \| R2)$ 6- Computes $M3=H(ID_x \| K_x \| R1 \| R2)$				
	$\xrightarrow{6-M3}$			
7-Updates: If x=new: $ID_{new}^{j+1} \leftarrow H(ID_{new}^{j})$ $ID_{old}^{j+1} \leftarrow ID_{new}^{j}$ $K_{new}^{j+1} \leftarrow H(K_{new}^{j} \oplus ID_{new}^{j+1})$ $K_{old}^{j+1} \leftarrow K_{new}^{j}$ Else if x=old: No update			$\xrightarrow{8-M3}$	9- Re-computes: $H(ID_i \parallel K_i \parallel R1 \parallel R2)$ Updates: $ID_i^{j+1} \leftarrow H(ID_i^j)$ $K_i^{j+1} \leftarrow H(K_i^j \oplus ID_i^{j+1})$

Figure 5.2: The proposed lightweight RFID mutual authentication protocol

If x=new, where ID_i is found in ID_{new}
$ID_{new}^{j+1} \leftarrow H(ID_{new}^{j})$
$ID_{old}^{j+1} \leftarrow ID_{new}^{j}$
$K_{new}^{j+1} \leftarrow H(K_{new}^{j} \oplus ID_{new}^{j+1})$
$K_{old}^{j+1} \leftarrow K_{new}^{j}$
Else if x=old, where ID is found in ID_{old}: No update

If there is no match in ID_{new} and ID_{old} or $M'1 \neq M1$, then the server keeps the tag's data the same, and sends an end session message to the reader to terminate the session.

8. Reader: The reader sends M3 to the tag.
9. Tag: The tag checks whether the received value of M3 is equal to $H(ID_i \parallel K_i \parallel R1 \parallel R2)$. If there is a match, the tag authenticates the server and updates its values to:
   $ID_i^{j+1} \leftarrow H(ID_i^j)$
   $K_i^{j+1} \leftarrow H(K_i^j \oplus ID_i^{j+1})$
   If the check fails or M3 is not received, the tag keeps the current values unchanged.

5.4 Protocol Analysis

In this section, we present the analysis of the proposed protocol in terms of informal, and formal analysis using CasperFDR and Scyther.

5.4.1 Informal Protocol Analysis

Our proposed protocol meets the following goals:

- Tag anonymity: The i^{th} tag stores two 128-bit values, namely ID_i and K_i that are supposed to be secret and not revealed to any entity except the legitimate server. The values of ID_i and K_i are protected during transmission using a secure one-way hash function. Moreover, it will take up to 2^{128} attempts to guess the value of ID_i or K_i.

- Tag location privacy (untraceability): In the proposed protocol, the tag's responses are changed with new updated values and fresh random numbers; thus an attacker will obtain new responses every time he/she eavesdrops on a session. Moreover, if the previous authentication session failed, and the tag's data remain unchanged, M1 and M2 messages will change due to the existence of random numbers (R1 and R2) generated by the reader and tag respectively.

- Forward secrecy: The values of ID_i and K_i are updated after each run in order to prevent forward secrecy invasion, using a hash function that is irreversible. If an adversary compromises the tag's memory, he/she will not be able to trace the previous communications of the tag, as the obtained messages involve the use of previous secret values ID_i and K_i, which are not stored in the tag. The stored updated values are used in the calculation of the next session and cannot be irreversible as a result of using a hash function.

- Resistance to replay attack: Our protocol utilises a challenge-response scheme. In messages M1 and M2, the tag sends the responses in which the reader's challenge (R1), and tag's random number R2 are included. The server must therefore include the tag's challenge R2 and R1 in its response (M3). Therefore, only legitimate parties (server+tag) can send valid answers, since random numbers are sent with secret values only known to the server and tag.

- Resistance to desynchronisation incidents: The communication session between the reader and tag may be accidentally or intentionally interrupted. Therefore, to avoid this sort of incidents, each time the server finds a match with the new values (ID_{new}, K_{new}), it updates the tag's data, and the old previous values of

Table 5.2: Comparison between the related work and our proposed protocol

	WP [93]	W2 [121]	OP [96]	MP [115]	DP [110]	DP2 [114]	CP [95]	SG [102]	PP [122]	SP [89]	YP [105]	YoP [117]	HP [103]	Our protocol 5.3
Tag anonymity	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Location privacy	$\times$	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$	$\times$	$\times$	$\surd$	$\surd$
Replay attacks	$\times$	$\times$	$\times$	$\surd$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Denial of service attacks	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$	$\times$	$\surd$
Tag impersonation attacks	$\times$	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\surd$	$\surd$	$\times$	$\times$	$\times$	$\surd$	$\surd$
Server impersonation attacks	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$

$\times$: Means does not provide protection $\surd$: Means resists such an attack

ID_i and K_i are maintained. Moreover, the server will not update its data either when there is no match, or if there is a match in (ID_{old}, K_{old}); it keeps the stored data the same. Thus, when the attacker blocks M3 more than once in consecutive order, the tag's data (ID_i, K_i) will still match the server's data (ID_{old}, K_{old}).

- Resistance to tag/server impersonation attacks: This is when an attacker attempts to impersonate a legitimate server to obtain information from a tag. This kind of attack is not feasible because the transmitted message (M3) includes secret values shared only by the tag and the server, and sent within a secure one-way hash function that is irreversible. The same applies to the tag impersonation.

- Mutual Authentication: All the exchanged messages include secret values (ID_i, K_i) sent using the hash function, so only a legitimate server and tag can calculate such messages, preventing any other from recovering and creating valid messages. The size of the data is 128-bit length, which means that the attacker needs to make 2^{128} attempts to recover the secret data.

In Table 5.2, we compare our protocol with the related research work in terms of the main requirements shown in Section 5.3.1. The result shows that our protocol is immune to the identified attacks, and provides better protection than the related work.

5.4.2 Formal Protocol Analysis

To formally analyse the proposed protocol and confirm that secrecy and authenticity between the server and tag are achieved, we used CasperFDR [104] and Scyther [75] tools.

CasperFDR Analysis of the Proposed Protocol

We prepared the CasperFDR script to obtain some indicative results if there is an attack on the protocol. The script is shown in Appendix A. We identified the following data in the #Free variables section:

T : Agent
S : Server
R1, R2 : Nonce
ID, K : Data
h: HashFunction
InverseKeys= (h,h)

The section #Specification, specifies the security and authentication requirements of the protocol, in which the Secret goals are:

Secret (T, K, [S])
Secret (T, ID, [S])

These goals indicate that the values of K and ID should only be known by the tag (T) and legitimate server (S).

The lines starting with Agreement are for providing authentication for instance:

Agreement (T, S, [ID, K])

This goal means that the tag is authenticated to the server using the data values (ID, K).

In addition, in the #Intruder information section, the intruder is defined to be Mallory, who can take full control of the session; impersonate any entity in the protocol, generate a random number (R3), read the messages transmitted in the network, intercept, analyse, and/or modify messages. The intruder is depicted as:

#Intruder Information
Intruder = Mallory
IntruderKnowledge ={Tag, ServerDB, Mallory, R1, R2, R3}

CasperFDR did not find any feasible attacks on the proposed protocol.

Scyther Analysis of the Proposed Protocol

Scyther performs a formal analysis of security protocols using a Dolev-Yao model [67] for an unbounded number of instances. It is mainly used to verify the authenticity of the exchanged messages between entities as in the proposed protocol considered here.

We conducted the analysis of our protocol with respect to three goals, namely secret, aliveness and agreement. The script is shown in Appendix A.2. Three roles are defined, namely a server (S), a reader (R), and a tag (Ti). The random numbers R1 and R2 are defined as Nonce; and IDi (tag identifier), and Ki (tag key) are defined as Data. The XoR and hash functions are defined as global functions.

Both roles, server and tag, share the secret goal over the two secret values IDi and Ki as follows:

claim_ S1(S, Secret, IDi);
claim_ S2(S, Secret, Ki);
claim_ Ti1(Ti, Secret, IDi);
claim_ Ti2(Ti, Secret, Ki);

Also both roles claim to be alive and share the agreement and synchronisation goals as follows:

claim_ S3(S, Alive);
claim_ S4(S, Niagree);
claim_ Ti3(Ti, Alive);
claim_ Ti4(Ti, Niagree);
claim_ Ti5(Ti, Nisynch);

After compiling the Scyther script, it did not find any feasible attacks within bound.

5.5 Protocol Implementation and Performance Measurement

For our experiment, we implemented the proposed protocol using the tools discussed in Chapter 3. We present the performance measurements after running the protocol for 100 runs on the DemoTag.

5.5.1 Implementation Process

The tag is provided with two 128-bit secret values: ID$_i$ and K$_i$, which are stored in the tag's EEPROM. Because we did not implement the protocol on the server, we assume that the reader knows ID$_i$ and K$_i$.

The reader starts by generating a 128-bit (16-byte) random number (R1) using the .NET Framework *rngCsp* method. Then, it sends R1 to the tag by sending eight *WriteTagData_ EPC_ C1G2* commands; each command writes 2 bytes of data. When the tag receives the reader's random number, it generates a random number R2 (128-bit) using the built-in PRNG, computes the two messages, namely M1 and M2 , each of which is 128-bit length. Subsequently, the tag writes the messages (R2, M1 and M2) in its memory using the *syscall_ writeWord* commands ready to be read on demand. The reader later sends three *ReadTagData_ EPC_ C1G2* commands to read R2, M1 and M2, and re-computes M1. If there is a match with the received M1, it calculates M3 and updates the values of ID$_i$ and K$_i$. Finally, the reader sends 8 *WriteTagData_ EPC_ C1G2* commands to the tag representing M3. The tag re-computes M3 to authenticate the reader. If successful, the tag updates ID$_i$ and K$_i$.

5.5.2 Performance Measurement

The performance measurements are as follows:

1. DemoTag memory cost: In the proposed protocol, the memory cost is:
    - 348 bytes are used from the 4 KB EEPROM memory for storing tag's data, messages (responses), and random numbers.
    - 33 KB used from the 128 KB Flash memory to store the tag's firmware.

Table 5.3: Data exchange time cost (milliseconds)

	Tag	Reader
Read	-	240
Write	1.28	985.74

2. Communication cost is shown in Table 5.3: For the tag to write R2, M1 and M2 into its memory to be read by the reader, the average timer counter after running the protocol 100 times is 20484.48, so the time cost, based on equation 3.1, is:

$$20484.48 * 0.0000000625 = 0.00128sec \equiv 1.28ms \tag{5.1}$$

Regarding the reader, after 100 runs, the average communication cost is: around *985.74 ms* to write R1 and M3 into the tag's memory, and *240 ms* to read the whole tag's response.

3. DemoTag computing cost: In a successful run of the proposed protocol, the tag generates R2, computes three messages (M1, M2, and M3), updates two values and write its messages in the memory. Table 5.4 demonstrates that the time cost of running the whole protocol on the DemoTag is around *9.31 ms*, which means that the tag can respond to reader's query in less than a second, and this demonstrates the relatively low computing cost of the protocol on the tag.

Table 5.4: Computing operations time cost (milliseconds)

	R2	M_1	M_2	M_3	Update	Write
Computing cost	0.11	1.61	1.52	1.66	3.13	1.28
Total	9.31					

The protocols discussed in this chapter did not provide any performance measurements, thus we could not present a performance comparison between our protocol and other related work.

5.6 Summary

In this chapter, we proposed our own RFID mutual authentication protocol that attempted to meet the identified goals and avoid the weaknesses found in related works and in the Song protocol. The proposed protocol has been informally and formally analysed, and the results demonstrated that our proposal can resist active RFID attacks. Moreover, the performance measurement demonstrated that the protocol execution time on the tag requires only *9.31 ms*. So far we have ignored one vital security aspect of secure RFID systems; the secret key distribution problem. The following chapter aims to address this aspect in RFID-enabled supply chains.

Chapter 6

Enhancing the Key Distribution Model in the RFID-Enabled Supply Chains

Contents

This chapter discusses the use of secret-sharing strategies as a promising solution for managing key distribution and recovery in the RFID-enabled supply chains. Firstly, we provide an overview of how RFID technology can be used in RFID-enabled supply chain systems. Existing approaches for distributing secret keys among tagged products in supply chains are discussed. Then, we point out the weaknesses found in the related work, and propose our enhanced scheme. We also present the analysis of our proposal, and finally illustrate the implementation of the proposed protocol.

6.1 Introduction

As already discussed in Chapter 2, RFID technology is extensively used in various applications. One of these applications is supply chain systems, where millions of inbound and outbound products move from manufacturers to customers [107]. These products should be correctly identified, verified and sorted at different points in the supply chain. In an RFID-enabled supply chain, the product information is embedded inside the RFID tag's memory, and this information can be efficiently collected, tracked, shared, and managed remotely via a nearby reader(s).

Shi et al. [123] classified the nodes in an RFID-enabled supply chain into five categories as follows:

1. Entry node, where the supply chain starts; for example in a manufacturing plant.
2. Aggregation node, where a number of cases are grouped into a larger unit; for example, in a distributor centre (DC).
3. Disaggregation node, where large units of products are disaggregated into smaller units, such as in a DC.
4. Normal node, where the cases are transported via land, air or sea logistics systems.
5. End node, where the supply chain ends; for example, in a retail shop.

As shown in Fig. 6.1, a manufacturer creates the products, embeds them with RFID tags that store unique data, stores these data in the server for further processing, then groups the products in cases, and ships them to the DC by land, air and/or sea. After that, the DC decomposes cases, then recomposes them into larger or smaller cases based on the next regional DCs, and ships them to the next regional DCs. Finally, the cases in the regional DCs are distributed to retailers. Typically, the number of cases starts off in large units (such as pallets) and these are reduced to smaller units as they make their way from the manufacturer to the retailers [107].

It is assumed that the manufacturers and distributor centres have physically secure environments, whereas areas outside these environments are prone to attacks as they are open to adversaries. Hence, protocols emerged to protect tags' data and preserve tags' anonymity in the retail and consumer environments [38, 70, 107, 137, 125]. In this chapter, we propose a protocol for protecting tags' data in an RFID-enabled supply chain, and secondly, we present a key update protocol incorporating a resynchronisation capability to counter the disruptive effects of location tracking, replay attacks and

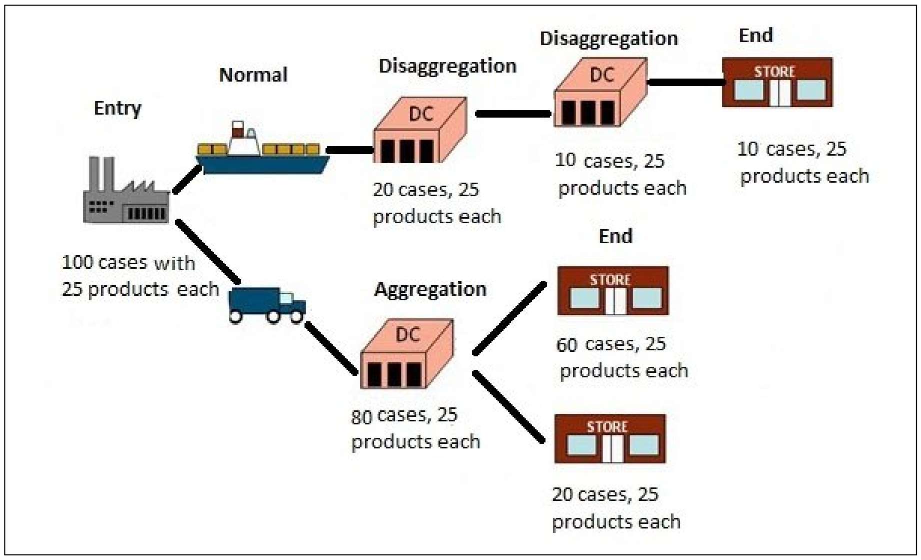

Entry
Normal
Disaggregation
Disaggregation
End
DC
DC
STORE
100 cases with 25 products each
20 cases, 25 products each
10 cases, 25 products each
10 cases, 25 products each
Aggregation
End
DC
STORE
STORE
80 cases, 25 products each
60 cases, 25 products each
20 cases, 25 products each

Figure 6.1: Supply chain parties [137]

desynchronisation attacks [126]. In the next section 6.2, we discuss the current problem in adopting RFID technology in RFID-enabled supply chain, and introduce our proposed solution.

6.2 Secret-Sharing Approach

RFID has captured the attention of many leading supply chain companies that want to make this technology feasible [137]. RFID technology enables a supply chain to identify, track and verify products remotely. To use this technology, all parties have to store the product data in their databases, and these data should be protected during transmission.

One way to secure the transmission of data between the reader and tags, is to encrypt the exchanged data with a secret key. Basically, secret key distribution must rely on secure channels established through pre-existing trust relationships. However, in supply chain practice, especially for ad-hoc supply chain structures, there is a lack of trust between the parties involved as the products' manufacturer may not know the next owner (distributor and/or retailer) [137]. So, the question remains of how to distribute the secret key safely. Proposals have been made for distributing the secret key securely in the RFID-enabled supply chain using the secret-sharing approach proposed by Adi Shamir [127]. Shamir [127] proposed a secret-sharing approach, where a secret can be divided into n parts (shadows) that, individually, do not provide any useful information about the secret. To reconstruct the secret, not all the parts are needed; any k of the

Table 6.1: A summary of notation

Notation	Description
R	Reader
T_i	The ith tag of the RFID system, where $1 \leq i \leq n$
K	A symmetric secret key
ID_i	The ith tag ID
EPC_i	The ith tag Electronic Secret Code value
S_i	The ith tag share of a secret key *K*
$E_K\{M\}$	A symmetric key encryption on a message M
h	A hash function, h:$\{0,1\}^* \leftarrow \{0,1\}^L$, where L is equal to the length of the data
n	The number of tags
$\oplus$	An XoR operation
$\|$	A concatenation operator
$A \leftarrow B$	The value of A is updated to that of B

parts are sufficient to reconstruct the original secret. This scheme is called a *(k, n)* threshold scheme. The generation of the shares is briefly shown below.

6.2.1 LaGrange Interpolating Polynomial Scheme

Adi Shamir uses polynomial equations in a finite field to construct a threshold scheme [127].

Let *p* be a prime, which is larger than the number of possible shadows and larger than the secret. For a given secret *M*, generate an arbitrary polynomial of degree *M-1*, i.e.:

$$F(x) = (ax^{k-1} + bx^{k-2} + ... + cx + M) \bmod p$$

The coefficients (*a, b, c*) are chosen randomly, and kept secret. The shadows are obtained by evaluating the polynomial at *n* different points:

$$k_i = F(x_i)$$

6.3 Related Work

In this section, we present proposals based on secret-sharing approach in RFID-enabled supply chains. For the rest of this section, we will use the notation summarised in Table 6.1.

Langheinrich et al. [108] proposed a "Shamir tag", which was the first proposal based on dividing the secret tag's ID into shares in the RFID systems. This approach splits the ID of a tag into multiple shares based on Shamir's secret-sharing scheme [127], and stores all the shares on the tag itself. These shares are concatenated to form the new ID of the tag. Following a reader's inquiry, an initial set of random bits from the new ID is released, followed by subsequent throttled single-bit releases. Once the entire new ID is released, the reader can compute the original ID. In this scheme, an RFID reader requires several minutes to recover the ID, which is not practical in a supply chain, where a large number of tags need to be processed in an efficient manner [70].

Li et al. [137] proposed another secret-sharing scheme called "Resilient Secret-Sharing (RSS)". They designed a secure and practical key distribution system between three parties (A, B, C) in the supply chain. Each tag stores two shares; one share belongs to the secret key *K1* between A and B, and the other share is intended for the secret key*K2* between A and C. The secret key *K1* is divided into multiple shares that are stored in n tags, and the remaining shares *(r-k-n)* are stored in the database, where r is the number of shares, and k is the number of shares required to reconstruct the secret key. The same process is done on the secret key *K2*.

In [107], Juels et al. proposed a secret key-sharing approach to be used within RFID-enabled supply chains to protect the transmission of the tags' EPC values. Their proposal complies with the predominant RFID standard EPCC1Gen2 discussed in Section 2.3.1. This approach does not require any computations on the tag side; the tag just stores two values and sends them to the reader.

The authors suggested the following:

- The manufacturer generates a secret key K and splits it into shares using the Reed-Solomon ECC-based secret-sharing scheme [109].
- Using a threshold scheme *(k, n)* where $k \leq n$, the secret key is divided into n shares, but only k shares are needed to reconstruct the secret key.
- The ith tag T_i stores two values: share (S_i) and symmetrically encrypted information ($\mathrm{E}_K\{\mathrm{EPC}_i\}$).
- Each T_i sends (S_i, $\mathrm{E}_K\{\mathrm{EPC}_i\}$) to any distributor/retailer reader who queries it.
- When the reader receives k shares from the products' tags, it recovers the secret key, and obtains the EPC value for each tag by decrypting ($\mathrm{E}_K\{\mathrm{EPC}_i\}$).

Juels et al. claimed that their protocol provides the following features:

1. This scheme provides an efficient solution for tag ownership transfer, as there is no need to distribute the keys in multiple supply chain databases.

2. An adversary will not be able to recover the secret key when he/she obtains the two values (S_i , $E_K\{EPC_i\}$) from a customer's tag or from the tags in the retail shop, as he/she needs to collect k shares to recover the secret key and decrypt ($E_K\{EPC_i\}$).

3. This scheme assumes that the manufacturer and distributor centres are secure areas, where the adversary does not have access, only legitimate parties can collect the shares.

However, Li et al. [137] claimed that Juels et al.'s proposal renders the system impractical, because it is difficult for any intermediate party in the supply chain to change the threshold of the shares, since the manufacturer pre-assigned all the shares to the tags according to a fixed secret-sharing scheme. In other words, if a group of products is aggregated or disaggregated into larger or smaller group, the threshold should be changed according to the new number of groups. Thus, the model should support updating the threshold scheme.

In addition, Cai et al. [70] pointed out that the tag's response can be tracked, as (S_i, $E_K\{EPC_i\}$) are fixed and sent to any reader that queries it. Also, an adversary can obtain the fixed tag's reply, and thus he/she is able to counterfeit the tag. Hence, Cai et al. [70] proposed an enhanced protocol based on Juels et al.'s scheme to avoid tag location tracking and counterfeiting attacks. They presented a protocol for updating the secret key (K) and shares (S) after recovering the secret key and authenticating the tag successfully, so the tag will respond with new values in each session. Cai et al.'s protocol is depicted in Fig. 6.2.

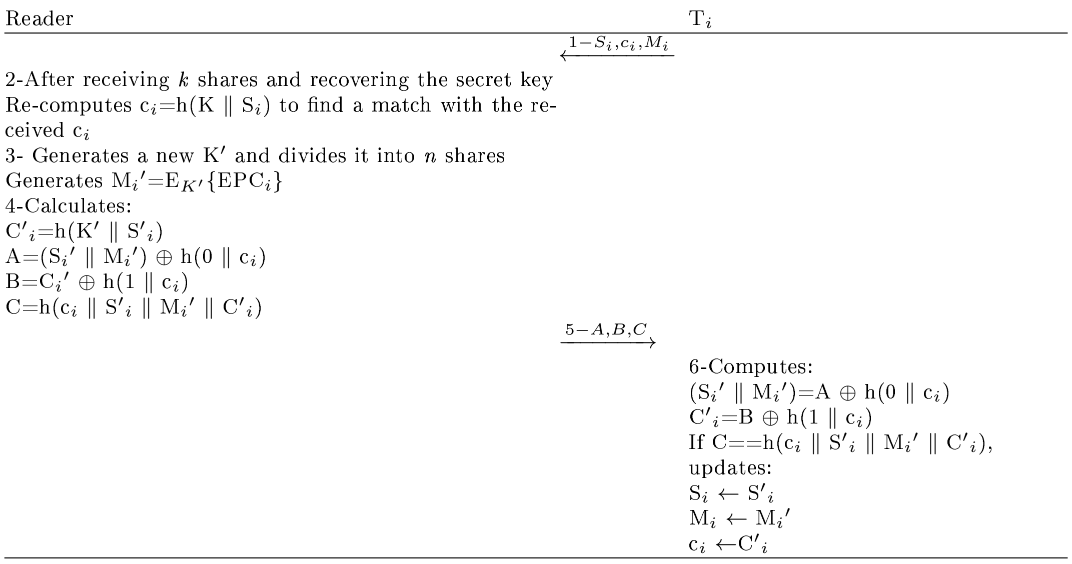
The authors proposed storing a new value (c) to serve as authenticating the reader before updating the data on the tag. This value is stored in the tag, i.e., c_i=h(K $\|$ S_i). The tag responds to the reader query with three values: (S_i, c_i, and M_i), where M_i=E_K(EPC_i).

During manufacturing, the manufacturer assigns the initial data (S_i, c_i, M_i) to the tag. The protocol in [70] is described as follows:

1. Tag: The tag T_i sends (S_i, c_i, M_i) to the reader.

2. Reader: After receiving k shares and recovering the secret key, the reader calculates c_i=h(K $\|$ S_i) to find a match with the received c_i. If there is a match, it authenticates the tag.

3. Reader: If the reader successfully authenticates the tag T_i, the reader generates a new secret key K$'$, divides it into n new shares (S'_i) and calculates M_i'=$E_{K'}\{EPC_i\}$.



Reader | T_i

$\xleftarrow{1-S_i,c_i,M_i}$

2-After receiving k shares and recovering the secret key
Re-computes c_i=h(K ∥ S_i) to find a match with the received c_i
3- Generates a new K′ and divides it into n shares
Generates M_i'=$E_{K'}${EPC$_i$}
4-Calculates:
C'_i=h(K′ ∥ S'_i)
A=(S_i' ∥ M_i') ⊕ h(0 ∥ c_i)
B=C_i' ⊕ h(1 ∥ c_i)
C=h(c_i ∥ S'_i ∥ M_i' ∥ C'_i)

$\xrightarrow{5-A,B,C}$

6-Computes:
(S_i' ∥ M_i')=A ⊕ h(0 ∥ c_i)
C'_i=B ⊕ h(1 ∥ c_i)
If C==h(c_i ∥ S'_i ∥ M_i' ∥ C'_i),
updates:
S_i ← S'_i
M_i ← M_i'
c_i ←C'_i

Figure 6.2: Cai et al.'s secret key update protocol

4. Reader: For the ith tag T_i, the reader calculates:

   - C'_i = h(K′ ∥ S'_i)
   - A = (S_i' ∥ M_i') ⊕ h(0 ∥ c_i)
   - B =C'_i ⊕ h(1 ∥ c_i)
   - C = h(c_i ∥ S'_i ∥ M_i' ∥C'_i)

5. Reader: The reader sends (A,B,C) to T_i.

6. Tag: After receiving (A,B,C) from the reader, T_i computes:

   - (S_i' ∥ M_i') = A ⊕ h(0 ∥ c_i)
   - C'_i = B ⊕ h(1 ∥ c_i)

   If C = h(c_i ∥ S'_i ∥ M_i' ∥C'_i), the reader is authenticated. Then T_i updates its values to:

   - S_i ← S'_i
   - M_i ← M_i'
   - c_i ←C'_i

Cai et al. assumed that any place outside the manufacturing area is insecure. Cai et al. claimed that their protocol is immune against location tracking, as the tag data are updated after a successful key recovery process. Thus, the tag will reply with new data every time the reader queries it.

We found two attacks on their protocol were possible: desynchronisation attack and location tracking. The attacks are shown below:

1. The intruder eavesdrops on a normal session between the tag and the reader.

2. The intruder captures the tag's values (S_i, c_i, and M_i).

3. The intruder blocks the A, B and C messages from reaching the tag, resulting in an update of the tag's data on the reader but not on the tag. Therefore, the reader will not authenticate this tag in future due to the mismatch between the reader's and tag's data.

4. Since the tag does not receive any data from the reader,it will reply with the same data (S_i, c_i, and M_i) for every query it receives, and these data can be used to trace the tag's location.

5. The attacker can then replay the captured messages in another session, thus causing the tag to be updated with old values.

   Therefore, Cai et al. protocol will not provide a mutual authentication between the reader and the tag when a desynchronisation incident occurs. As a result of such data desynchronisation, the tag will reply with the same values, thus permitting tracking of the tag's location. In Section 6.4, we attempt to address the weaknesses found in this work.

6.4 Enhancing the Key Distribution Model in the RFID-Enabled Supply Chains

In order to solve the security problems found in previous works while maintaining their merits, we propose a secure key management and recovery model as an enhancement to the Juels et al.'s model and Cai et al.'s secret key update protocol. Our proposal is illustrated below.

6.4.1 Design Goals

Juels et al.'s scheme does not preserve the tag's location privacy, and is prone to tag impersonation attacks. Cai et al.'s protocol is also vulnerable to location tracking and desynchronisation attacks. Therefore, our proposed model should avoid such vulnerabilities and meet the following goals:

- *Privacy:* The designed scheme must achieve three important goals related to privacy, namely:
  - Tag anonymity: RFID tags should provide a mechanism for preventing the tag information from being revealed to any malicious entity. For example, encrypting the tag's reply will only allow an authorised entity to decrypt it, such as a server.
  - Untraceability: If the data being sent from the tag to the reader is static or linked to data sent previously, the tag holder's location can be tracked without his/her knowledge. Therefore, the RFID tag's data should be anonymous and unlinkable.
  - Forward secrecy: The proposed protocol should ensure that if an attacker compromises the tag's memory, he/she will not be able to trace previous communication session(s) using previously known messages.
- *Security:* The designed protocol should resist the following attacks:
  - Replay attacks: The adversary may eavesdrop on the communication between a reader and tag, reuse the data and send it repeatedly.
  - Desynchronisation incidents: The adversary may eavesdrop on the communication between a reader and tag, and block messages from reaching their target, thus causing a data desynchronisation between the tag and server if there is an update process.
  - Tag and server impersonation attacks: The attacker may send a message to the server that claims to come from a legitimate tag, and this message fabrication enables the attacker to masquerade as a legitimate tag. The same applies to the server impersonation.
- Mutual authentication: The scheme should provide a mutual entity authentication, where the communication should take place between valid tags and server, and provide assurance to the receiver (server) about the identity of the sender (tag) and vice versa.
- Performance: The tag's memory storage space, and the computing cost should be appropriate to the tag's limited resources.
- Flexibility: The proposed model should allow all the distributor centres that participate in the supply chain to change the threshold parameters according to the new groups of products re-packaged into larger or smaller cases.

6.4.2 System Scenario

The general structure of our proposal is as follows:

1. *Manufacturer initialisation process*: To dispatch the products to the next distributor, the manufacturer generates a secret key for the products, specifies the threshold, and divides it into shares. Then, the manufacturer stores one share in each product's tag. Finally, it dispatches the tagged items to the distributor centre.

2. *Distributor key recovery process*: The distributor collects the required number of shares to recover the secret key for identifying the products. Then, it generates a new secret key and divides it into new shares, and specifies a new threshold based on the aggregated/disaggregated products.

3. *Distributor secret key update process*: After specifying the new threshold to recover the new secret key, the distributor updates the tags' stored data with the new shares and encoded-IDs, where encoded-ID_i = ID-$\mathrm{Tag}_i = \mathrm{E}_{K_T}\{\mathrm{ID}_i\}$

4. *Retailer key recovery process*: The retail shop recovers the secret key by collecting the required number of shares to identify each tagged product.

6.4.3 Assumptions

The proposed system in this chapter operates under the following assumptions:

- Passive tags are capable of computing XoR operation, generating a pseudo-random number and calculating hash functions.
- The reader contacts the tag through a wireless channel that is susceptible to attacks.
- The communication channel between the reader and the server is secure.
- The tags can be read by a single reader upon arrival.
- The tag's data is stored in a non-volatile memory, such as EEPROM or Flash memory, where they can be updated.
- The number of shares should be large to prevent the attacker from obtaining the tag's ID using the recovered secret key.
- All the operations in the tag are atomic.

6.4.4 Threat Model

We use the Dolev-Yao adversary model [67], where the adversary has powerful resources to control the communication channel by performing the following:

- Eavesdropping
- Modifying messages
- Blocking messages from reaching targets
- Replaying previous messages
- Injecting new messages (forgery)
- Impersonating any entity

6.4.5 Protocol Description

We propose a secure and flexible key distribution and recovery model for use in the RFID-enabled supply chains. The proposed scheme is described below:

A. Manufacturer initialisation process:

   The main goal of this process is to protect tags' transmitted IDs from being revealed to any entities except the legitimate distributors and retailers. The manufacturer does the following:

   (a) The manufacturer generates a random number (R1).

   (b) The manufacturer calculates K_T=h(R1), where K_T is the secret key for the products to be dispatched to the next distributor.

   (c) The manufacturer specifies the threshold *(k, n)* for recovering the (K_T) secret key, then splits K_T into n shares (S_n) using the threshold scheme discussed in Section 6.2, and stores one share (S_i) in the i^{th} product's tag.

   (d) For each tag T_i, the manufacturer computes C_i=h($K_T \oplus S_i$) and stores C_i in T_i's memory for authenticity purpose.

   (e) To ensure privacy of the tag's data, such as the tag's ID during transmission in insecure environments, the manufacturer encrypts each i^{th} tag's ID value and stores it in the tag's memory, i.e., ID-Tag$_i$=$E_{K_T}\{ID_i\}$, where E_{K_T} represents a symmetric key encryption using K_T. The tag also stores its ID_i.

   (f) The manufacturer dispatches the tagged products to the next distributor.

B. Distributor key recovery process:

In this section, we discuss the key recovery process when all the products reach the distributor. The distributor uses the threshold scheme to recover the secret key (K_T), decrypts the encoded-ID (ID-Tag_n), and obtains the ID_n values, where n is the total number of products.

- Reader: When the distributor ensures that all the expected products have arrived, the reader scans the products' tags.
- Tag: Each tag sends (S_i, C_i, ID-Tag_i) to the reader.
- Reader: The reader collects k shares, and sends the collected shares and the tags' messages (C_n, ID-Tag_n) to the server.
- Server: The server recovers the secret key (K_T) based on the received collected shares.
- Server: The server decrypts ID-Tag_i for each product's tag to retrieve the tag's ID value.
- Server: The server re-calculates C_i=h($K_T \oplus S_i$) for each tagged product to authenticate it. Then, it performs the next step.

C. Distributor secret key update process:

The main goals of updating the secret key and the threshold are to prevent location tracking and counterfeiting attacks. If S_i, C_i and ID-Tag_i are fixed, the attacker will be able to trace the location of the tag and/or obtain such data to counterfeit a legitimate tag. The secret key update process is shown in Table 6.2. For simplicity, in Table 6.2 we refer to the server as a reader and a server.

Once the server has recovered the secret key, and obtained ID_i for each product's tag T_i as shown above, it does the following:

1 Server: The server generates a random number (R1), and sends it to the tag. The tag then generates a random number (R2), and sends it back to the server.

2 Server: The server generates a random number (R), then, calculates K'_T=h(R), where K'_T is the new secret key for the products.

3 Server: For each tag, the server computes ID-$Tag2_i$=$E_{K'_T}\{ID_i\}$ for the i^{th} tag in the system.

4 Server: It specifies the new threshold parameters (k_{new}, n_{new}) based on the number of the new group of products to be dispatched, then divides K'_T into n new shares (S'_n).

5 Server: It calculates C'_i=h($K'_T \oplus S'_i$) for each i^{th} tag.

6 Server: To distribute the new values of S'_i and C'_i securely for the i^{th} tag, the server acts as follows:

- Calculates M1= h($ID_i \oplus$ ID-$Tag2_i \oplus R1 \oplus R2 \oplus C_i$) $\oplus S'_i$.
- Calculates M2= h($S'_i \oplus ID_i \oplus R1 \oplus R2$) $\oplus C'_i$.
- Calculates M3= h($S'_i \oplus ID_i \oplus R1 \oplus R2 \oplus C'_i$).
- Sends M1, M2, M3, and ID-$Tag2_i$ to the tag via the reader.

7 Tag: When the tag receives the messages, it obtains S'_i and C'_i by calculating:

S'_i= M1 $\oplus$ h($ID_i \oplus$ ID-$Tag2_i \oplus R1 \oplus R2 \oplus C_i$)

C'_i= M2 $\oplus$ h($S_i' \oplus ID_i \oplus R1 \oplus R2$)

Then, the tag authenticates the server by re-calculating M3. If there is a match, the tag guarantees that the server has successfully recovered the secret key, decrypted ID-$Tag2_i$, and obtained ID_i. Subsequently, the tag updates its values to:

$S_i \leftarrow S'_i$

ID-$Tag_i \leftarrow$ ID-$Tag2_i$

$C_i \leftarrow C'_i$

8 Tag: To inform the server that the tag has received the new data, it calculates M4= H($ID_i \oplus S_i \oplus C_i \oplus R1 \oplus R2$), where S_i and C_i are the updated values, and sends M4 to the reader.

9 Reader: The reader sends M4 to the server.

10 Server: The server then re-calculates M4. If there is a match, the server guarantees that the tag has updated its values successfully, and sends an OK message to the reader.

11 Once the tags are embedded with the new data and arranged in the new group, the distributor dispatches them to the next receiver (distributor or retailer).

Table 6.2: Distributor secret key update process

Server		Tag_i
1- Generates R1	$\xrightarrow{R1}$	
	$\xleftarrow{R2}$	1- Generates R2
2- Generates R, calculates K'_T=h(R) 3- Encrypts ID-Tag2_i=$E_{K'_T}\{ID_i\}$ 4- Specifies new threshold, and divides K'_T into new n shares 5- C'_i=h($K'_T \oplus S'_i$) 6- Generates R2 Computes: M1=h($ID_i \oplus$ID-Tag$2_i \oplus R1 \oplus R2 \oplus C_i) \oplus S'_i$ M2= h($S'_i \oplus ID_i \oplus R1 \oplus R2) \oplus C'_i$ M3= h($S'_i \oplus ID_i \oplus R1 \oplus R2 \oplus C'_i$)		
	$\xrightarrow{M1, M2, M3, ID-Tag2_i}$	
		7- Computes: S'_i=M1$\oplus$h($ID_i \oplus$ID-Tag$2_i \oplus R1 \oplus R2 \oplus C_i$) C'_i=M2$\oplus$h($S'_i \oplus ID_i \oplus R1 \oplus R2$) Re-calculates M′3 If M′3==M3, updates its values to: $S_i \leftarrow S'_i$ ID-Tag$_i \leftarrow$ ID-Tag2_i $C_i \leftarrow C'_i$ 8- Calculates: M4=h($ID_i \oplus S_i \oplus C_i \oplus R1 \oplus R2$)
10- Recalculates M4	$\xleftarrow{M4}$	

If there is no match with the received M4 message, or if the reader does not receive M4 from the tag, the reader starts the *resynchronisation process* as shown below:

(a) Server: To re-distribute (S'_i, C'_i) securely for the i^{th} tag, the server acts as follows:

- Generates a new random number (R3), and sends it to the tag. The tag generates a random number (R4), and sends it to the server.
- Computes M5= h($ID_i \oplus$ ID-Tag$2_i \oplus R3 \oplus R4 \oplus C_i) \oplus S'_i$.
- Computes M6= h($S_i' \oplus ID_i \oplus R3 \oplus R4) \oplus C'_i$.
- Calculates M7= h($S'_i \oplus ID_i \oplus R3 \oplus R4 \oplus C'_i$).
- Sends a resynchronization request with M5, M6, M7, and ID-Tag2_i to the tag.

(b) Tag: When the tag receives the resynchronisation request, M5, M6, M7, and ID-Tag2_i, it re-computes M5, and M6 as in step C7 above.

i. If the received values of S'_i, ID-Tag2_i and C'_i are equal to the current values of S_i, ID-Tag_i and C_i, the tag assumes that M4 did not reach the reader, and keeps the values the same without update.

ii. If the received values of S'_i, ID-$Tag2_i$ and C'_i are not equal to the current values of S_i, ID-Tag_i and C_i, the tag re-computes M7, and if there is a match, it updates its values to:
$S_i \leftarrow S'_i$
ID-$Tag_i \leftarrow$ ID-$Tag2_i$
$C_i \leftarrow C'_i$

(c) Tag: The tag calculates M8= $H(ID_i \oplus S_i \oplus C_i \oplus R3 \oplus R4)$ and sends it to the reader to inform the reader that it received the new values. The process is iterated until it reaches an upper limit set by the system owner. If it reaches the upper limit, the reader should issue an error message that requires attention.

D. Retailer key recovery process:

At this stage, the retailer wants to authenticate all the required products' tags and retrieve their ID values. It recovers the secret key K_T to obtain the ID values from each tag.

The retailer key recovery process is summarised below:

(a) Reader: When the retail shop confirms that all the expected products have arrived, the reader scans the products' tags.

(b) Tag: Each tag sends (S_i, C_i, ID-Tag_i) to the reader.

(c) Reader: The reader sends the the collected shares and tags' messages (S_n, C_n, ID-Tag_n) to the server.

(d) Server: The server recovers the secret key (K_T).

(e) Server: The server re-calculates C_i for each tagged product to authenticate it.

(f) Server: The server decrypts ID-Tag_i for each product's tag using the recovered secret key (K_T) to retrieve the ID value.

6.5 Protocol Analysis

In this section, the proposed protocols are analysed informally and formally as shown below.

6.5.1 Informal Protocol Analysis

The proposed protocol meets the following goals:

- Tag anonymity: The aim of the proposed model is to protect the tag's ID, which is 128-bit long. The value of tag's ID is protected by encrypting it with a secret key. This key is not openly distributed, and is not stored in the participants' database. Also, the tag does not maintain the secret key in its memory. Moreover, it will take up to 2^{128} attempts to guess the secret key value.

- Untraceability: The tag's values of S_i, ID-Tag$_i$ and C_i are updated after each successful key recovery process, so the tag's responses will be different for every reader query. Hence, the attacker cannot track the tag's location. Moreover, the responses contain random numbers that are freshly generated in each session. The resynchronisation process plays an important role in preventing location tracking, as the reader keeps resending the new updated data until it confirms that the tag has successfully changed its data.

- Forward secrecy: In the proposed protocol, even if the attacker compromises the tag's memory and obtains the values of (S_i, ID-Tag$_i$, C_i), he/she cannot re-compute the previous values, as they are calculated using a secret key only known to the legitimate parties.

- Protection against replay attacks: All the messages transmitted in the communication channel are updated with new values and include fresh random numbers. Hence, the attacker cannot re-send the obtained messages. Our protocol uses a challenge-response scheme. In messages M1, M2 and M3, the server sends the tag's data, in which the server's random number R1 is included as a challenge. The tag must therefore include the server's random number (R1) in its response (M4). Similarly, the tag sends a challenge (R2) to the server, therefore, the server must include the tag's challenge (R2) in its responses (M1, M2, M3). Hence, only legitimate parties (server and tag) can send valid answers, since random numbers are sent with secret values only known to the server and tag.

- Protection against desynchronisation incidents: The new secret update protocol addresses the realistic scenario in which messages might not reach their intended recipient due to accidental or malicious interference. If the reader does not receive M4 from the tag, the reader will assume that the tag did not receive the reader's messages, or that M4 was lost during transmission, so the reader will start the resynchronisation process. Similarly, if the resynchronisation messages are blocked or M8 is lost, after timeout the reader will restart the resynchronisation process until it reaches an upper limit set by the system owner.

- Protection against server impersonation attacks: An attacker may attempt to impersonate the server and update the tag with incorrect values. However, the attacker cannot calculate M1, M2 and M3, as they involve a secret value, (ID_i), that is 128-bit long and only known to the tag and the server. The tag's ID is sent within a one-way hash function that is pre-image resistant.

- Protection against tag impersonation attacks: The attacker cannot send M4 on behalf of the tag, as it involves three secret values unknown to the attacker (ID_i, S_i, C_i), and they are not sent in clear; they are sent within a one-way hash function.

- Mutual authentication: The proposed protocol allows the distributor's server to access the tag and update the tag's data by sending authentication messages M3, which confirm that the server has successfully recovered the secret key and has obtained the right value of (ID_i). Similarly, the tag also sends M4, which confirms to the server that a legitimate tag has successfully changed the values of S_i, ID_i and C_i, which can only be obtained by a legitimate tag equipped with a valid ID_i and C_i values.

- Flexibility: In our model, any distributor party in the supply chain can generate new secret-sharing parameters (k_{new},n_{new}). Thus, a downstream party may choose $k_{new} \leq k$ and $n_{new} \leq n$ to process a small group of tags, or choose $k_{new} \geq k$ and $n_{new} \geq n$ to process a large group of tags.

As shown in Table 6.3, Cai et al. [70] presented several issues that we have addressed.

Table 6.3: Security features comparison

	Cai et al [70]	Our protocol 6.4
Reader impersonation	$\surd$	$\surd$
Tag impersonation	$\surd$	$\surd$
Desynchronisation attack	$\times$	$\surd$
Tag information privacy	$\surd$	$\surd$
Untraceability	$\times$	$\surd$
Replay attack	$\times$	$\surd$
Forward security	$\surd$	$\surd$
Mutual authentication	$\times$	$\surd$

$\times$: Means does not provide protection $\surd$: Means resists such an attack

6.5.2 Formal Protocol Analysis

This section presents the formal analysis of the *secret key update process* using CasperFDR [104] and Scyther [75] tools. The aim of this section is to prove that the data exchanged between the tag and reader is protected.

CasperFDR Analysis of the Proposed Protocol

The intruder's capability modelled in CasperFDR scripts for the proposed protocol is shown below:

1. An intruder can impersonate any entity in the network.
2. An intruder can read and maintain the messages transmitted by each entity in the network.
3. An intruder can intercept, analyse messages, and/or re-send any transmitted messages.

The script is shown in Appendix B.1. In the script we assumed that the key recovery process is already achieved, and the reader has generated the new values of K_T, S_i, C_i and ID-Tag. The defined variables are:

S, Ti: Agent
R1, R2: Nonce
ID, Ci, IDProductnew, Snew, Cnew: Data
h: HashFunction
InverseKeys = (h,h)

The main goals of the key update process are to ensure that the value of tag's ID is secret and the new shared values (S_{new}) and (C_{new}) are transmitted securely to the tag; this is depicted as follows:

Secret(S, ID, [Ti])
Secret(S, Snew, [Ti])
Secret(S, Cnew, [Ti])

We also specify the goal predicate *Agreement(S, Ti, [ID])*, where both the server and tag are agreed on the value of ID_i.

The intruder is defined to be Mallory, who knows all the entities, nonces, C message and ID-Tag value as shown below:

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Serveri, Tagi, Mallory, R11, R22, R33, CTag, IDProductNew}

The CasperFDR tool evaluated the protocol and did not find any attack(s).

Scyther Analysis of the Proposed Protocol

Similar to CasperFDR, the intruder is modelled using the channel (dy) [67]. The threat model is defined as an intruder, who has full control over the network, such that all messages sent by agents can be eavesdropped by the intruder. Moreover, the intruder may intercept, analyse, modify messages, and/or send any message he/she composes to other agents pretending to come from a legitimate agent.

Two roles are identified in the script: a server (S) and a tag (Ti) as shown in Appendix B.2. The variables defined are:

fresh ID: Data;
fresh Ci: Data;
fresh IDProductnew: Data;
fresh Snew: Data;
fresh Cnew: Data;
fresh R1 : Nonce;
var R2 : Nonce;

Each role specifies the goals that the protocol attempts to meet. These goals are within the *Claim* section. In the protocol, there are, for example, five *Claim* goals specified in the tag role as follows:

claim_ Ti1(Ti, Secret, ID), which means the value of the tag's ID should remain secret.
claim_ Ti2(Ti, Secret, Snew), which means the value of Snew (the new generated share (S'_i)) should remain secret.
claim_ Ti3(Ti, Secret, Cnew), which means the value of Cnew (the new generated share (C'_i)) should remain secret.
claim_ Ti4(Ti, Alive), which means that the protocol run is fresh and the tag Ti is alive.

claim_ Ti5(Ti, Niagree), which means that the tag Ti agree to the values of the variables.
claim_ Ti6(Ti, Nisynch), which means that the messages must be executed in the expected order as specified in the specification.

Similarly the goals specified in the server role are:

claim_ S1(S, Secret, ID);
claim_ S2(S, Secret, Snew);
claim_ S3(S, Secret, Snew);
claim_ S4(Si, Alive);
claim_ S5(S, Niagree);
claim_ S6(S, Nisynch);

After running the script, Scyther did not find any feasible attack within bounds, which means no attack was found within the bounded state space.

6.6 Protocol Implementation and Performance Measurement

In this section, we present the implementation process of the distributor secret key update process (successful run), and the performance measurement taken from DemoTag after 100 runs.

6.6.1 Implementation Process

The tag is provided with two 128-bit secret values, ID$_i$ and C$_i$, which are stored in the tag's EEPROM. The reader stores four 128-bit values, namely ID$_i$, S$_i$, ID-Tag$_i$ and C$_i$.

The reader generates a random number R1 (128-bit), and sends it to the tag using 8 *WriteTagData_EPC_C1G2* commands. The tag generates a random number R2 (128-bit) using the built-in PRNG, and writes it in the memory to be read by the reader. The reader reads the tag's random number, and generates a 128-bit random number (R) using the *rngCsp* method and calculates the hash of (R) to generate the secret key for the products (K'_T). Then, it encrypts the ID-Tag value with this secret key using the AES library built into the .Net framework. Following that, the reader computes C'_i. Subsequently, it computes 3 messages (M1, M2, M3) using the SHA256 library imported from the *Crypto-avr-lib SHA 256* library. Finally, the reader sends

M1, M2, ID-Tag and M3 (64-byte) to the tag by sending 32 *WriteTagData_ EPC_ C1G2* commands, each write command sends 2 bytes of data.

When the tag receives the reader's messages, it extracts S'_i form M1 and C'_i from M2. Then, it re-computes M3 using the extracted values to authenticate the reader. If there is a match with the received M3, it updates its values accordingly and uses the updated values in the calculation of M4. Finally, the tag writes (M4) in its memory using the *syscall_ writeWord* command ready to be read on demand.

The reader later sends *ReadTagData_ EPC_ C1G2* command to read M4, and re-computes M4. If there is a match with the received M4, it authenticates the tag and confirms that the tag has updated its values successfully.

6.6.2 Performance Measurement

The performance measurements for the *distributor secret key update process* are as follows:

1. DemoTag memory cost: In the proposed protocol, the memory cost is:
    - 348 bytes are used from the 4 KB EEPROM memory for storing the tag's data, messages (responses), and random numbers.
    - 32.7 KB used from the 128 KB Flash memory to store the tag's firmware.

Table 6.4: Data exchange time cost

	Tag	Reader
Read	-	160.66 ms
Write	1.8 ms	4.73 sec

2. Data exchange time cost is shown in Table 6.4: For the tag to write R2 and M4 into its memory to be read by the reader, the average timer counter after running the protocol 100 times is 28809.74, so, based on equation 3.1, the time cost is:

$$28809.7 * 0.0000000625 = 0.0018sec \equiv 1.8ms \tag{6.1}$$

   Regarding the reader, we found that the reader needs *4.73 sec* to write R1, M1, M2, M3, ID-Tag2 into the tag's memory, and *160.66 ms* to read the whole tag's responses.

3. DemoTag computing cost: In a successful run of the proposed protocol, the tag generates a random number, computes four messages, writes the messages in the

memory and updates two values. Table 6.5 demonstrates that the time cost of running the protocol on the DemoTag is around *10.66 ms*.

Table 6.5: Computing operations time cost (milliseconds)

	M_1	M_2	M_3	Update	R2	M_4	Write
Computing cost	1.63	1.63	1.56	0.12	0.11	3.81	1.8
Total	10.66						

The related works' protocols discussed in this chapter did not provide any performance measurements, thus we could not present a performance comparison between our protocol and other related work.

6.7 Summary

We proposed an improved version of a key distribution and recovery model in the RFID-enabled supply chain. We found that the Juels et al.'s model is not flexible as no party other than the manufacturer can update the secret key threshold. Moreover, the updated secret key model proposed by Cai et al. to improve the Juels et al.'s model is not resistant against location tracking, and desynchronisation attacks. Hence in this chapter, we proposed the following: Firstly, our scheme distributes a secret key securely in an RFID-enabled supply chain by using the secret-sharing approach. Secondly, it updates the secret key after each successful key recovery, and thus eliminates the threats associated with location tracking. Thirdly, the proposed protocol avoids replay attack by using fresh random numbers generated by the server and tags. Fourthly, to counter the disruptive effects of desynchronisation attacks, the protocol has a resynchronisation phase that is initiated by the reader whenever it suspects a desynchronisation with the tag. Fifthly, the proposed scheme permits the distributor to change the threshold based on the dispatched items. Finally, the proposed protocol attempts to accommodate the limited resources of the low-cost RFID tags in terms of storage and computational costs. The proposal in this chapter suits a conventional RFID deployment with the assumption of a secure server; however, many physical servers are being migrated to cloud solutions; so we investigate this in the next chapter.

Chapter 7

Secure Improved Cloud-Based RFID Authentication Protocol

Contents

In this chapter we study and enhance a recent proposed protocol regarding the security and privacy of RFID tag's data that resides in the cloud. Maintaining RFID tags' data in the cloud reduces the cost of deployment and storage, but raises more concerns regarding the security and privacy of RFID systems. Xie et al. proposed a cloud-based RFID protocol that mitigates such concerns. Improvements to this study will be the focus of this chapter. Our proposed protocol is analysed using CasperFDR and Scyther. Then, we conclude this chapter with an illustration of the implementation and performance measurements.

7.1 Introduction

As already discussed in Chapter 2, a traditional RFID system consists of tags, a reader and a server. The server is responsible for maintaining the tags' data and processing them for various purposes. The server plays an important role in authenticating the RFID tags' data, as the communication channel between the reader and tags is vulnerable to interception, modification, fabrication and replay attacks. There have been extensive studies attempting to achieve mutual authentication between the server and tags, such as in [66, 95, 135, 136, 122, 39, 128, 130, 117].

In [130], the authors divided the proposed RFID mutual authentication proposals into two approaches:

- Server-based RFID mutual authentication: In this approach, the tag and reader depend on the backend server for authentication. When the reader receives a message from the tag, it forwards the tag's message to the server to be authenticated. The server stores secret data related to the tags. The researchers in this case made the assumption that the server is secure, and their main focus was to secure the data transmission between tags and readers.
- Server-less RFID mutual authentication: This approach takes into account an offline authentication, where the reader authenticates the tag offline without the need to contact the server. This scheme is normally used for searching for a particular tag within a group of tags, and the reader does this process without querying the server. A server-less RFID scheme is based on tag authentication credentials previously stored in the reader. For instance, the reader contacts the Certification Authority (CA) during the initialisation phase to retrieve a list of legitimate tags data.

A new approach, where the RFID tags data can be stored in a remote server residing in the cloud, has gained increasing attention. According to the National Institute of Standards and Technology (NIST) [138], cloud computing can be defined as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Cloud computing provides a promising solution for handling and using data collected with powerful computing and massive storage abilities [131].

Confidentiality and privacy are generally regarded as two of the main concerns in cloud computing. This is largely due to the fact that customers outsource their data to cloud servers, thus losing direct control over their data, for example, customers' data may be altered, lost, or deleted [140]. Considering the cloud as the processor and storage for an RFID system, the tags' data should be protected against any malicious internal or external attacks, such as the attacks on the cloud described in [133, 134, 141]. Data privacy is also another critical concern, as customers' data is outsourced to a third party, meaning that a customer's sensitive data is out of his/her control, and might be disclosed to public or business competitors [139, 140]. To sum up, if cloud confidentiality is compromised, then privacy will also be violated [140].

To tackle the aforementioned concerns, the authors in [130] proposed a new scheme called "cloud-based RFID authentication", which provides the following features:

- It aims to address the confidentiality and privacy concerns regarding RFID tags' data in the cloud, where the cloud server is regarded as untrusted.
- The reader is connected to the tag through a wireless channel, where the communication between the reader and the cloud server is assumed to be secure.
- The manufacturer stores the tags' and readers' data in an encrypted hash table in the cloud server.

The authors in [130] claimed that their proposed protocol resists reader-to-tag impersonation attacks, as only the legitimate reader can compute the authentication messages. Furthermore, they also claimed that their protocol preserves tag's data anonymity by hashing the tag's data with a random number, which ensures confidentiality and freshness in all protocols' runs. Xie et al.'s protocol is discussed in detail in Section 7.2.

We examined the cloud-based RFID authentication protocol [130], and discovered the following:

1. An attacker is able to impersonate the reader without compromising the secret data shared with the tag, thus causing the tag to be updated with a wrong value, and permitting tracking of the tag's location.
2. By using CasperFDR, we found that tag data anonymity is not achieved, as the attacker can perform a man-in-the-middle attack and obtain the secret data at the end of the protocol session.

Hence, we propose a new protocol that uses some of the notions in [130] while introducing new approaches that improve the cloud-based RFID authentication protocol. Our improved cloud-based RFID authentication protocol is shown in Section 7.4.

Table 7.1: A summary of notation

Notation	Description
R	The identity of an RFID reader
T	The identity of an RFID tag
S	The number of authentication sessions between a reader and a tag, with bit length L
M	The last number of sessions between a reader and a tag
Nr	The random number generated by a reader
Nt	The random number generated by a tag
PRNG()	The Pseudo Random Number Generation (PRNG) function
H( )	The secure one-way hash function with output length L, that is, H(): $\{0,1\}^* \rightarrow \{0,1\}^L$
E( )	The encryption function using a symmetric algorithm with a reader secret key
D( )	The decryption function using a symmetric algorithm with a reader secret key
$\oplus$	An XoR operation
$\|$	A concatenation operation

7.2 Review of the Cloud-Based RFID Authentication Protocol

This section reviews the cloud-based RFID authentication protocol as proposed in [130]. The notation used in the cloud-based RFID authentication protocol is shown in Table 7.1.

The cloud-based RFID authentication protocol consists of two processes: the registration process, and the authentication process, which are summarised below:

- Registration process:

  The tag is encoded with three secret values: R, T and S. The manufacturer also stores unique initialised records, i.e., {H(R‖T‖S) and E(R‖T‖S)} to the cloud server. The authors assumed that the registration is secure and performed in a "closed" environment.

- Authentication process:

  The cloud-based RFID authentication protocol is shown in Fig. 7.1 and works as follows:

  1. Tag: The tag generates H(R‖T‖S) as an authentication request and sends it to the reader.

2. Reader: The reader sends the index message ($H(R\|T\|S)$) to the cloud, and retrieves $E(R\|T\|S)$ from the cloud's index table. Then, the reader decrypts $D(E(R\|T\|S))$, verifies R and obtains T and S.
3. Reader: The reader generates a random number (Nr) as a challenge to the tag, and sends (Nr) to the tag.
4. Tag: The tag calculates $H(R\|T\|Nr)$ as a response and generates a random number (Nt) as a challenge to the reader.
5. Reader: The reader verifies the tag's response, and if valid, the next step is started; otherwise, the protocol is terminated.
6. Reader: The reader tries to read the next record indexed by $H(R\|T\|(S+1))$ from the cloud server and checks the integrity. If there is a valid record, this implies that the tag has been desynchronized. The reader attempts to read the $S+2^{nd}$ record indexed by $H(R\|T\|(S+2))$, until it finds the last valid record.
7. Reader: The reader writes $E(R\|T\|M')$ with the index $H(R\|T\|M')$ into the cloud server, where $M'=M+1$.
8. Cloud: The cloud sends $H(R\|T\|M') \oplus H(E(R\|T\|M'))$ to the reader to confirm that the update process has been successful.
9. Reader: The reader sends the authentication messages $H(R\|T\|Nt) \oplus M'$, and $H(T\|R\|M')$ to the tag.
10. Tag: The tag calculates $H(R\|T\|Nt)$ XoRed with the received $H(R\|T\|Nt) \oplus M'$ to obtain M', and then it calculates and verifies $H(T\|R\|M')$. If successful, this implies that M' has not been modified by an attacker, and subsequently synchronisation is achieved again by updating $S=M'$ on the tag. The validity of M' also means that the reader is authenticated by the tag.

7.3 Security Analysis of the Cloud-Based RFID Authentication Protocol

In this section, we show the main weaknesses found in the cloud-based RFID authentication protocol.

7.3.1 Reader Impersonation Attack

In [130], the authors claim that their proposed protocol achieves a mutual authentication between the tag and the reader, as only the legitimate reader knows the data (R, T,

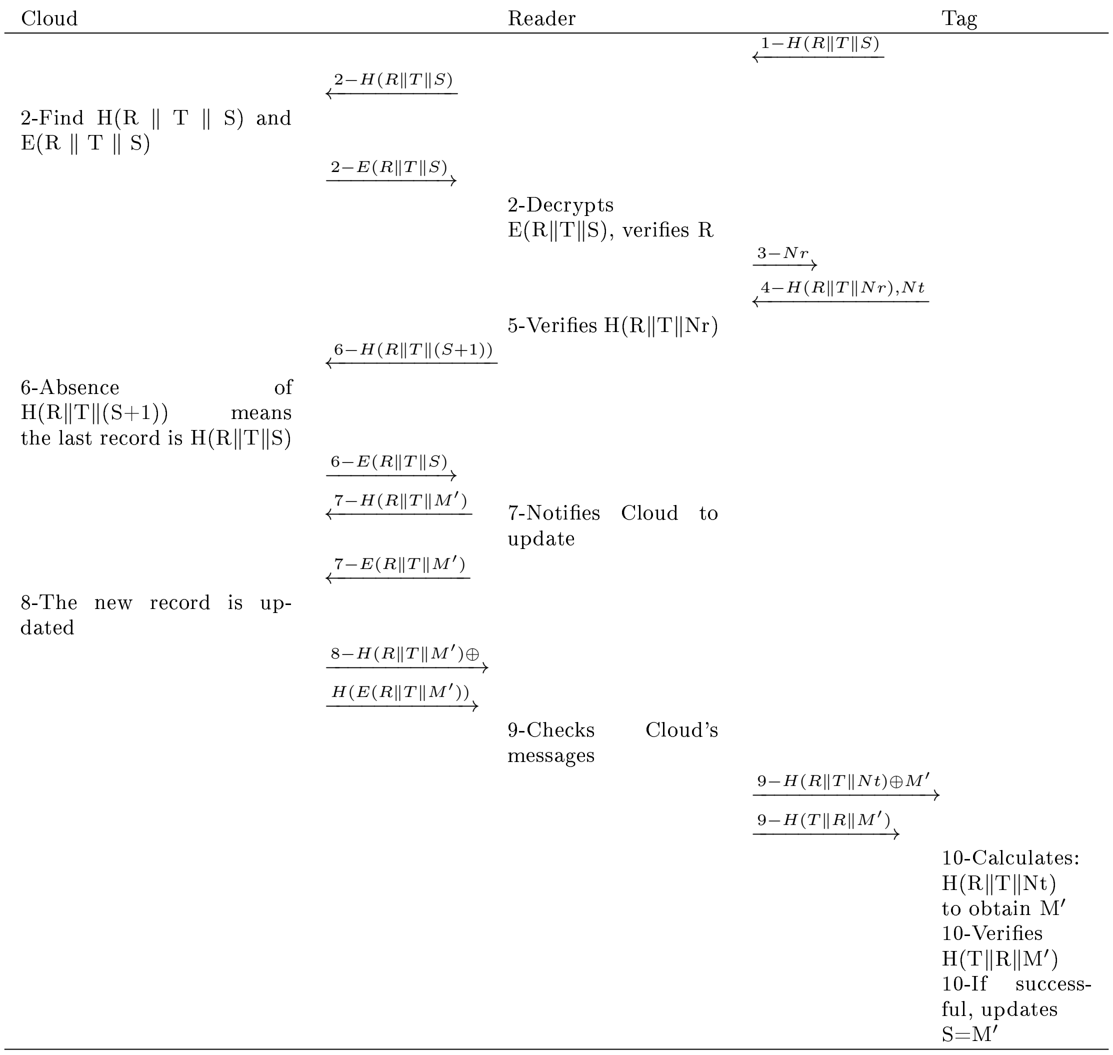

Cloud
Reader
Tag
1−H(R‖T‖S)
2−H(R‖T‖S)
2-Find H(R ‖ T ‖ S) and E(R ‖ T ‖ S)
2−E(R‖T‖S)
2-Decrypts E(R‖T‖S), verifies R
3−Nr
4−H(R‖T‖Nr),Nt
5-Verifies H(R‖T‖Nr)
6−H(R‖T‖(S+1))
6-Absence of H(R‖T‖(S+1)) means the last record is H(R‖T‖S)
6−E(R‖T‖S)
7−H(R‖T‖M′)
7-Notifies Cloud to update
7−E(R‖T‖M′)
8-The new record is updated
8−H(R‖T‖M′)⊕
H(E(R‖T‖M′))
9-Checks Cloud's messages
9−H(R‖T‖Nt)⊕M′
9−H(T‖R‖M′)
10-Calculates: H(R‖T‖Nt) to obtain M′
10-Verifies H(T‖R‖M′)
10-If successful, updates S=M′

Figure 7.1: Cloud-based RFID authentication protocol

M). However, we found that the attacker can impersonate a legitimate reader and be successfully authenticated by the tag without compromising the internal tag's data. The scenario for accomplishing this attack is as follows:

- Eavesdrops one session of the protocol, blocks the reader's message from reaching the tag and obtains all the exchanged messages including N and G, where N=H(R‖T‖Nt) ⊕ M, and G=H(T‖R‖M). As a result, the tag will not update the (S) value, hence the attacker will track the tag's location in subsequent sessions.
- The tag starts a new session with the reader, and sends H(R‖T‖S) to the reader.
- The reader sends a new random number (Nr′) to the tag.

- The tag generates a new random number (Nt') and sends $H(R\|T\|Nr')$ and Nt' to the reader.
- After the reader authenticates the tag, it asks the tag to update its value by sending

  $N'=H(R\|T\|Nt') \oplus M'$, and $G'=H(T\|R\|M')$, where $M'=S+1$.
- The attacker blocks N' and G', and calculates the following:
  1. Since $M' = M + 1$, the attacker changes the 2 least significant bits (LSB) of N to be equivalent with N' after addition, and then assigns the result to N''. In other words, if for example N is 111000 and N' is 101011, the attacker changes N to 111010 and assigns it to N''.
  2. $N'' \oplus N'= H(R\|T\|Nt) \oplus M'' \oplus H(R\|T\|Nt') \oplus M'$
  3. $(N'' + 1) \oplus N'= (H(R\|T\|Nt) \oplus M'' + 1) \oplus H(R\|T\|Nt') \oplus M'$. Note that $M''+1= M'$.
  4. $(N'' + 1) \oplus N'= H(R\|T\|Nt) \oplus H(R\|T\|Nt')$
  5. $(N'' + 1) \oplus N' \oplus N=(H(R\|T\|Nt) \oplus H(R\|T\|Nt')) \oplus H(R\|T\|Nt) \oplus M$
  6. $(N'' + 1) \oplus N' \oplus N=H(R\|T\|Nt') \oplus M$
- The attacker impersonates the reader and sends $H(R\|T\|Nt') \oplus M$ and the obtained G, i.e., $G=H(T\|R\|M)$ to the tag.
- The tag calculates $H(R\|T\|Nt')$ XoRed with the received $H(R\|T\|Nt') \oplus M$ to obtain M, then calculates and verifies $H(T\|R\|M)$; if successful, the tag authenticates the attacker not the legitimate reader and updates S with the wrong value.

7.3.2 Man-in-the-Middle Attack

In Xie et al.'s protocol, the value of (M), which represents the session number, should be kept secret. However, after analysing the protocol using CasperFDR, we found that the attacker can obtain the secret session number (M), thus tag data anonymity is not achieved.

We used CasperFDR to formally analyse the cloud-based RFID authentication protocol between the reader and the tag. CasperFDR was used to model communication and security protocols and verify the authentication and secrecy requirements of the protocol, which are the main goals of the Xie et al.'s protocol.

We prepared a CasperFDR script as shown in Appendix C.1. In the script, we assume that the reader knows about the tag's data. The communication between the reader and server is secure, therefore we did not check the protocol in this area. In the #Free variables Section, the reader (R) and tag (T) are defined as Agent; the random numbers (Nr) and (Nt) are defined as Nonce; and TID (tag identifier), RID (reader identifier), S, and M are defined as Data.

#Free variables
T, R : Agent
Nr: Nonce
Nt: nonce
TID, RID, S, M: Data
h: HashFunction
InverseKeys=(h,h)

As mentioned in Section 7.2, the main goals of the cloud-based RFID authentication protocol are authenticating the reader to the tag, and vice versa, and verifying that the data, such as R, T, S and M, remain secret between the reader and tag. These goals are shown in the script in the #Specification Section, where data secrecy is depicted as Secret, such as Secret(R, M, [T]), and the goal predicate authentication takes the form of Agreement:

#Specification
Agreement(T, R, [TID, RID])
Secret(R, M, [T])
Secret (T, S, [R])
Secret (T, TID, [R])
Secret(T, RID, [R])

After compiling the CasperFDR script and feeding the output to the verifier tool FDR, a man-in-the-middle attack was found. FDR states that the goal predicate Secret(R, M, [T]) was not achieved. The attack is illustrated below:

1. $T \rightarrow Mallory : H(R \parallel T \parallel S)$
2. $Mallory \rightarrow R: H(R \parallel T \parallel S)$
3. $R \rightarrow Mallory : Nr$
4. $Mallory \rightarrow T : Nr$
5. $T \rightarrow Mallory : H(R \| T \| Nr), Nt$

6. $Mallory \rightarrow R : H(R\|T\|Nr), Nr$

7. $R \rightarrow Mallory : N{=}H(R\|T\|Nr)\oplus M,\ G{=}H(T\|R\|M)$

The attacker performs a man-in-the-middle attack and eavesdrops on a session between the tag and the reader. The attacker impersonates the reader to obtain the tag's messages, and then impersonates the tag to send the tag's messages to the reader. CasperFDR shows that the attacker can replace the tag's random number (Nt) with the reader's generated random number (Nr), and at the end of the protocol run, the attacker can calculate $M{=}H(R\|T\|Nr)\oplus N$ and obtain M, which is assumed to be secret; thus, tag's data privacy is compromised.

As a result, in Section 7.4, we propose a new improved cloud-based RFID authentication protocol that prevents reader impersonation and man-in-the-middle attacks and takes into account other goals discussed in the next Section 7.4.1.

7.4 Improved Cloud-Based RFID Authentication Protocol

In this section, we explain the proposed protocol in detail.

7.4.1 Design Goals

The proposed protocol aims to protect the tag's data from being revealed to any entity except a legitimate reader. The proposed protocol should meet the following goals:

- Tag data anonymity: The RFID tag should support a mechanism for concealing the tag's data from any entity except the legitimate readers.
- Untraceability: If the data being sent from the tag to the reader is static or linked to data sent previously, the tag holder's location can be tracked. Therefore, the RFID tag's responses should be anonymous and unlinkable in order to prevent such an attack.
- Forward secrecy: The proposed protocol should ensure that if an attacker compromises the tag's memory, he/she will not be able to trace previous communication session(s) using previously known messages.
- Resistance to replay attacks: An adversary may eavesdrop on the communication between the reader and tag, re-use the data and send it repeatedly. Therefore, the generated messages should be fresh to the protocol session.

- Resistance to desynchronisation incidents: The proposed protocol should recover from de-synchronisation incidents, either when an attacker blocks the exchanged message(s), or when the messages are lost during transmission due to system malfunction or communication error.
- Resistance to impersonation attacks: An attacker can respond to a reader query and can claim that this response is coming from a legitimate tag, and this fabrication enables the attacker to masquerade as a legitimate tag. Similarly, an attacker may impersonate the legitimate reader and attempt to obtain access to the tag's data. Hence, to prevent such attacks, the tag's data should be protected during transmission.
- Mutual authentication: The protocol should provide a mutual entity authentication, where the communication should take place between valid tag, reader and cloud server. The readers should authenticate the cloud server before validating the tag's messages. At the end of the protocol run, the tag should receive a message from the reader that confirms the legitimacy of the reader.
- Performance: The tag's memory storage and the computing cost should be appropriate to the tag's limited computing capability, and the amount of data communicated should be minimised.

7.4.2 Assumptions

We present an improved cloud-based RFID authentication protocol, which operates under the following assumptions:

- The reader contacts the tag through a wireless channel that is susceptible to attacks.
- The communication channel between the reader and the cloud server is secure.
- There are multiple readers in the system, so a tag can be read in many different locations.
- This scheme only supports readers that are tamper-resistant, for example, they have a secure memory and a rigid access control mechanism.
- The tag's data are stored in a non-volatile memory, such as EEPROM or Flash memory, where they can be updated.
- All the operations in the tag are atomic.

- The cloud server is not trusted; it might be compromised to reveal the tag's data to intruders or competitors, and/or internal employees.

7.4.3 Protocol Design

The main protocol features are discussed below:

- Tags are capable of computing XoR, generating a pseudo-random number and calculating hash functions.
- The reader can compute XoR, generate a pseudo-random number, calculate hash functions and perform symmetric encryption and decryption.
- The proposed protocol uses random numbers in an attempt to prevent location tracking and replay attacks.
- After a successful authentication between the cloud server and tag, both parties update their values to be used in the next transaction.
- The cloud server does not store the tag's ID and the tag's secret key; it stores the hash of the tag's ID and the encryption of the tag's ID and tag's secret key to provide confidentiality and anonymity to the tag's data in the cloud.
- The cloud server stores both the old and the new tag's data in order to prevent desynchronization incidents.
- Each legitimate reader contains a master key used for a symmetric encryption.
- The reader does not store any data related to the tags.

7.4.4 Threat Model

We consider that the communication between the reader and the tag is vulnerable to both passive and active attackers. Accordingly, we used the Dolev-Yao adversary model [67], where the adversary has powerful resources to control the communication channel by performing the following:

- Eavesdropping
- Modifying messages
- Blocking messages from reaching targets
- Replaying previous messages

Table 7.2: Protocol notation

Notation	Description
ID_i	The i^{th} tag's ID, where $1 \leq i \leq n$
K_i	The i^{th} tag's secret key
MK	The master key shared by all the legitimate readers, and used for a symmetric encryption and decryption
$H(ID_{new})$	A hash of the updated ID, $H:\{0,1\}^* \leftarrow \{0,1\}^L$
$H(ID_{old})$	A hash of the old ID, $H:\{0,1\}^* \leftarrow \{0,1\}^L$
x	The value kept as either new or old to show whether the tag uses the old or new values of (ID_i and K_i)
R1	A pseudo-random number generated by the reader
R2	A pseudo-random number generated by the tag
$E_{MK}(M)$	A message M encrypted with a master key
$A \leftarrow B$	The value of A is updated to that of B
$\oplus$	An XoR operation
$\|\|$	A concatenation operation
j	The transaction number
i	The number of the tag in the system
n	The number of tags in the system

- Injecting new messages (forgery)
- Impersonating any entity

7.4.5 Notation

The notation used in the proposed protocol is presented in Table 7.2:

7.4.6 Protocol Description

The scheme consists of two phases: initialisation and authentication.

- Initialisation phase:

  We assume that the initialisation phase is carried out via a secure channel in a secure environment. The initialisation process is summarised below:

  1. For each tag the system operator manages, the system operator assigns a unique $H(ID_{new})$, which serves as an index, and $E_{MK}(ID_{new} \| K_{new})$ in the cloud server.
  2. Initially, $H(ID_{old})$, and $E_{MK}(ID_{old} \| K_{old})$ in the cloud server are set to null.
  3. The system operator assigns ID_i and K_i in the i^{th} tag.

4. The system operator assigns the master key in each reader the system manages.

- Authentication phase:

  The authentication process is shown in Table 7.2 and is described as follows:

  1. Reader: The reader starts the session by generating a random number R1 of L bits, where L is a security parameter, and sending it to the tag.
  2. Tag$_i$: The tag performs the following:
     - Generates R2 of L bits.
     - Computes the following messages:
       HID=H(ID$_i$ $\oplus$ R1 $\oplus$ R2), which serves as an index message
       M1=H(ID$_i$ $\|$ K$_i$ $\|$ R1 $\|$ R2), which serves as an authentication message
     - Sends HID, M1 and R2 to the reader.
  3. Reader: The reader sends HID, R1 and R2 to the cloud server.
  4. Cloud server: The cloud server performs the following:
     - For all the stored H(ID$_{new}$) and H(ID$_{old}$), it searches for H(ID$_x$ $\oplus$ R1 $\oplus$ R2) until there is a match. Marks x=new or old based on the matched H(ID$_i$).
     - Retrieves the associated data, i.e., E$_{MK}$(ID$_x$ $\|$ K$_x$).
     - Sends E$_{MK}$(ID$_x$ $\|$ K$_x$) and x to the reader.
  5. Reader: The reader performs the following:
     - Decrypts E$_{MK}$(ID$_x$ $\|$ K$_x$) using the master key, and obtains ID$_x$ and K$_x$.
     - Re-computes M1$'$=H(ID$_x$ $\|$ K$_x$ $\|$ R1 $\|$ R2). If there is a match, the reader authenticates the tag. Furthermore, the reader confirms that the data within the server's message are correct, and authenticates the cloud server.
       If M1$'$==M1 and x=new, this implies that the tag's data are synchronised with the server's data. The reader updates (ID$_{new}$ and K$_{new}$), to be used in the next transaction (j+1) by calculating the following:
       ID$_{new}^{j+1}$ $\leftarrow$ H(ID$_i^j$)
       K$_{new}^{j+1}$ $\leftarrow$ H(ID$_{new}^{j+1}$ $\oplus$ K$_i^j$)

Cloud Server		Reader		Tag$_i$
		1- Generates R1		
			$\xrightarrow{\text{R1}}$	

				2- Generates R2
				HID=H(ID$_i\oplus$R1$\oplus$R2)
				M1=H(ID$_i\|$K$_i\|$R1$\|$R2)

			$\xleftarrow{\text{HID,M1,R2}}$	
	$\xleftarrow{3-HID,R1,R2}$			
4- Searches for H(ID$_x\oplus$R1$\oplus$R2) Marks x=new or old Retrieves E$_{MK}$(ID$_x\|$K$_x$)				
	$\xrightarrow{E_{MK}(ID_x\|K_x),x}$			
		5- Decrypts E$_{MK}$(ID$_x\|$K$_x$) Re-computes: M1$'$=H(ID$_x\|$K$_x\|$ R1$\|$R2) If M1$'$==M1 and the x value is new, then updates: ID$^{j+1}_{new}$ $\leftarrow$ H(ID^j_i) K$^{j+1}_{new}$ $\leftarrow$ H(ID$^{j+1}_{new}$ $\oplus$ K^j_i) Calculates H(ID$^{j+1}_{new}$) and E$_{MK}$(ID$^{j+1}_{new}$ $\|$ K$^{j+1}_{new}$)		
	$\xleftarrow{\text{H(ID}^{j+1}_{new})}$			
	$\xleftarrow{\text{E}_{MK}\text{(ID}^{j+1}_{new} \| \text{K}^{j+1}_{new})}$			
6- Writes: H(ID$_{new}$)$\leftarrow$H(ID$^{j+1}_{new}$) E$_{MK}$(ID$_{new}\|$K$_{new}$)$\leftarrow$E$_{MK}$(ID$^{j+1}_{new}\|$K$^{j+1}_{new}$) H(ID$_{old}$)$\leftarrow$H(ID$^j_{new}$) E$_{MK}$(ID$_{old}\|$K$_{old}$)$\leftarrow$E$_{MK}$(ID$^j_{new}\|$K$^j_{new}$)				
	$\xrightarrow{\text{OK}}$			
		7- Calculates: M2=H(K$^{j+1}_{new}\|$R1$\|$R2)		
			$\xrightarrow{\text{M2}}$	
				8- Computes: ID$^{j+1}_i$$\leftarrow$H(ID$^j_i$) K^{j+1}$\leftarrow$H(ID$^{j+1}\oplus$K^j) Re-calculates M2$'$=(H(K$^{j+1}|$R1$|$R2), then updates ID and K

Figure 7.2: Improved cloud-based RFID mutual authentication protocol

- Calculates H(ID$^{j+1}_{new}$), and encrypts the new values, i.e., E$_{MK}$(ID$^{j+1}_{new}$ $\|$ K$^{j+1}_{new}$).
- Notifies the server to update its values by sending:

H(ID_{new}^{j+1}), and E_{MK}(ID_{new}^{j+1} ∥ K_{new}^{j+1})

6. Cloud server: The cloud server performs the following:
   - Writes the following data in its database:
     $\text{H}(\text{ID}_{new}) \leftarrow \text{H}(\text{ID}_{new}^{j+1})$
     $\text{E}_{MK}(\text{ID}_{new}\|\text{K}_{new}) \leftarrow \text{E}_{MK}(\text{ID}_{new}^{j+1}\|\text{K}_{new}^{j+1})$
     $\text{H}(\text{ID}_{old}) \leftarrow \text{H}(\text{ID}_{new}^{j})$
     $\text{E}_{MK}(\text{ID}_{old}\|\text{K}_{old}) \leftarrow \text{E}_{MK}(\text{ID}_{new}^{j}\|\text{K}_{new}^{j})$
   - Sends an OK message to notify the reader that the update process has been successful.

7. Reader: The reader performs the following:
   - If the reader received the OK message from the cloud server, the reader notifies the tag to update its data such as (ID_{new}, K_{new}) by calculating M2=H(K_{new}^{j+1} ∥ R1 ∥ R2) using the updated tag's secret key.
   - Sends M2 to the tag.

8. Tag_i: The tag performs the following:
   - Computes $\text{ID}_i^{j+1} \leftarrow \text{H}(\text{ID}_i^{j})$ and $\text{K}_i^{j+1} \leftarrow \text{H}(\text{ID}_i^{j+1} \oplus \text{K}_i^{j})$.
   - Re-calculates M2′=(H(K_i^{j+1} ∥ R1 ∥ R2), and if it is equal to the received value of M2, then it authenticates the reader and updates ID_i and K_i to:
     $\text{ID}_i^{j+1} \leftarrow \text{H}(\text{ID}_i^{j})$
     $\text{K}_i^{j+1} \leftarrow \text{H}(\text{ID}_i^{j+1} \oplus \text{K}_i^{j})$

If M1′==M1 and x=old, the reader still authenticates the tag, but this implies that the tag's data has been desynchronised, thus the reader does not update the current values of the ith tag (ID_i and K_i). It sends no update to the server, and sends M2=H(K_{new}^{j} ∥ R1 ∥ R2) to the tag using the current value of the tag's secret key. Then, the tag re-computes M2 using the current values. If there is a match with the received M2, the tag authenticates the reader and updates its data, as shown in the previous step.

If there is no match with the received M2 using the current or updated values of ID_i and K_i, then the tag will not authenticate the reader, and will not update its data.

7.5 Protocol Analysis

In this section, we present the analysis of the proposed protocol in terms of informal, and formal analysis using CasperFDR and Scyther.

7.5.1 Informal Protocol Analysis

Our proposed protocol meets the following goals:

- Mutual authentication: If the reader successfully calculates the tag's responses M1, it authenticates the tag, as only the legitimate tag knows the values of (ID_i and K_i) and thus can calculate such responses. Similarly, if the tag calculates M2 and it finds a match with the received M2, it confirms that the reader has successfully recovered the values of (ID_i and K_i). Furthermore, the reader decrypts the server's message ($E_{MK}(ID_x \parallel K_x)$), and if the tag's message M1 is authenticated, this implies that the cloud server is sending legitimate data within the message, and hence the reader authenticates the server.

- Tag data anonymity: The tag stores two secret values, each of which is 128-bit length: ID_i and K_i, which are not revealed to any entity except the legitimate readers. The tag's data are not sent in clear in messages HID, M1 and M2, as they are protected using the hash function. Therefore, only the legitimate entity can extract these values. Furthermore, if the cloud server is a malicious entity, this will not affect the tag's data privacy, as the cloud server stores the hash of the tag's ID and the encrypted tag's data; and without the master key, the cloud server cannot disclose the tag's data. Finally, the size of the data is 128-bit length, which means that the attacker needs to make 2^{128} attempts to recover the secret data.

- Tag location privacy (untraceability): In the proposed protocol, the tag's responses are changed in each session using the updated tag's values and fresh random numbers (R1 and R2), thus the attacker will obtain new responses every time he eavesdrops on a session. Even if the the tag does not update its values, the responses will not be static due to the use of fresh random numbers R1 and R2. However, the cloud server can track the location of the tag's holder.

- Forward secrecy: The values of ID_i and K_i are updated after each run in order to prevent forward secrecy invasion, using a hash function that is irreversible. If an adversary compromises the tag's memory, he/she will not be able to trace the previous communications of the tag as the obtained messages involve the use of previous secret values ID_i and K_i, which are not stored in the tag. The stored updated values are used in the calculation of the next session, and cannot be irreversible as a result of using a hash function.

- Resistance to replay attacks: The proposed protocol uses a challenge-response scheme. In messages HID and M1, the tag includes the reader's and tag's fresh random numbers as a challenge. The reader must therefore include R1 and R2 in its response (M2). Therefore, only legitimate parties (reader and tag) can send valid answers, since random numbers are sent with secret values only known to the reader and tag.

- Resistance to desynchronisation incidents: In the proposed protocol, desynchronisation incidents are avoided by storing the previous values of the tag's data in the cloud server, hence achieving synchronisation. If the tag's data is being desynchronised, the cloud server keeps the tag's data fixed. For instance, if the attacker blocks M2 more than once in consecutive sessions, the tag and cloud will not update the tag's data. In the next session, the reader contacts the desynchronised tag, and sends the tag's HID message to the cloud server, then the cloud server finds a match with the tag's old data, and sends $E_{MK}(ID_{old} \parallel K_{old})$, as HID matches $H(ID_{old})$; thus synchronisation is still achieved.

- Resistance to tag and reader impersonation attack: To impersonate the tag, the attacker must be able to compute a valid response (HID, M1) to a reader query. However, it is hard to compute such responses without knowledge of ID_i and K_i, each of which is 128-bit length, and it will take up to 2^{128} attempts to guess each value. Similarly, the attacker needs to be in possession of ID_i and K_i to impersonate the legitimate reader and send M2.

- Compromising the reader: The only risk that the system may encounter is compromising the reader, allowing the attacker to access the master key. However, in the Assumption Section 7.4.2, we assumed that the proposed protocol only supports readers that are tamper-resistant; for example, they have a secure memory and a rigid access control mechanism.

Table 7.3 shows how our proposed protocol provides more security and privacy features than the cloud-based RFID authentication protocol. Based on the discovered weaknesses, we found that the cloud-based RFID authentication protocol is vulnerable to reader impersonation attacks; hence mutual authentication is not achieved. Moreover, we used CasperFDR to analyse the cloud-based RFID authentication protocol, and it showed that an attacker can discover the secret value (M); hence the tag data anonymity is compromised. Finally, if the tag does not update the value of (S), it will reply with the same answer, and hence allows location tracking.

Table 7.3: Comparison between the cloud-based RFID authentication protocol and our proposed protocol

	Cloud-based protocol [130]	Our proposed protocol 7.4
Tag data anonymity	×	√
Tag location privacy	×	√*
Resistance to replay attack	√	√
Resistance to desynchronisation	√	√
Resistance to tag impersonation	√	√
Resistance to reader impersonation	×	√
Mutual authentication	×	√

*: The cloud server can track the tag's location ×: Means does not provide protection √: Means resists such an attack

7.5.2 Formal Protocol Analysis

To formally analyse the proposed protocol and confirm that secrecy and authenticity between the reader and tag are achieved, we used CasperFDR [104] and Scyther [75] tools.

CasperFDR Analysis of the Proposed Protocol

We prepared a CasperFDR script as shown in Appendix C.2. In the script, we assume that the reader knows about the tag's data. The communication between the reader and server is secure, therefore we did not check the protocol in this area. In the #Free variables Section, the tag (T) and reader (R) are defined as Agent; the random numbers R1 and R2 are defined as Nonce; and ID (tag identifier), K (tag key) are defined as Data.

#Free variables
T, R : Agent
R1: Nonce
R2: nonce
ID,K : Data
h: HashFunction
InverseKeys=(h,h)

As mentioned in Section 7.4.1, the main goals of our protocol are authenticating the reader to the tag, and vice versa, as well as verifying that the data, such as ID and K, are secure. These goals are shown in the script in the #Specification Section as shown below:

#Specification
Agreement(T, R, [ID, K])
Secret (T, ID, [R])
Secret (T, K, [R])

In addition, in the #Intruder information Section, the intruder is defined as Mallory, who can take full control of the session; he/she can impersonate any entity in the protocol, read messages transmitted in the network, intercept, analyse and/or modify messages.

After compiling the CasperFDR script and feeding the output to FDR, CasperFDR found no feasible attacks, which means that mutual authentication is achieved successfully between the reader and the tag, and the tag's data are protected and transferred securely.

Scyther Analysis of the Proposed Protocol

Scyther performs a formal analysis of security protocols using a Dolev-Yao model [67] for an unbounded number of instances. It is mainly used to verify the authenticity of the messages exchanged between entities such as in the proposed protocol considered here.

We conducted the analysis of our protocol with respect to three goals: secret, aliveness and agreement. The script is shown in Appendix C.3. Two roles are defined, namely a reader (R) and a tag (Ti). The random numbers R1 and R2 are defined as Nonce; and IDi (tag identifier), Ki (tag key) are defined as Data.

Both roles the reader and tag share the secret goal over the two secret values IDi and Ki as follows:

claim_ R1(R, Secret, IDi);
claim_ R2(R, Secret, Ki);
claim_ Ti1(Ti, Secret, IDi);
claim_ Ti2(Ti, Secret, Ki);

Also both roles claim to be alive and shre the agreement goal as follows:

claim_ R3(R, Alive);
claim_ R4(R, Niagree);
claim_ Ti3(Ti, Alive);

claim_ Ti4(Ti, Niagree);
claim_ Ti5(Ti, Nisynch);

After compiling the Scyther script, it found no feasible attacks.

7.6 Protocol Implementation and Performance Measurement

In this section, we present the implementation process and the performance measurements taken from the DemoTag.

7.6.1 Implementation Process

The tag is provided with two 128-bit secret values: ID_i and K_i, which are stored in the tag's EEPROM. We did not implement the server side, so we assumed in the implementation that the reader already knows ID_i and K_i.

The reader starts by generating a 128-bit random number (R_1) using the *rngCsp* method. Then, it sends R1 to the tag by sending 8 *WriteTagData_ EPC_ C1G2* commands. When the tag receives the reader's random number, it generates a random number R_2 (128-bit) using the built-in PRNG, computes two messages, namely HID and M1, each of which is 128-bit length. Subsequently, the tag writes the messages (R2, HID, M1) in its memory using the *syscall_ writeWord* command, ready to be read on demand. The reader later sends three *ReadTagData_ EPC_ C1G2* commands to read R2, HID, and M1, and re-computes M1. If there is a match with the received M1, it temporarily updates ID_i and K_i, and uses them in the calculation of M2 (128 bits). Finally, the reader sends 8 *WriteTagData_ EPC_ C1G2* commands to the tag representing M2. The tag updates ID_i and K_i and re-computes M2 to authenticate the reader. If successful, the tag changes the values of ID_i and K_i with the updated ones.

7.6.2 Performance Measurement

The performance measurements are as follows:

1. DemoTag memory cost: In the proposed protocol, the memory cost is:
    - 348 bytes used from 4 KB EEPROM memory for storing the tag's data, messages (responses), and random numbers.
    - 32.7 KB used from 128 KB Flash memory to store the tag's firmware.

Table 7.4: Data exchange time cost

	Tag	Reader
Read	-	1.23 sec
Write	1.3 ms	990 ms

2. Data exchange time cost is shown in Table 7.4: For the tag to write R2, HID and M1 into its memory to be read by the reader, the average timer counter after running the protocol 100 times is 20802.7, so, based on equation 3.1, the time cost is:

$$20802.7 * 0.0000000625 = 0.0013sec \equiv 1.3ms \tag{7.1}$$

   Regarding the reader, we found that the reader needs *990 ms* to write the random number R1 and M2 into the tag's memory, and *1.2 sec* to read the whole tag's response.

3. DemoTag computing cost: In a successful run of the proposed protocol, the tag generates R2, computes three messages including data update, and writes messages in its memory. Table 7.5 demonstrates that the time cost of running the protocol on the DemoTag is around *9.34 ms*, which means that the tag can respond to the reader's query in less than a second, and this demonstrates the relative efficiency of the proposed protocol.

Table 7.5: Computing operations time cost (milliseconds)

	R2	HID	M_1	M_2 + Update	Write
Computing cost	0.11	1.63	1.63	4.67	1.3
Total	9.34				

7.7 Summary

In this chapter, we examined the cloud-based RFID authentication protocol, and found that the protocol is prone to reader impersonation and man-in-the-middle attacks. Therefore, we proposed an improved cloud-based RFID authentication protocol that avoids the cloud-based RFID authentication protocol security issues. The proposed protocol has been analysed informally, and we showed that it is more immune to reader impersonation attacks and can resist replay, desynchronisation, and tag impersonation attacks. Also, we illustrated that tag's data anonymity is preserved, and hence the cloud server and attackers cannot obtain the tag's data. In addition, the communication session between the reader and the tag was formally analysed using CasperFDR and Scyther, and found no feasible attacks. Finally, we showed that the proposed protocol imposes relatively low memory storage and computing costs on the RFID tags. So far we have only considered security protocols where there is only one tag being read at a time; however, we mentioned at the outset that a group of items may be presented simultaneously for reading. Proving that a group of legitimate tags are present and that there is no way for a fake tag to join the group is considered in the next chapter.

Chapter 8

Two Rounds RFID Grouping-Proof

Contents

In this chapter, we focus on a particular RFID application called a grouping-proof, where an entity such as a reader generates a proof of simultaneous presence of two or more tagged items. We begin by discussing related studies and their weaknesses in terms of security and performance. Then, we present a two rounds grouping-proof protocol that provides immunity against active attacks on RFID protocols and improves performance. An informal analysis is provided for the proposed protocol, followed by a formal analysis using CasperFDR and Scyther. Finally, we describe the implementation of our protocol, and then illustrate the performance measurements.

8.1 Introduction

In 2004, Juels introduced a new RFID application called a "yoking-proof" ("yoke" means joining things together) [106]. Yoking-proof can be defined as proof of the simultaneous presence of a pair of RFID tagged items in the broadcast range of an RFID reader within a short time period. Since its introduction, the yoking-proof has evolved to include *multiple tags* and is now known as the "grouping-proof". In an RFID grouping-proof, the generated record of the simultaneous presence of multiple tags is verified by the server that processes the system tags' data.

There are two modes for verifying the existence of RFID tags; online and offline. In the online mode, the server that verifies the proof is running during the protocol execution, while in the offline mode the server is not present during the scanning process. Where the server is online, the solution is straightforward as each tag can authenticate itself directly to the server. On the other hand, if the server is offline the solution is challenging, as a fake tag may participate in the proof that is verified later, and the tags' responses should be completed within a specified time-window [19].

A grouping-proof can be used in many systems including [142]:

- Hospitals: proving that a certain patient has been given his/her medications at the same time.
- Manufacturing: proving that devices have been sold with their attachments.
- Access control: establishing that a group of people with legitimate RFID tokens were present.
- Supply chains: proving that tagged products are shipped together in groups.
- Airport: associating an electronic passport with an owner or with any of his/her luggage.

In such systems, the server, which can be an auditor or a verifier, might not participate in the scanning process. Since the server is offline during the scanning process, unrelated tags might take part in the session. To solve this issue, a proof needs to be generated, and at a later time, a server verifies the simultaneous existence of the related and legitimate tags along with other goals discussed in Section 8.3.1.

In a typical grouping-proof scenario where there are n RFID tags in the group [143], the ith tag (T$_i$), i.e., $1 \leq i \leq n$, sends a message (M_i) to the reader, then the reader transfers M_i to the next tag (T$_{i+1}$) in the same group and waits for its response (M$_{i+1}$). The reader repeats this operation n times. Finally, when the reader receives n

Table 8.1: A summary of notation

Notation	Description
S	Server
R	Reader
T$_i$	The ith tag in the group, where $1 \leq$ i $\leq$ n
K$_x$	Entity x secret key
ID$_x$	The identity of entity x
MAC$_{K_i}$(M)	Keyed message authentication code for a message M
f_{K_i}	A keyed pseudo-random function
r$_x$	A random number generated by entity x
TS	Timestamp
sn	Session number
n	The number of tags in the group

responses, it creates the proof and sends it to the server to be verified later. Hence, the number of rounds is proportional to the number of tags in the group.

In this chapter, we aim to propose a grouping-proof protocol that is secure against active attacks on RFID protocols, and to improve the protocol's performance by providing the following features:

- Fewer rounds, to reduce time delay.
- Concurrency, where each tag does not need to wait for the T_{i-1} message to respond. Hence, dependency between tags is omitted. The tags only wait for the reader's message before responding.
- Reading order independence, so the tags can be verified by the server in any order. This approach reduces failure rates [18].

8.2 Related Work

This section reviews the literature regarding *RFID yoking-proof* and *RFID grouping-proof*. For the rest of this section, we will use the notation summarised in Table 8.1.

8.2.1 Yoking-proof RFID Protocols

Starting with Juels' yoking-proof protocol [106], each ith tag T$_i$ is embedded with an ID$_i$ and a secret key K$_i$ shared with the server. The tags are capable to generate random number (r$_i$) and compute an MAC function. Juels' protocol is depicted in Fig. 8.1. When the reader generates the yoking-proof P$_{1-2}$ for T$_1$ and T$_2$ for example, it sends it later to the server to verify the simultaneous existence of T$_1$ and T$_2$.

T_1		Reader		T_2
	$\xleftarrow{Query}$			
a= (ID_1, r_1)				
	$\xrightarrow{a}$			
			$\xrightarrow{Query, r_1}$	
				m_2=$MAC_{K_2}[r_1]$
			$\xleftarrow{ID_2, r_2, m_2}$	
	$\xleftarrow{r_2}$			
m_1=$MAC_{K_1}[r_2]$				
	$\xrightarrow{m_1}$			
		P_{1-2}=$(ID_1, ID_2, r_1, r_2, m_1, m_2)$		

Figure 8.1: Juels' yoking-proof protocol

In 2005, Saito et al. [144] discovered a replay attack against Juels' protocol [106]. They pointed out that the server does not provide any randomness to ensure the freshness of the reader's generated proof. Therefore, they proposed their own protocol that combated replay attack using timestamps. The server verifies the timestamp that is included in both the tags' messages and the reader's generated proof.

However, Piramuthu [145] showed that Saito et al. protocol [144] is still vulnerable to replay attack as the attacker can predict a timestamp for some future point in time, and then replay it to interrogate another tag. Accordingly, Piramuthu proposed the use of random numbers instead of timestamps to provide freshness to the tags' messages. The server generates a random number (r_S) and sends it to the reader. r_S is used as a seed to generate the tags' random numbers. Piramuthu protocol is depicted in Fig. 8.2. Piramuthu claimed that no secret data is transmitted in transit, and the data transmitted is refreshed in every protocol run, hence his protocol achieves user privacy and location privacy.

T_1		Server $r_S \downarrow$ Reader		T_2
	$\xleftarrow{Query, r_S}$			
a= (ID_1, r_1)				
	$\xrightarrow{a}$			
			$\xrightarrow{Query, r_S, r_1}$	
				m_2=$MAC_{K_2}[r_S, r_1]$
			$\xleftarrow{ID_2, r_2, m_2}$	
	$\xleftarrow{m_2}$			
m_1=$MAC_{K_1}[m_2, r_1]$				
	$\xrightarrow{m_1}$			
		P_{1-2}=$(r_1, r_2, r_S, m_1, m_2)$		

Figure 8.2: Piramuthu's yoking-proof protocol

According to Peris-Lopez [124], in Piramuthu's protocol, T_2's random number (r_2) is not authenticated by T_1, and this leads to a multi-session attack. This attack is shown in Fig. 8.3. Firstly, the adversary eavesdrops on a normal communication session between T_1 and T_2 to obtain the random numbers and m_2 to be used later. Then, the adversary impersonates the reader and interrogates tag T_x to generate a proof that shows that T_2 and T_x were simultaneously scanned, which is not valid.

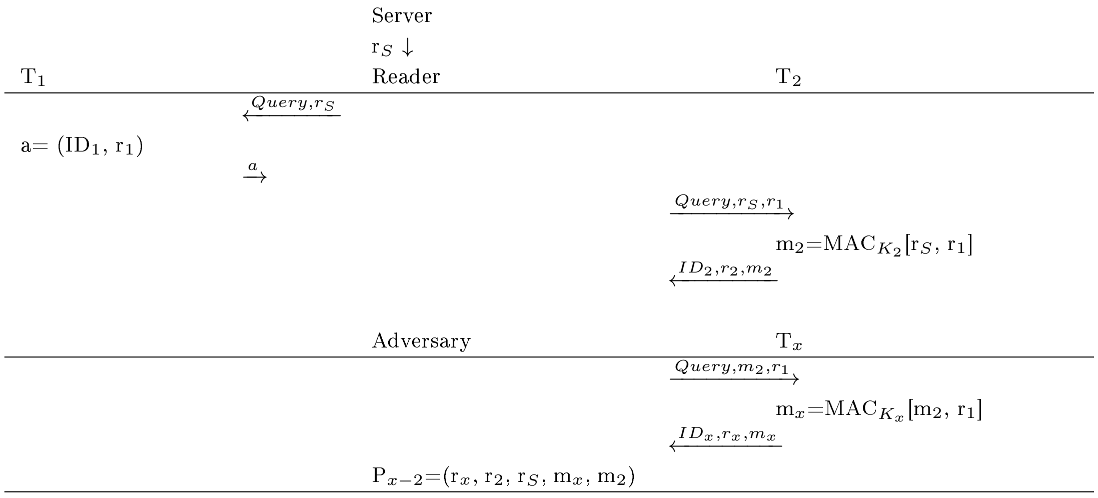

Server
$r_S \downarrow$
Reader
T_1
T_2
$\xleftarrow{Query, r_S}$
a= (ID_1, r_1)
$\xrightarrow{a}$
$\xrightarrow{Query, r_S, r_1}$
m_2=$MAC_{K_2}[r_S, r_1]$
$\xleftarrow{ID_2, r_2, m_2}$
Adversary
T_x
$\xrightarrow{Query, m_2, r_1}$
m_x=$MAC_{K_x}[m_2, r_1]$
$\xleftarrow{ID_x, r_x, m_x}$
P_{x-2}=$(r_x, r_2, r_S, m_x, m_2)$

Figure 8.3: A multi-session attack on Piramuthu's protocol

Chien et al. [147] proposed a yoking-proof that is based on a tree structure, where each tag is a kin to a leaf on a tree and the tag's identity is the path from the root to the leaf, to reduce the computational cost of authenticating each tag by the verifier from O(N) to O(1). Nevertheless, Peris- Lopez et al. [146], found that the Chien et al.'s protocol was vulnerable to replay attacks.

8.2.2 Grouping-Proof RFID Protocols

The notion of grouping-proof was introduced by Saito et al. [144]. They proposed using a "pallet tag". The pallet tag (PT) can be a large metal plate, on which the products can be placed. They assumed that the pallet tag has more computing resources than normal tags and shares a secret key (K_{PT}) with the server. The idea behind Saito et al.'s protocol is that the pallet tag gathers the tags' messages via the reader and generates a proof that is verified by the server later. The reader acts as middleware between the tags, pallet tag and server. The protocol is shown in Fig. 8.4.

T_i		Server TS ↓ Reader		PT
	$\xleftarrow{TS}$		$\xrightarrow{TS}$	
$m_1=MAC_{K_1}[TS]$				
	$\xrightarrow{m_1}$		$\xrightarrow{m_1}$	
	$\xrightarrow{m_2}$		$\xrightarrow{m_2}$	
	...		...	
	$\xrightarrow{m_n}$		$\xrightarrow{m_n}$	
				$C_P=E_{K_{PT}}[TS, m_1, m_2, \dots , m_n]$
			$\xleftarrow{C_P}$	
		$P_n=(TS,C_P)$		

Figure 8.4: Saito et al.'s grouping-proof protocol

Bolotnyy et al. [148] enhanced Juels' work and extended the yoking-proof to include groups of more than two tags. This scheme was introduced to preserve privacy of the tags by transmitting the output of a keyed hash function instead of transmitting the tags' static identifiers. Each tag generates a fresh random number and computes $a_i = f_{K_i} [r_i, a_{i-1}]$, where ($a_{i-1}$) represents T_{i-1}'s message. This process continues until the reader reaches the last tag T_n. The reader forwards T_n messages (a_n) to the first tag to link the tags' chain, and create proof, i.e., $m = f_{K_1}[a_1, a_n]$. Finally, the reader sends the server message P, i.e., $P_n = (r_1, r_2, \dots , r_n, m)$ for verification purposes.

Peris-Lopez et al. [146] recommended some guidelines for designing a secure RFID grouping-proof protocol, which are discussed in detail in Section 8.3.1. Accordingly, they proposed an RFID grouping-proof protocol shown in Fig. 8.5. The tags within a group share a group identifier ID_{group} and a group secret key K_{group} to prevent unrelated tags from participating. The tags also store an identifier (ID_i) and a secret key (K_i) to be authenticated by the server. The verifier computes encrypted timestamps $TS=f_{K_S}(Timestamp)$, where K_S, is the server's secret key. The generated timestamps will be valid for a limited time-window. However, Peris-Lopez et al.'s protocol is suitable only if there are only two tags in the system; if there are thousands of tags in the system, their protocol is impractical as each tag has to wait for the predecessor's tag output to include it in its response, producing a large time delay.

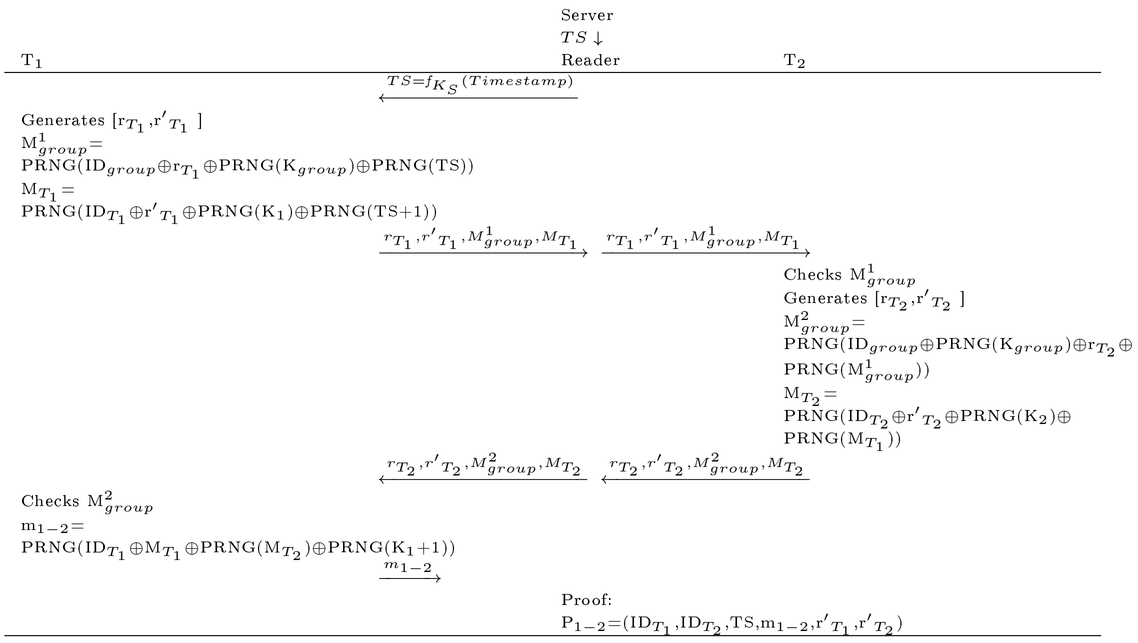

Server
$TS \downarrow$
Reader
T_1
T_2
$TS=f_{K_S}(Timestamp)$
Generates [r_{T_1},r'_{T_1}]
M^1_{group}=
PRNG($ID_{group} \oplus r_{T_1} \oplus$PRNG($K_{group}$)$\oplus$PRNG(TS))
M_{T_1}=
PRNG($ID_{T_1} \oplus r'_{T_1} \oplus$PRNG($K_1$)$\oplus$PRNG(TS+1))
$r_{T_1}, r'_{T_1}, M^1_{group}, M_{T_1}$
$r_{T_1}, r'_{T_1}, M^1_{group}, M_{T_1}$
Checks M^1_{group}
Generates [r_{T_2},r'_{T_2}]
M^2_{group}=
PRNG($ID_{group} \oplus$PRNG(K_{group})$\oplus r_{T_2} \oplus$
PRNG(M^1_{group}))
M_{T_2}=
PRNG($ID_{T_2} \oplus r'_{T_2} \oplus$PRNG($K_2$)$\oplus$
PRNG(M_{T_1}))
$r_{T_2}, r'_{T_2}, M^2_{group}, M_{T_2}$
$r_{T_2}, r'_{T_2}, M^2_{group}, M_{T_2}$
Checks M^2_{group}
m_{1-2}=
PRNG($ID_{T_1} \oplus M_{T_1} \oplus$PRNG($M_{T_2}$)$\oplus$PRNG($K_1$+1))
m_{1-2}
Proof:
P_{1-2}=(ID_{T_1},ID_{T_2},TS,m_{1-2},r'_{T_1},r'_{T_2})

Figure 8.5: Peris-Lopez et al.'s grouping-proof protocol

Ma et al. [116] extended Peris-Lopez et al.'s grouping-proof protocol to include more than two tags. According to Sundaresan et al. [142], Ma et al.'s protocol is prone to forward secrecy invasion, and it relies on an active clock tag, which can be compromised as it participates significantly in the protocol.

All the previous protocols are *sequential*, where a tag must wait for a response from the previous tag before it can proceed.

In contrast, proposals moved to designing a *concurrent* grouping-proof, where each tag sends its authenticator message independently to the reader that generates the proof. Burmester et al. [149] proposed a grouping-proof protocol, where the main aim is that all the tags within the group share a group ID (ID_{group}) and a secret key (K_{group}), so the tags within the group will recognise each other, and can be linked by the reader. Burmester et al.'s protocol is depicted in Fig. 8.6. There are three phases in this protocol: the first phase, where the reader challenges the tags with the server random number, and the tags respond with their group ID. In the second phase, the reader links the tags based on received the group ID. In the third phase, the tags prove membership in their group. The first and second phase are concurrent, but the last phase is sequential, where each tag has to wait for the predecessor tag's message to respond.

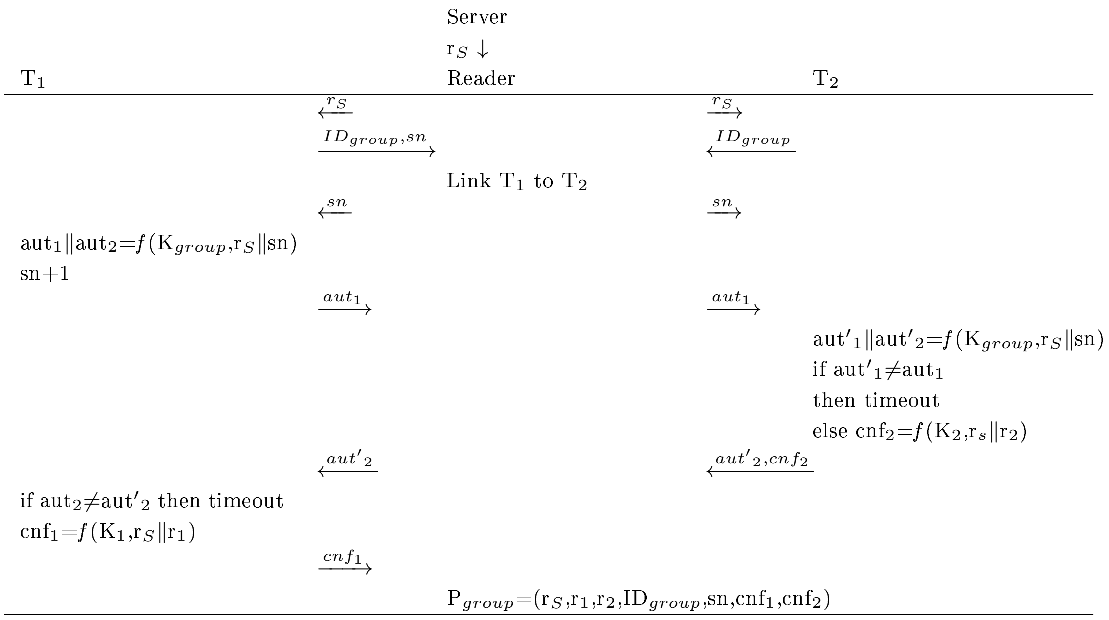


Figure 8.6: Burmester et al.'s grouping-proof protocol

Peris-Lopez et al. [146], discovered that Burmester et al.'s protocol [149] is still vulnerable to multi-proof replay attack as shown in Fig. 8.7. To achieve this attack, the adversary eavesdrops on the communication channel between T_1 and T_2 first. Then, the adversary impersonates T_1 and the captured messages are replayed to T_x to build a counterfeit proof that deceives the server into thinking that T_1 and T_x have been scanned simultaneously.

Another scheme was proposed by Lien et al. [18], who introduced the idea of a "reading order independence" system, which can be defined as verification of the proof regardless of the order in which the tags were scanned. This approach reduces failure rates. In other words, if the server assumes that the tags should be scanned in a specific order, and the received proof contains tags' messages in random order, the server proof will not match the received proof and the grouping-proof will fail. Lien et al. protocol's uses a pallet tag (PT) to generate the proof. Reading order independence is achieved using the XoR operator, which is commutative. When the pallet tag receives n responses, it generates m_{PT}=$MAC_{K_{PT}}[r_S, r_{PT}, m_1 \oplus m_2 ... \oplus m_n]$, where m_i=MAC_{K_i} $[m_{PT}, r_i]$, and m_{PT}=$MAC_{K_{PT}}[r_i, r_{PT}]$, and sends it to the reader. The reader generates proof, i.e., P=(r_S, r_1, r_2, ... , r_n, m_{PT}), and sends it to the server.

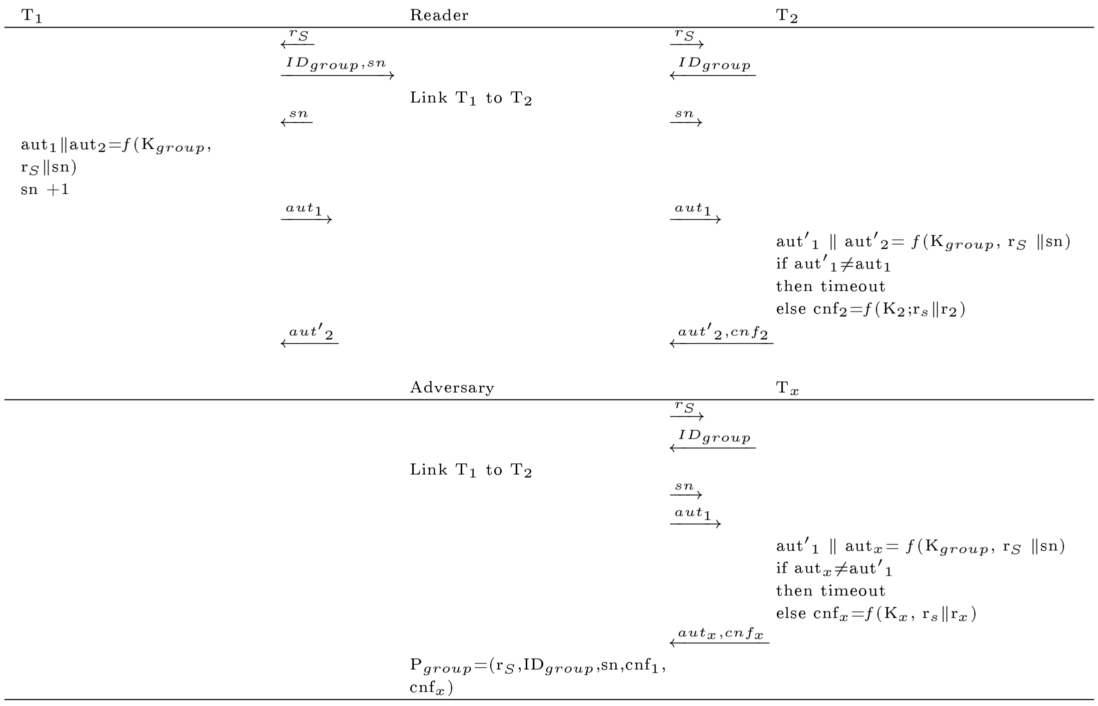

T$_1$
Reader
T$_2$
r_S
r_S
ID_{group},sn
ID_{group}
Link T$_1$ to T$_2$
sn
sn
aut$_1$‖aut$_2$=f(K$_{group}$, r$_S$‖sn)
sn +1
aut_1
aut_1
aut$'_1$ ‖ aut$'_2$= f(K$_{group}$, r$_S$ ‖sn)
if aut$'_1$≠aut$_1$
then timeout
else cnf$_2$=f(K$_2$;r$_s$‖r$_2$)
aut'_2
aut'_2,cnf_2
Adversary
T$_x$
r_S
ID_{group}
Link T$_1$ to T$_2$
sn
aut_1
aut$'_1$ ‖ aut$_x$= f(K$_{group}$, r$_S$ ‖sn)
if aut$_x$≠aut$'_1$
then timeout
else cnf$_x$=f(K$_x$, r$_s$‖r$_x$)
aut_x,cnf_x
P$_{group}$=(r$_S$,ID$_{group}$,sn,cnf$_1$, cnf$_x$)

Figure 8.7: A multi-proof attack on Burmester et al.'s protocol

Sundaresan et al. [142] proposed a grouping-proof protocol that complies with the EPCC1Gen2 standard. They claimed that their protocol resists well-known attacks on RFID systems, and provides forward security, which is an open research issue in RFID grouping-proof applications. Their protocol offers forward secrecy, as the messages are computed using a freshly generated random number that is not stored in the tag. Also, the tag stores the current value of data, which is updated after each protocol run; the old data is not stored. Hence, the adversary will not be able to compute the previous messages even if a tag's memory is compromised. The proposed protocol uses simple XoR and 128-bit PRNG operations. Moreover, the reader plays an important role in authenticating the tags, thus providing extra protection against illegitimate tags. However, we showed in Section 8.4.1 that Sundaresan et al.'s protocol is not efficient in terms of memory, computing costs, and server scalability. Server scalability in RFID context refers to the workload on the server to retrieve a single tag should not be proportional to the number of deployed RFID tags (O(n)).

All the above RFID grouping-proof protocols require n rounds to generate the proof. Moriyama [143] proposed an RFID grouping-proof that requires only two rounds. Each tag's message is independent from the other tags' messages and the reader communicates with the tags only in two rounds. This proposal uses a PRF for generating the messages. Moriyama's protocol is shown in Fig. 8.8. Moriyama claimed that his protocol is immune against impersonation and man-in-the-middle attacks. This approach is efficient in terms of performance as the proof can be generated in two rounds and the computing cost is reasonable for RFID tags. However, Moriyama's protocol does not provide forward secrecy; he assumed it is an open problem. Also, the tags reply with their IDs to any reader that queries them, thus affecting tags data anonymity and location tracking.

Reader		T_i
	$\xrightarrow{1, TS, r_R}$	
		Generates r_i $m_i = PRF(K_{group}, (TS, r_R, ID_i, r_i))$
	$\xleftarrow{r_R, r_i, ID_i, m_i}$	
Waits responses from all tags $m_R = (TS, r_R, \{ID_j, r_j, m_j\}_{j \in Z_n})$		
	$\xrightarrow{2, r_R, m_R}$	
		Checks m_j for all $j \in Z_n$ $v_i = PRF(K_i, m_R)$
	$\xleftarrow{r_R, v_i}$	
$P_R = (TS, r_R, \{ID_j, r_j, m_j, v_j\}_{j \in Z_n})$		

Figure 8.8: Moriyama's grouping-proof protocol

A comparison between such protocols in terms of security and tag's performance is demonstrated in Table 8.3 and Table 8.4 respectively.

To conclude, we found that all the discussed related studies have weaknesses. Hence, in the next section, we focus on proposing a grouping-proof protocol that takes into account the strengths of previous protocols and avoids their deficiencies.

8.3 Two Rounds RFID Grouping-Proof Protocol

The proposed protocol is discussed in detail in this section.

8.3.1 Design Goals

For a protocol to support the RFID grouping-proof, it should meet the following goals:

1. Forward secrecy: The proposed protocol should ensure that if an attacker compromises the tag's memory, he/she will not be able to trace previous communication session(s) using previously known messages.

2. Protection against replay attacks: An adversary may eavesdrop on the communications between reader and tag, obtain exchanged messages and resend these

messages repeatedly. Therefore, any generated message should be fresh to the protocol session to protect against replay attacks.

3. Protection against de-synchronisation incidents: The proposed protocol should recover from de-synchronisation incidents, either when an attacker blocks the exchanged message(s), or when the messages are lost during transmission due to system malfunction or communication error.

4. Protection against location tracking: The proposed protocol should confirm that the tag's responses are not static or linkable in order to prevent attackers from tracking the tag's location.

5. Protection against reader/tag impersonation attacks: The proposed protocol should guarantee that the reader's and tag's secret values cannot be obtained by any attacker, thus preventing an attacker from impersonating the reader or the tag.

6. Authentication: The reader and each tag in the grouping-proof should confirm their legitimacy to the server.

7. Concurrency: The tags should only depend on the reader's message to respond, and should not wait for the T_{i-1} message.

8. Reading order independence: The server verifies the proof regardless of the order in which the tags were scanned.

Peris-Lopez et al. [146] provided guidelines for designing a grouping-proof protocol. These guidelines are as follows:

- Computing capabilities: Due to the restricted computation power of RFID tags, the protocol should take into account this limitation. Moreover, the memory and communication costs should also be minimised.

- Matching: Only tags that belong to a group participate in the calculation of the proof, thus reducing the time needed for the server to verify the existence of tags.

- Dependency: A tag should include all messages received from its predecessor tags in its response to prove simultaneity.

- Identification: This is related to tag anonymity and tag location privacy as discussed earlier.

- Verification: Using an encrypted timestamp to avoid predictable timestamps and hence avoid replay attacks.

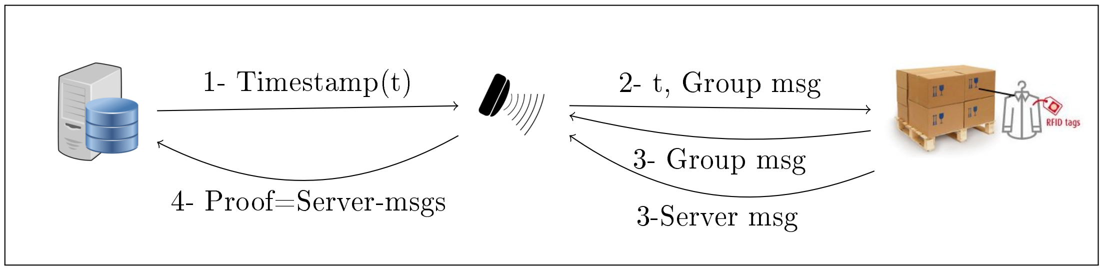


Figure 8.9: System scenario

- Forward secrecy: Peris-Lopez et al. pointed out that forward secrecy in grouping proofs is an open problem, as there are multiple tags, an untrusted reader and offline verifier involved in the process, which makes the updating process complex. However, in this study we have attempted to solve this problem.

In this chapter we take into account these goals and guidelines as they are efficient in terms of security and performance, with the exception of the dependency guideline. The dependency guideline is impractical as it will encounter a heavy computational demand when thousands of tags participate in the grouping-proof.

8.3.2 System Scenario

The proposed system scenario is shown in Fig. 8.9, and summarised as follows:

1. The server sends an encrypted timestamp t to the reader (before the grouping-proof begins).

2. The reader acts as a filter that separates the tags belonging to the group from the tags that do not belong to the group. It computes a *Group message* to link the tags in the group. Then, it broadcasts the *Group message* to the tags.

3. The i^{th} tag, i.e., $1 \leq i \leq n$, verifies the *Group message*. If it succeeds, it sends two messages: the *Group message* to prove it belongs to the group, and the *Server message* to be authenticated by the server.

4. When the reader receives n responses within a pre-defined time window, it verifies the *Group message* for each tag, and generates the *Proof* containing all the received *Server messages*. Then, it sends the *Proof* to the server for later validation of the grouping-proof.

8.3.3 Assumptions

We present an RFID grouping-proof protocol, which operates under the following assumptions:

- We consider an active adversary, who has complete control over all communications in the protocol.
- The tag can compute XoR, generate a random number, and calculate hash functions.
- The reader contacts the tag through a wireless channel that is susceptible to attacks, while the communication channel between the reader and the server is secure.
- The server is offline during the scanning process.
- All the operations in the tag are atomic.
- The groups are pre-defined and static.

8.3.4 Threat Model

Complying with the previous proposed protocols in this thesis, we assume the Dolv-Yao model. Moreover, for grouping-proof, adversarial attacks target the following:

- RFID tags: The adversary can impersonate the reader to collect the tag's responses and forge a grouping-proof.
- RFID reader: The adversary attempts to impersonate the tags to make the reader generates an invalid proof.
- The communication channel: The adversary controls the communication channel by performing the following:
  - Eavesdropping
  - Modifying messages
  - Blocking messages from reaching targets
  - Replaying previous messages
  - Injecting new messages (forgery)
  - Impersonating any entity

Table 8.2: Protocol notation

Notation	Description
ID_G	The shared group unique identity
TS_G	The shared group secret value
ID_i	The i^{th} tag's unique identity
TS_i	The i^{th} tag's secret value
K_{SR}	The reader's private key for signing the proof
K_{PR}	The reader's public key
K_S	The server's secret key
r_x	A random number generated by entity x
H(Z)	The result of generating a hash of data Z, where H: $\{0,1\}^* \rightarrow \{0,1\}^l$
$E_{K_x}(M)$	A message M encrypted with the secret key of entity x
$Sign_{K_x}(Z)$	A signature on data Z, signed using entity x's private key
TS	The server's timestamp
t	A timestamp encrypted by the server's secret key
$\oplus$	An XoR operation
$\|\|$	A concatenation operation
n	The number of tags in the group

8.3.5 Notation

The notation used in the proposed protocol is shown in Table 8.2:

8.3.6 Protocol Description

The proposed protocol is divided into two phases, the setup phase and the grouping-proof phase.

- Setup phase: The manufacturer's server assigns the initial values to the tags, the reader and the server. For a specific group *(G)*, each tag stores (ID_G, ID_i, TS_G, TS_i), the reader stores (K_{SR}, K_{PR}), and the server stores (ID_G, ID_i, TS_G, TS_i, K_S, K_{PR}) for all i^{th} tags belonging to group *G*.

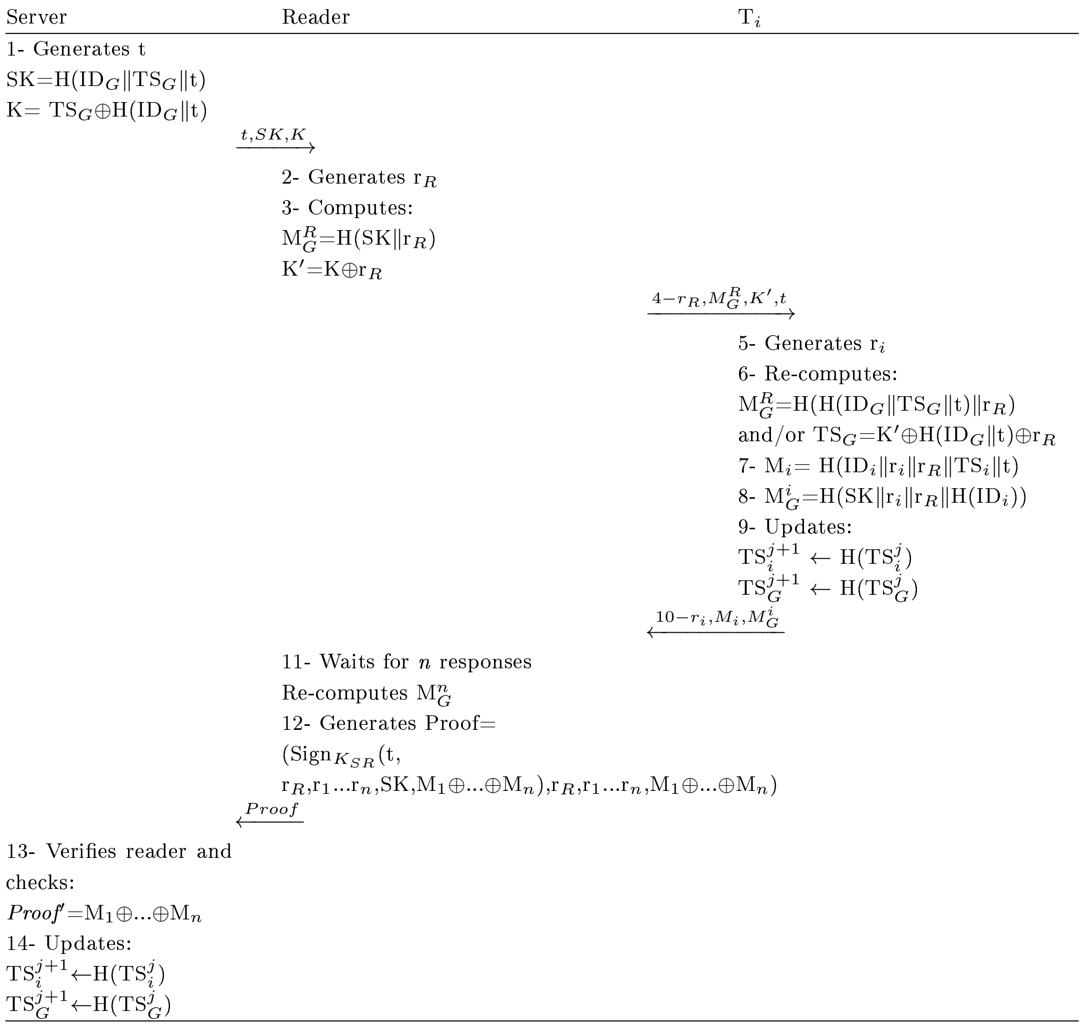
- Grouping-proof phase: a protocol is shown in Table 8.10 and works as follows:

  1. Server: The server *(S)* computes encrypted timestamps t=E_{K_S}(TS). Each timestamp is valid for a limited time-window, within which the reader should respond. The server stores the encrypted timestamps and time-window. The server computes an index session message with the reader as SK=H(ID_G $\|$ TS_G $\|$ t), where t is the current encrypted timestamp, and K= $TS_G \oplus$ H(ID_G $\|$ t) to inform the tags of the current value of TS_G in case a desynchronisation incident occurs. The server sends t, SK and K to the reader. Moreover,

the server sends a list of the hashed tags' identifiers to identify missing tags.

Round 1:

2. Reader: The reader generates a random number r_R, i.e., $r_R \in \{0,1\}^L$, where L is the security parameter.
3. Reader: The reader computes two messages to link the tag chain: the first message is M_G^R=H(SK $\|$ r_R), and the second message is $K' = K \oplus r_R$.
4. Reader: The reader broadcasts r_R, t, M_G^R and K'.
5. Tag: The i^{th} tag T_i generates a random number r_i, i.e., $r_i \in \{0,1\}^L$.
6. Tag: T_i re-computes M_G^R to check that it belongs to the group. If it succeeds, T_i performs the next step. If it fails, this implies either:
   (a) T_i has been desynchronised in the previous session(s) resulting in failure to update the value of TS_G. In this case, it needs to obtain the current value of TS_G from message (K'). If succeeds, T_i performs the next step.
   (b) T_i does not belong to the group if it fails to re-compute message (K'), therefore it aborts the session.
7. Tag: T_i calculates:
   M_i= H(ID_i $\|$ r_i $\|$ r_R $\|$ TS_i $\|$ t) to be included in the grouping-proof and verified by the server.
8. Tag: T_i calculates:
   M_G^i= H(SK $\|$ r_i $\|$ r_R $\|$ H(ID_i)) to prove to the reader that it belongs to the group.
9. Tag: T_i updates $TS_i^{j+1} \leftarrow H(TS_i^j)$, where j is the current session, and $TS_G^{j+1} \leftarrow H(TS_G^j)$ to be used in the next session *j+1.*
10. Tag: T_i sends r_i, M_i and M_G^i to the reader.
    These steps are performed for each i^{th} tag in the group until the reader receives n responses.
    Round 2:
11. Reader: When the reader receives n responses within a pre-defined time window, for every received r_i , the reader checks that M_G^i= h(SK $\|$ r_i $\|$ r_R $\|$ h(ID_i)), to confirm that only tags belonging to the group are included in the proof.



Server		Reader		T_i
1- Generates t $SK=H(ID_G \| TS_G \| t)$ $K= TS_G \oplus H(ID_G \| t)$				
	$\xrightarrow{t,SK,K}$			
		2- Generates r_R 3- Computes: $M_G^R=H(SK \| r_R)$ $K'=K \oplus r_R$		
			$\xrightarrow{4-r_R,M_G^R,K',t}$	
				5- Generates r_i 6- Re-computes: $M_G^R=H(H(ID_G \| TS_G \| t) \| r_R)$ and/or $TS_G=K' \oplus H(ID_G \| t) \oplus r_R$ 7- $M_i= H(ID_i \| r_i \| r_R \| TS_i \| t)$ 8- $M_G^i=H(SK \| r_i \| r_R \| H(ID_i))$ 9- Updates: $TS_i^{j+1} \leftarrow H(TS_i^j)$ $TS_G^{j+1} \leftarrow H(TS_G^j)$
			$\xleftarrow{10-r_i,M_i,M_G^i}$	
		11- Waits for n responses Re-computes M_G^n 12- Generates Proof= $(Sign_{K_{SR}}(t,$ $r_R,r_1...r_n,SK,M_1 \oplus ... \oplus M_n),r_R,r_1...r_n,M_1 \oplus ... \oplus M_n)$		
	$\xleftarrow{Proof}$			
13- Verifies reader and checks: $Proof'=M_1 \oplus ... \oplus M_n$ 14- Updates: $TS_i^{j+1} \leftarrow H(TS_i^j)$ $TS_G^{j+1} \leftarrow H(TS_G^j)$				

Figure 8.10: The proposed grouping-proof protocol

12. Reader: For each tag belonging to the group, the reader generates *Proof* containing the received messages (M_i ... M_n), i.e. Proof= ($Sign_{K_{SR}}$(t, r_R, r_1 ... r_n, SK, $M_1 \oplus ... \oplus M_n$), r_R, r_1 ... r_n, $M_1 \oplus ... \oplus M_n$), and then sends it to the server.

13. Server: Later, the server verifies the reader's signature, checks the timestamp, and retrieves tags' data based on the value of index SK. Then, it computes the expected grouping-proof (*Proof'* = $M_1 \oplus ... \oplus M_n$) regardless of the order the tags were scanned, and compares the result of *Proof'* with the received value of ($M_1 \oplus ... \oplus M_n$) in *Proof*. If there is a match, the server believes that all the tags in the grouping-proof are present and legitimate.

14. Server: The server updates:

$TS_i^{j+1} \leftarrow H(TS_i^j)$ for all the legitimate tags in the group, and updates $TS_G^{j+1} \leftarrow H(TS_G^j)$.

8.4 Protocol Analysis

In this section we present the analysis of the proposed protocol in terms of informal, and formal analysis using CasperFDR and Scyther.

8.4.1 Informal Protocol Analysis

If the reader did not receive messages from a tag, whether the tag was not present, or time period has expired before receiving the tag's messages, the following process is performed:

- Missing tag(s): If the reader does not receive a response from the *4th* tag for example, it informs the server about the missing tag by including the missing tag's *H(ID$_4$)* value, which was not retrieved from the hashed identifiers list during the protocol running, within *Proof*. The reader generates Proof= ($Sign_{K_{SR}}$(t, r_R, r_{n-1}, SK, M_{n-1}, H(ID_4)), then, sends *Proof* to the server. The server retrieves the missing tag data based on SK and H(ID_4) values, and calculates *Proof'* without taking into account the *4th* tag. The server should also alert the system about the missing tag.

The proposed protocol attempts to meet the goals discussed in Section 8.3.1 as follows:

1. Forward secrecy: The values of TS_i and TS_G are updated after each run in order to prevent forward secrecy invasion using a hash function that is irreversible. If an adversary compromises the tag T_i memory, he/she will not be able to trace the previous communications of the tag as $(M_i)^{j-1}$ and $(M_G^i)^{j-1}$ involve the use of previous secret values TS_i and TS_G, which are not stored in the tag. The stored updated values of TS_i and TS_G are used in the calculation of the next session and cannot be irreversible as a result of using a hash function. Hence, the attacker cannot re-compute the previous messages.

2. Protection against replay attacks: The inclusion of random numbers in the messages is vital to confirm that the messages are intended for a specific reader or tag, which originally generate r_R and r_i respectively. Moreover, since r_R and r_i are fresh random numbers, it is impossible for the attacker to predict them in the next session. Replaying the reader's message will not pose a threat to the

protocol, as the server will detect such attacks; the server will not accept proofs that were generated with expired timestamps.

Regarding the tag's message M_i, it contains a timestamp that is encrypted by the server to avoid predictable timestamps and is valid for a limited time during the session. Hence, the attacker cannot re-send the previous M_i message as the timestamp will not be valid.

3. Protection against desynchronisation incidents: There are two scenarios, where a desynchronisation of data might occur, but the proposed protocol tackles these incidents as follows:

   - If an attacker blocks the i^{th} tag messages (r_i, M_i, M_G^i), the reader will detect that there is a missing tag, and perform *Missing tag* step.
   - If an attacker blocks the reader's messages from reaching the i^{th} tag T_i, the server will update TS_G while T_i's TS_G value will remain the same, thus causing a desynchronisation in the next session. However, our protocol can prevent this attack as the server computes an additional message (K) that contains the current updated value of TS_G, and can only be obtained by the legitimate tag belonging to the group (based on ID_G).

4. Tag location tracking: The tags' messages involve fresh random numbers, timestamps, and/or updated TS_i and TS_G values, which means that the responses are not static. Hence, an attacker will not be able to track the tag's location. Moreover, even if the i^{th} tag does not update its values, the tag's messages are randomised with the inclusion of random numbers (r_R, r_i) and timestamp.

5. Protection against tag impersonation attack: The adversary needs to be in possession of 128-bit secret values, such as ID_i, ID_G, TS_i, and TS_G, which are not sent in clear, in order to impersonate or clone T_i. Moreover, it will take up to 2^{128} attempts to guess each secret value.

6. Protection against reader impersonation attack: The attacker has to have a valid digital signature private key to impersonate a legitimate reader. Moreover, even if the attacker replays previous messages to the tags and impersonates the reader, the server will detect such an attack.

7. Authentication: The reader verifies the existence of each tag in the group based on M_G^n messages, and the server verifies the legitimacy of each tag in the group based on M_i^n messages.

8. Matching: The reader verifies that only the tags that belong to the group participate in the proof.

9. Verification: To prevent replay attacks on the server side, an encrypted timestamp is generated by the server and sent to the tags, so an attacker cannot predict future timestamps as the actual timestamp is encrypted by the server.

10. Concurrency: Reducing time delay by allowing all the tags to compute their messages without waiting for the other tags' responses; they send their responses when they receive the reader's messages.

11. Reading order independence: The reader generates the proof using an XoR operator, so all tags' responses can be verified in random order regardless of the order in which the tags were scanned.

Inspired by [146], Table 8.3 demonstrates a comparison between our protocol and the other yoking/grouping-proof protocols in terms of security. As shown in Table 8.3, the proposed protocol protects the system from different attacks, and provides forward secrecy. Sundaresan et al.'s protocol [142] provides similar protection but it has major issues in terms of performance as shown in Table 8.4. In Sundaresan et al.'s protocol the number of rounds is proportional to the number of tags, the tags need to compute twelve PRNG functions, and the server needs to perform *O(n)*, computational operations to authenticate the tags in the group, which implies that their protocol is large in terms of tag's computing cost and server scalability respectively, making their protocol heavy to implement.

Inspired by [116] and [146], Table 8.4 illustrates the performance of the proposed protocols compared with the other grouping-proof only. We took into account the performance of the RFID tags not the reader. Our protocol is efficient for the following reasons: Firstly, our protocol enhances performance by engaging the tags in one round only and the whole protocol is executed in two rounds as in Moriyama's protocol [143]. However, Moriyama's protocol does not provide forward secrecy; they assumed it is an open problem. Also, any attacker can impersonate the reader, as the reader does not provide any proof of its identification to the tag, and obtained the tag's ID, hence tag's privacy is not achieved. Secondly, our protocol provides concurrency and reading order independence features that reduce time delay and failure rates respectively. Thirdly, the performance of our protocol is appropriate for RFID tags, as it does not require mass memory storage; it stores four values each of which is 128-bit length, and requires average communication cost as each tag only sends three messages to the reader (384 bits). Fourthly, regarding the server indexing scalability, the server retrieves the tags'

Table 8.3: Security comparison of yoking/grouping-proof protocol

	Juels [106]	Saito [144]	Piramuthu [145]	Chien [147]	Bolotny [148]	Burmester [149]	Ma [116]	Sundaresan [142]	Lien [18]	Peris-Lopez [146]	Moriyama [143]	Our protocol 8.3
Forward secrecy invasion	×	×	×	×	×	×	×	✓	×	×	×	✓
Replay attack	×	×	×	×	×	✓	✓	✓	✓	✓	✓	✓
Tag impersonation	×	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓
Reader impersonation	×	×	✓	×	✓	×	✓	✓	✓	✓	×	✓
Traceability	×	✓	✓	✓	✓	✓	✓	✓	×	✓	×	✓

✓: Means resists such an attack
×: Means does not protect against such an attack

Table 8.4: Tag's performance comparison of grouping-proof protocols

	Chien [147]	Bolotny [148]	Burmester [149]	Ma [116]	Sundaresan [142]	Lien [18]	Moriyama [143]	Our protocol 8.3
# Rounds	n	n	n	n	n	n	2	2
# Messages	5	3	3	5	4	4	4	3
# Hashing	-	2	2	-	-	1	-	6*
# PRNG	3	-	-	12	12	1	3	1
# stored data	2	3	4	4	9	2	3	4
Server retrieving cost	O(1)	O(n)	O(n)	O(n)	O(n)	O(1)	O(1)	O(1)
Order independent	×	×	×	×	×	✓	×	✓
Concurrency	×	×	✓	×	×	✓	✓	✓

*The ith tag might compute 7 hash functions in case it has been de-synchronised
-: Means not applicable, ×: Means does not provide such feature, ✓: Means provides this feature

data based on the value of SK, hence the retrieval computing cost for identifying a tag in the server is *O(1)*.

8.4.2 Formal Protocol Analysis

This section presents the formal analysis of the proposed protocol using CasperFDR [104] and Scyther [75] tools. The aim of this section is to prove that the data exchanged between the tag and reader is secure, not to prove the authenticity of the generated proof.

CasperFDR Analysis of the Proposed Protocol

The script is shown in Appendix D.1. The variables defined in the script are :

#Free variables
R, Ti: Agent
S: Server

rR, ri: Nonce
IDG, IDi, SK,: Data
h: HashFunction
t: TimeStamp
InverseKeys = (h,h)

The security specifications for which CasperFDR evaluated the communication channel between the reader and a tag is as shown below:

1. *Agreement (R, Ti, [SK])*, which means that the tag (Ti) is authenticated to the reader (R), and both parties agree on the data value SK.

2. *Secret (S, IDG, [Ti])*, and *Secret (S, IDi, [Ti])*, which means that the value of IDG and IDi should be secret between the server (S) and the tag (Ti).

The intruder is defined to be Mallory, who knows all the entities and the exchanged random numbers as shown below:

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Server1, Reader, Tagi, Mallory, Rr, Ri, Rm}

The CasperFDR tool evaluated the protocol and did not find any attacks.

Scyther Analysis of the Proposed Protocol

Three roles are identified in the script shown in Appendix D.2: a server, a reader and a tag. The defined variables are:

fresh Time: Timestamp;
fresh IDi: Data;
fresh TSi: Data;
fresh IDG: Data;
fresh TSG: Data;
fresh SK: Data;
var ri, rR: Nonce;

Each role specifies the goals that the protocol attempts to meet. These goals are within the *Claim* section. In the protocol, there are, for example, seven *Claim* goals specified in the tag role as follows:

claim_ Ti1(Ti, Secret, IDG);
claim_ Ti2(Ti, Secret, TSG);
claim_ Ti3(Ti, Secret, IDi);
claim_ Ti4(Ti, Secret, TSi);
claim_ Ti6(Ti, Alive);
claim_ Ti7(Ti, Niagree);
claim_ Ti8(Ti, Nisynch);

After running the script, Scyther did not find any feasible attacks within bounds, which means no attack was found within the bounded state space.

8.5 Protocol Implementation and Performance Measurement

In this section, we present the performance measurements of the proposed protocol.

8.5.1 Implementation Process

The tag is provided with four 128-bit secret values namely ID_i, TS_i, ID_G, and TS_G, which are stored in the tag's EEPROM. Similarly, the reader acts as the server and stores the same values as the tag.

The reader starts by generating a 128-bit random number (r_R) using the *rngCsp* method and a 64-bit timestamp (t) using the *DateTime.Ticks* property in .Net Framework. Then, it computes (M_G^R) and K, and sends r_R, t, M_G^R and K to the tag by sending 28 *WriteTagData_ EPC_ C1G2* commands. When the tag receives the reader's messages, it generates a random number r_i (128-bit) using the built-in PRNG, and re-computes M_G^R. If there is a match, it computes the two messages, namely M_i and M_G^i, each of which is 128-bit length. Subsequently, the tag writes (r_i, M_i and M_G^i) in its memory using the *syscall_ writeWord* commands ready to be read on demand. The reader later sends three *ReadTagData_ EPC_ C1G2* commands to read (r_i, M_i and M_G^i), and re-computes M_G^i . If there is a match with the received M_G^i, it generates the Proof using RSA provided by .NET Framework.

8.5.2 Performance Measurement

The performance measurements are as follows:

1. DemoTag memory cost: In the proposed protocol, the memory cost is:
   - 348 bytes used from the 4 KB EEPROM memory for storing the tag's data, messages (responses), and random numbers.
   - 33.6 KB used from the 128 KB Flash memory to store the tag's firmware.

Table 8.5: Data exchange time cost

	Tag	Reader
Read	-	418.36 ms
Write	1.4 ms	2.23 sec

2. Data exchange time cost is shown in Table 8.5: For the tag to write (r_i, M_i and M_G^i) into its memory to be read by the reader, the average timer counter after running the protocol 100 times is 22548, so, based on equation 3.1, the time cost is:

$$22548 * 0.0000000625 = 0.0014 sec \equiv 1.4 ms \tag{8.1}$$

   Regarding the reader, we found that the reader needs *2.23 sec* to write (r_R, t, M_G^R and K) into the tag's memory, and *418.36 ms* to read the whole tag's responses.

3. DemoTag computing cost: In a successful run of the proposed protocol, the tag generates a random number, computes three messages, updates two values and writes the messages in the memory. Table 8.6 shows that the time cost of running the protocol on the DemoTag is around *9.71 ms*.

Table 8.6: Computing operations time cost (milliseconds)

	R2	M_G^R	M_i	M_G^i	Update	Write
Computing cost	0.11	1.65	1.71	1.71	3.13	1.4
Total	9.71					

All the other proposed grouping-proof protocols discussed in this chapter did not present any performance measurements; hence, we could not provide a performance comparison with such protocols.

8.6 Summary

In this chapter, we discussed another field in authenticating a group of RFID tags, where the server is offline during the reading process; this is known as an offline RFID grouping-proof. The challenges in designing a grouping-proof protocol reside in the absence of the server. Based on the main goals in designing a secure RFID grouping-proof, we designed a grouping-proof protocol that tackles these challenges with a low probability of delay in the responses, as the tags respond in two rounds, and do not need to wait for their predecessor tags' messages, and with a low probability of failure, as the server verifies the proof regardless of the order in which the tags were scanned. Our solution also improves existing related work by protecting the system from active attacks and providing forward secrecy, which is assumed to be an open problem. We then informally analysed the proposed protocol and this analysis was subsequently extended to formal analysis by CasperFDR and Scyther. Finally, we implemented the proposed protocol in order to measure the performance of the tag's limited memory and computing resources, and the results showed that the protocol can be implemented with a relatively low memory and computing costs.

Chapter 9

Conclusion and Future Work

Contents

In this chapter we conclude the thesis by summarising our contributions and discussing potential future work.

9.1 Summary and Conclusions

The focus of this thesis directed towards the security and privacy concerns relating to the use of passive RFID tags. Providing a lightweight mutual authentication for RFID tags in different schemes is a very active research area, which has resulted in the publication of hundreds of RFID authentication protocols. Many of these RFID mutual authentication protocols claim to be secure in the presence of a malicious agent, who is assumed to have complete control over the communications network. Unfortunately, we found that some of the protocols that have been suggested, do not succeed in their stated goals. Hence, in this thesis, we focused on proposing mutual authentication protocols that provide adequate levels of security and privacy for several RFID schemes, and avoid the weaknesses found in related work. We formally analysed some of the protocols' main goals such as providing protection against data leakage, replay and impersonation attacks, and preserving the privacy of the transmitted data.

The five main contributions are listed below:

- One way to avoid replay attacks, forward secrecy invasion and tracking the location of the tag's holder is to update the tag's data on the server and the tag after each successful session. However, updating a tag's data comes with a cost. If the receiver does not receive a response, it will not update the data, resulting in a data desynchronisation in the next session, as the sender has already updated its data. We discovered that data desynchronisation attacks can be applied to one of the widely cited RFID mutual authentication protocols [39, 89]. Although the Song protocol updates the data after a successful authentication session and the server stores the old and new tag's data, it is vulnerable to data desynchronisation, as the server updates the data even if the received data matches the old values. Therefore, an adversary can easily cause synchronisation failure by intercepting and blocking messages during transmission in two or more consecutive sessions, resulting in mismatched values. Hence, we proposed a modified data update process that mitigates the data desynchronisation incidents. This contribution is discussed in Chapter 4 and published in [66]. We also found that this attack can be applied in other proposals such as [103, 102].

- Besides reviewing the Song protocol, we reviewed other proposals in the field of RFID mutual authentication protocols. We found that attacks on RFID mutual authentication proposals are still ongoing, and these weaknesses in other's protocols formed a benchmark for us to propose a new RFID mutual authentication protocol that improves upon such weaknesses. The wireless communication channel between reader and tag may allow an attacker to perform active attacks, such as replay, impersonation, DoS, location tracking, forward secrecy invasion, and compromising a tag's privacy. The proposed RFID mutual authentication protocol attempted to provide immunity against such attacks and has been formally analysed to confirm that it achieves the desired protection. The performance of our protocol has been measured to prove its efficiency. This contribution is shown in Chapter 5 and published in [150].

- In RFID-enabled supply chains, hundreds or even thousands of tagged items need to be tracked and identified remotely. Normally, in supply chain practice, there is a lack of trust between the parties involved, as the products' owner may not know the next owner. Moreover, for the RFID reader to identify each passively tagged item, the wireless channel between the reader and tags is vulnerable to active attacks. One solution to such issues is to encrypt the data with a secret key. However, the question remains of how to distribute the secret key securely between such parties, and protect the tagged items' data in transit. To this end, we proposed using Shamir's [127] secret-sharing approach (threshold scheme), where the secret key is divided into shares that, individually, do not provide any useful information about the secret. Each tag stores one share in its memory. In our proposal, the tag's ID is encrypted with a secret key, which is not stored in the tag. To extract the tag's ID, the reader must collect sufficient shares to decrypt the tag's encoded-ID. Moreover, to allow the next owner to change the threshold scheme, we proposed a secret key update protocol incorporating a resynchronisation capability to counter the disruptive effects of location tracking, replay attacks and desynchronisation incidents. This contribution is illustrated in Chapter 6 and published in [126].

- So far in this thesis, the discussed proposals suit a conventional RFID deployment with the assumption of a secure server. However, many physical servers are being migrated to cloud solutions, so we investigated this aspect in Chapter 7. Cloud computing represents a new era in information technology that presents substantial benefits to sectors and organisations. Processing and storing RFID tags data in the cloud provides a promising solution with powerful computing and massive storage ability. Nevertheless, confidentiality and privacy are regarded as two of the main concerns in cloud computing. Cloud-based RFID has gained relatively little attention in the literature. Hence, we attempted to review the current work. We found that the first proposal to protect tags' data from an untrusted cloud provider is vulnerable to reader-to-tag impersonation attacks, and man-in-the-middle attacks. Hence, we proposed an enhanced version of their protocol that takes into account such vulnerabilities. The proposed protocol was formally analysed to provide an indicative results about its security, and implemented to show its efficiency. This contribution is published in [77].

- To this point we had only considered security protocols, where there is only one tag being read at a time. However, there are other applications that require a group of items to be presented for reading. Reading all the tags in a group should be done within a specific time period and there must be no way for an illegitimate tag to join the group. These issues are addressed under the research heading of "grouping proofs", which are considered in Chapter 8. The grouping-proof is normally generated by the reader to confirm the legitimacy and the simultaneous presence of the tagged items in the group. The challenge in designing such proofs is that the server that authenticates the proof is not present during the scanning process, and this could allow a fake tag to participate in the group, which means the proof should be completed within a time window. Furthermore, the wireless communication between the reader and the tags is prone to active attacks. Hence, we proposed an RFID grouping-proof that involves only two rounds to generate the proof, and provides immunity against the attacks found in the literature. The proposed protocol has been formally analysed and implemented to prove its secrecy and efficiency respectively. This contribution is published in [151].

9.2 Reflection on Citations

During my PhD six papers were submitted: four to international conferences [66, 126, 152, 151], one to a workshop [77] and one to a journal [150]. Some of these published papers have been cited in scientific papers as shown below.

Our proposals [66, 126, 77] were cited in [45, 92, 91]. Based on the data desynchronisation, reader impersonation, man-in-the-middle attacks, and location tracking that we found in the Song protocol [89], Cai et al. key update protocol [70], and in the cloud-based RFID protocol [130], [45, 92, 91] designed their own protocol taking into account the attacks we found.

In [125, 45], the authors criticised the assumptions we made in designing our protocols in [126, 77]. For example, in [126], we assumed that the attacker cannot obtain enough shares from the tags in transit to recover the secret key in the RFID-enabled supply chain; while in [125], the authors assumed that the attacker can obtain such shares. They therefore proposed using dummy tags that contain a random (bogus) secret share when the tags are transferred between the supply chain parties. In [77], we assumed that the communication channel between the reader and cloud server is secure in RFID cloud-based systems; in [45], the authors contradicted our assumption and assumed that the channel between the cloud and a mobile reader is not secure. Hence, their protocol attempted to protect the data in transit between tag-reader and mobile reader-cloud server. These assumptions will be taken into account in proposing future protocols that will extend and enhance our current proposals.

9.3 Future Work

There is potential for further research into topics discussed in the thesis as follows:

- Deploying the Universal Composable (UC) Framework for analysis purposes:

  In this thesis, we analysed the proposed protocols using formal analysis tools such as CasperFDR and Scyther. Both tools work under the assumption of perfect cryptography and use the Dolev-Yao model [67] as the threat model. Some issues have been found in CasperFDR, such as *state space explosion*, where the state space grows exponentially with the number of runs, and exploration becomes infeasible. For instance, when the number of protocols run, the amount of data used, or the number of specification goals are high, the memory may run off and the FDR checker will stop working. In Scyther, similar issues have arisen, especially with heavy protocols that require many runs of the protocol in parallel. Moreover, these tools do not provide checking with regard to privacy, such as untraceability and forward secrecy. Finally, the data desynchronisation attack that was discussed in Section 4.2, is a post-protocol process (not within the communication channel) that cannot be checked by CasperFDR and Scyther.

  Another method for formally analysing the security protocols is called a "Universal Composable Framework" (UC) [153]. In the UC framework, there is an *ideal process* that conducts the protocol's tasks in a secure way, where all the parties send their inputs to a trusted third entity that processes the inputs, and then sends back the outputs to these parties. An adversary S is restricted to corrupting some of the parties and blocking messages. This can be regarded as the *security requirements* of the protocol. In the *real-life* protocol, the parties carry out the protocol execution and generate the outputs. Then, the UC framework compares a *real-life* execution of a protocol with the *ideal process*. If running the protocol in the real-life model amounts to "emulating" the ideal process for that task, then the protocol is considered as secure. The UC framework includes definitions of some common cryptographic tasks, such as authenticated and secure communication, key-exchange, public-key encryption, signature, commitment, oblivious transfer, zero-knowledge, secret sharing, and general function evaluation.

- Studying a wider range of physical attacks:

  The proposals in this thesis consider active attacks. However, physical attacks such as side channel attacks, power analysis, and traffic analysis were not taken into account. A study on these attacks and taking them into consideration when designing a secure RFID system will form part of future work.

- Server scalability:

  Most of the proposed RFID protocols focused on the tag's performance and ignored the server scalability performance. When a server receives a tag's message it searches the whole database to find a match with the received message, and this could exhaust the server's computing capability. More specifically, performing an exhaustive search to identify individual tags is difficult when the number of tags is large. Some of the techniques that already are deployed include binary search [154], and hash tables [100]. Utilising such techniques in RFID protocols will be a related part of future work.

Bibliography

[1] P. Peris López, “Lightweight cryptography in radio frequency identification (RFID) systems,” Ph.D. dissertation, Carlos III University of Madrid, 2008.

[2] G. Lowe, “Breaking and fixing the Needham-Schroeder public-key protocol using FDR”, in *Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 1996, pp. 147–166.

[3] A. Mathuria, G. Sriram, “New attacks on ISO key establishment protocols”, in *IACR Cryptology ePrint Archive*, Citeseer, 2008.

[4] G. Bertoni, J. Daemen, M. Peeters, G. Assche, “Keccak”, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2013, pp. 313–314.

[5] H. Gilbert, M. Robshaw, Y. Seurin, “Good variants of HB+ are hard to find”, in *International Conference on Financial Cryptography and Data Security*, Springer, 2008, pp. 156–170.

[6] H. Gilbert, M. Robshaw, Y. Seurin, “Active attack against HB^+: a provably secure lightweight authentication protocol”, in *Electronics Letters*, The Institution of Engineering & Technology, vol. 41, no. 21, pp. 1, 2005.

[7] , A. Juels, S. Weis, “Authenticating pervasive devices with human protocols”, in *Annual International Cryptology Conference*, Springer, 2005, pp. 293–308.

[8] N. Hopper, M. Blum, “Secure human identification protocols”, in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2001, pp. 52–66.

[9] L. Kulseng, Z. Yu, Y. Wei, Y, Guan, “Lightweight mutual authentication and ownership transfer for RFID systems”, in *IEEE Proceedings INFOCOM*, IEEE, 2010, pp. 1–5.

[10] S. Kardas, Suleyman and M. Akgün, M. Kiraz, H. Demirci, "Cryptanalysis of lightweight mutual authentication and ownership transfer for RFID systems", in *Workshop on Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec)*, IEEE, 2011, pp. 20–25.

[11] D. Holcomb, W. Burleson, K. Fu, Kevin, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags", in *Proceedings of the Conference on RFID Security*, Springer, 2007.

[12] T. Karygiannis, B. Eydt, G. Barber, L. Bunn, T. Phillips, "Guidelines for securing radio frequency identification (RFID) systems", in *NIST Special publication*, vol. 80, 2007, pp. 1–154.

[13] X. Leng, K. Mayes, K. Markantonakis, "HB-MP$^+$ protocol: An improvement on the HB-MP protocol", in *IEEE International Conference on RFID*, IEEE, 2008, pp. 118–124.

[14] G. Lowe, B. Roscoe, "Using CSP to detect errors in the TMN protocol", in *IEEE Transactions on Software Engineering*, vol. 23, no. 10, pp. 659–669, 1997.

[15] H. Kim, I. Kim, K. Han, J. Choi, "Security and privacy analysis of RFID systems using model checking", in *High Performance Computing and Communications*, Springer, 2006, pp. 495–504.

[16] A. Juels, R. Rivest, M. Szydlo, "The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy", in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ACM, 2003, pp. 103–111.

[17] A. Mitrokotsa, M. Rieback, A. Tanenbaum, "Classifying RFID attacks and defences", in *Information Systems Frontiers*, vol. 12, no. 5, pp. 491–505, 2010.

[18] Y. Lien, X. Leng, K. Mayes, J. Chiu, "Reading order independent grouping proof for RFID tags", in *IEEE International Conference on Intelligence and Security Informatics (ISI 2008)*, IEEE, 2008, pp. 128–136.

[19] T. Li, P. Peris-Lopez, L. Hernandez-Castro, *Security and Trends in Wireless Identification and Sensing Platform Tags: Advancements in RFID*, in *Information Science Reference*, 2013.

[20] C. Hoare, "Communicating Sequential Processes", Springer, 1978.

[21] Manual, "Failures-Divergence Refinement", in *Formal Systems (Europe) Ltd., Oxford University Computing Laboratory, 9 edition*, 2000.

[22] M. Roberti, "The history of RFID technology," *RFID journal*, vol. 16, no. 01, 2005.

[23] H. Stockman, "Communication by means of reflected power," *Proceedings of the IRE*, vol. 36, no. 10, pp. 1196–1204, 1948.

[24] S. A. Weis, "RFID(radio frequency identification): Principles and applications," *System*, vol. 2, p. 3Principles, 2007.

[25] M. R. Rieback, B. Crispo, and A. S. Tanenbaum, "The evolution of RFID security," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 62–69, 2006.

[26] R. Das and H. P. (2015, October) RFID forecasts, players and opportunities 2016-2026. IDTechEX. [Online]. Available: http://www.idtechex.com/research/reports/rfid-forecasts-players-and-opportunities-2016-2026-000451.asp

[27] *RFID for Item Management. ISO/IEC 18000*, International Organization for Standardization (ISO) Std., 2004.

[28] *EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID Specification for RFID Air Interface Protocol for Communications at 860-960 MHz Version 2.0.0 Ratified*, EPCGlobal, 2013.

[29] D. Klair, K. Chin, R. Raad, "A survey and tutorial of RFID anti-collision protocols", in *IEEE, Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 400–421,

[30] P. Nikitin, K. Rao, "Helical antenna for handheld UHF RFID reader", in *IEEE International Conference on RFID (IEEE RFID 2010)*, IEEE, 2010, pp. 166–173.

[31] S. Sandoval-Reyes, J. Perez, "Mobile RFID reader with database wireless synchronization", in *The 2nd International Conference on Electrical and Electronics Engineering*, IEEE, 2005, pp. 5–8.

[32] *Identification cards –Contactless integrated circuit(s) cards – Proximity cards. ISO/IEC 14443*, International Organization for Standardization (ISO) Std., 2003.

[33] H. Lehpamer, "RFID design principles", Artech House, 2012.

[34] *Identification cards –Contactless integrated circuit(s) cards – Vicinity cards. ISO/IEC 15693*, International Organization for Standardization (ISO) Std., 2003.

[35] "EPC radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 MHz-960 MHz," 2008. [Online]. Available: http://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_1_0_9-standard-20050126.pdf

[36] S. E. Sarma, S. A. Weis, and D. W. Engels, *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13-15, 2002 Revised Papers.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. RFID Systems and Security and Privacy Implications, pp. 454–469. [Online]. Available: http://dx.doi.org/10.1007/3-540-36400-5_33

[37] J. Waldrop, D. W. Engels, and S. E. Sarma, "Colorwave: A mac for rfid reader networks," in *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, vol. 3. IEEE, 2003, pp. 1701–1704.

[38] Z. Bilal, "Addressing security and privacy issues in low-cost RFID systems," Ph.D. dissertation, Royal Holloway, University of London, 2015.

[39] B. Song, "RFID authentication protocols using symmetric cryptography," Ph.D. dissertation, Royal Holloway, University of London, December 2009.

[40] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C.* john wiley & sons, 2007.

[41] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and implementation of PUF-based "unclonable" RFID ICs for anti-counterfeiting and security applications," in *2008 IEEE International Conference on RFID.* IEEE, 2008, pp. 58–64.

[42] P. Tuyls and L. Batina, "Rfid-tags for anti-counterfeiting," in *Cryptographers' Track at the RSA Conference.* Springer, 2006, pp. 115–131.

[43] L. Bolotnyy and G. Robins, "Physically unclonable function-based security and privacy in RFID systems," in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07).* IEEE, 2007, pp. 211–220.

[44] A.-R. Sadeghi and D. Naccache, *Towards Hardware-Intrinsic Security: Foundations and Practice.* Springer-Verlag New York, Inc., 2010.

[45] Q. Dong, J. Tong, Y. Chen, "Cloud-Based RFID Mutual Authentication Protocol without Leaking Location Privacy to the Cloud", in *International Journal of Distributed Sensor Networks.* Hindawi Publishing Corporation, 2015.

[46] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Towards Hardware-Intrinsic Security.* Springer, 2010, pp. 135–164.

[47] J. Bringer, H. Chabanne, and T. Icart, "Improved privacy of the tree-based hash protocols using physically unclonable function," in *International Conference on Security and Cryptography for Networks.* Springer, 2008, pp. 77–91.

[48] M. Akgun and M. U. Caglayan, "PUF based scalable private RFID authentication," in *Sixth International Conference on Availability, Reliability and Security (ARES 2011).* IEEE, 2011, pp. 473–478.

[49] M. Akgün and M. U. ÇaÇğlayan, "Providing destructive privacy and scalability in RFID systems using PUFs," *Ad Hoc Networks*, vol. 32, pp. 32–42, 2015.

[50] J. Bringer, H. Chabanne, and E. Dottax, "Hb^{++}: a lightweight authentication protocol secure against some attacks," in *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU'06).* IEEE, 2006, pp. 28–33.

[51] S. Piramuthu, "HB and related lightweight authentication protocols for secure RFID tag/reader authentication," *CollECTeR Europe 2006*, p. 239, 2006.

[52] J. Munilla and A. Peinado, "HB-MP: A further step in the HB-family of lightweight authentication protocols," *Computer Networks*, vol. 51, no. 9, pp. 2262–2267, 2007.

[53] E. Pagnin, A. Yang, G. Hancke, A. Mitrokotsa, "Hb^{+}db mitigating man-in-the-middle attacks against HB^{+} with distance bounding," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks.* ACM, 2015.

[54] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, Ö. Küçük, and B. Preneel, *MAME: A compression function with reduced hardware requirements.* Springer, 2007.

[55] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash." in *CHES*, vol. 6225. Springer, 2010, pp. 1–15.

[56] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, *PRESENT: An ultra-lightweight block cipher.* in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2007, pp. 450–466.

[57] C. Rolfes, A. Poschmann, G. Leander, C. Paar, "Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents" in *International Confer-*

ence on Smart Card Research and Advanced Applications. Springer, 2008, pp.89–203.

[58] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, and Y. Seurin, “Hash functions and RFID tags: Mind the gap,” in *Cryptographic Hardware and Embedded Systems–CHES 2008*. Springer, 2008, pp. 283–299.

[59] C. De Cannière, O. Dunkelman, and M. Knežević, *KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers*, 2009.

[60] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, *Piccolo: An Ultra-Lightweight Blockcipher*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 342–357.

[61] M. O’Neill, “Low-cost SHA-1 hash function architecture for RFID tags,” *RFIDSec*, vol. 8, pp. 41–51, 2008.

[62] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, “SPONGENT: A lightweight hash function,” in *Cryptographic Hardware and Embedded Systems–CHES 2011*. Springer, 2011, pp. 312–325.

[63] J. Guo, T. Peyrin, and A. Poschmann, “The PHOTON family of lightweight hash functions,” in *Advances in Cryptology–CRYPTO 2011*. Springer, 2011, pp. 222–239.

[64] P. Pessl and M. Hutter, “Pushing the limits of SHA-3 hardware implementations to fit on RFID,” in *Cryptographic Hardware and Embedded Systems-CHES 2013*. Springer, 2013, pp. 126–141.

[65] B. Abdolmaleki, K. Baghery, B. Akhbari, and M. Aref, “Cryptanalysis of two EPC-based RFID security schemes,” in *International ISC Conference on Information Security and Cryptology – ISCISC 2015*. IEEE, September 2015, pp. 1–6.

[66] S. Abughazalah, K. Markantonakis, and K. Mayes, “A vulnerability in the Song authentication protocol for low-cost RFID tags,” in *Security and Privacy Protection in Information Processing Systems: 28th IFIP TC 11 International Conference, SEC 2013*. Springer, 2013, pp. 102–110.

[67] D. Dolev and A. Yao, “On the security of public key protocols”, in *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, IEEE, 1983.

[68] M. Akgun and U. Caglayan, "Weaknesses of two RFID protocols regarding de-synchronization attacks," in *International Wireless Communications and Mobile Computing Conference (IWCMC 2015).* Dubrovnik, Croatia: IEEE, August 2015.

[69] G. Avoine, "Adversarial model for radio frequency identification." *IACR Cryptology ePrint Archive*, vol. 2005, p. 49, 2005.

[70] S. Cai, T. Li, C. Ma, Y. Li, and R. Deng, "Enabling secure secret updating for unidirectional key distribution in RFID-enabled supply chains," in *Information and Communications Security*, ser. Lecture Notes in Computer Science, S. Qing, C. Mitchell, and G. Wang, Eds. Springer Berlin Heidelberg, 2009, vol. 5927, pp. 150–164. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11145-7_13

[71] K. Ouafi and R. Phan, "Traceable privacy of recent provably-secure RFID protocols," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, S. Bellovin, R. Gennaro, A. Keromytis, and M. Yung, Eds. Springer Berlin Heidelberg, 2008, vol. 5037, pp. 479–489. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68914-0_29

[72] P. Peris-Lopez, J. Hernandez-Castro, J. Tapiador, and J. Lubbe, "Cryptanalysis of an {EPC} Class-1 Generation-2 standard compliant authentication protocol," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 6, pp. 1061 – 1069, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197611000613

[73] P. Rizomiliotis, E. Rekleitis, and S. Gritzalis, "Security analysis of the Song-Mitchell authentication protocol for low-cost RFID tags," *Communications Letters, IEEE*, vol. 13, no. 4, pp. 274–276, April 2009.

[74] M. Safkhani, N. Bagheri, and M. Naderi, "Strengthening the security of EPC C-1 G-2 RFID standard," *Wireless Personal Communications*, vol. 72, no. 2, pp. 1295–1308, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11277-013-1078-z

[75] C. J. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification.* Springer, 2008, pp. 414–418.

[76] R. Patel, B. Borisaniya, A. Patel, D. Patel, M. Rajarajan, and A. Zisman, "Comparative analysis of formal model checking tools for security protocol verification," in *Recent Trends in Network Security and Applications.* Springer, 2010, pp. 152–163.

[77] S. Abughazalah, K. Markantonakis, and K. Mayes, "Secure improved cloud-based RFID authentication protocol," in *The 9th International Workshop on Data Privacy Management (DPM)*, J. Garcia-Alfaro, J. Herrera-Joancomartí, E. Lupu, J. Posegga, A. Aldini, F. Martinelli, and N. Suri, Eds. Springer Berlin Heidelberg, 2015, vol. 53, pp. 147–164. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17016-9_10

[78] M. Aiash, G. Mapp, A. Lasebae, R. Phan, and J. Loo, "A formally verified AKA protocol for vertical handover in heterogeneous environments using Casper/FDR," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, pp. 1–23, 2012.

[79] A. K. Ranjan, V. Kumar, and M. Hussain, "Security analysis of TLS authentication," in *Contemporary Computing and Informatics (IC3I), 2014 International Conference on.* IEEE, 2014, pp. 1356–1360.

[80] A. M. Taha, A. T. Abdel-Hamid, and S. Tahar, "Formal verification of IEEE 802.16 security sublayer using Scyther tool," in *Network and Service Security, 2009. N2S'09. International Conference on.* IEEE, 2009, pp. 1–5.

[81] S. Xu, C.-T. Huang, and M. M. Matthews, "Modeling and analysis of IEEE 802.16 PKM protocols using CasperFDR," in *Wireless Communication Systems. 2008. ISWCS'08. IEEE International Symposium on.* IEEE, 2008, pp. 653–657.

[82] A. Alshehri and S. Schneider, "Formally defining NFC M-coupon requirements, with a case study," in *The 8th International Conference for Internet Technology and Secured Transactions (ICITST).* IEEE, 2013, pp. 52–58.

[83] C. Cremers and S. Mauw, *Operational semantics and verification of security protocols.* Springer Science & Business Media, 2012.

[84] M. Aigner, T. Plos, and S. Coluccini, "Secure semi-passive RFID tags – prototype and analysis," Bridge Project, Tech. Rep., 2008.

[85] CAEN, *R1260I - Slate*, http://www.caenrfid.it/en/CaenProd.jsp?idmod=753&parent=73.

[86] *CrossWorks for AVR*, Online, Rowley Associates Ltd., http://www.rowley.co.uk/avr/.

[87] *AVR-Crypto-Lib*, Online, https://www.das-labor.org/wiki/AVR-Crypto-Lib/en.

[88] *DT830 Series 3 1/2 Digital Multimeter*, OEM & ODM manufacturer of test & measurement instruments. [Online]. Available: https://www.rapidonline.com/pdf/DT-830B_v1.pdf

[89] B. Song and C. J. Mitchell, "RFID authentication protocol for low-cost tags," in *Proceedings of the First ACM Conference on Wireless Network Security*, ser. WiSec '08. New York, NY, USA: ACM, 2008, pp. 140–147.

[90] M. R. Rieback, B. Crispo, and A. S. Tanenbaum, "RFID guardian: A battery-powered mobile device for rfid privacy management," in *Information Security and Privacy*. Springer, 2005, pp. 184–194.

[91] A. Al-Adhami, M. Ambroze, I. Stengel, M. Tomlinson, "A Quorum RFID System Using Threshold Cryptosystem," in *IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2016, pp. 107–113.

[92] P. Bonnefoi, P. Dusart, D. Sauveron, R. Akram, K. Markantonakis, "A Set of Efficient Privacy Protection Enforcing Lightweight Authentication Protocols for Low-Cost RFID Tags," in *Trustcom/BigDataSE/ISPA*, IEEE, 2015, pp. 612–620.

[93] S. A. Weis, "Security and privacy in radio-frequency identification devices," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[94] S. Weis, S. Sarma, R. Rivest, and D. Engels, "Security and privacy aspects of low-cost radio frequency identification systems," in *Security in pervasive computing.* Springer, 2004, pp. 201–212.

[95] H. Chien and C. Chen, "Mutual authentication protocol for RFID conforming to EPC Class 1 Generation 2 Standards," *Computer Standards Interfaces*, vol. 29, no. 2, pp. 254–259, 2007.

[96] M. Ohkubo, K. Suzuki, S. Kinoshita *et al.*, "Cryptographic approach to 2̣privacy-friendly3̧ tags," in *RFID privacy workshop*, vol. 82. Cambridge, USA, 2003.

[97] G. Avoine, E. Dysli, P. Oechslin *et al.*, "Reducing time complexity in RFID systems," in *Selected Areas in Cryptography*, vol. 3897. Springer, 2005, pp. 291–306.

[98] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda, "Cryptanalysis of a novel authentication protocol conforming to EPC-C1G2 standard," *Computer Standards & Interfaces*, vol. 31, no. 2, pp. 372–380, 2009.

[99] X. Yi, L. Wang, D. Mao, and Y. Zhan, “An Gen2 based security authentication protocol for RFID system,” *Physics Procedia*, vol. 24, pp. 1385–1391, 2012.

[100] J. Korsh, *Data Structures, Algorithms and Program Style*, PWS Publishing Co., 1986.

[101] M. Safkhani, N. Bagheri, P. Peris-Lopez, A. Mitrokotsa, and J. C. Hernandez-Castro, “Weaknesses in another Gen2-based RFID authentication protocol,” in *RFID-Technologies and Applications (RFID-TA), 2012 IEEE International Conference on*, Nov 2012, pp. 80–84.

[102] M. A. B. Shemaili, C. Y. Yeun, and M. J. Zemerly, “RFID lightweight mutual authentication using shrinking generator,” in *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*. IEEE, 2009, pp. 1–6.

[103] Y. HanataniI, M. Ohkubo, S. Matsuo, K. Sakiyama, K. Ohta, “A Study on Computational Formal Verification for Practical Cryptographic Protocol: The Case of Synchronous RFID Authentication”, in *Financial Cryptography and Data Security: FC 2011 Workshops*, Springer Berlin Heidelberg, 2012, pp. 70–87.

[104] G. Lowe, “Casper: A compiler for the analysis of security protocols”, in *IEEE 10th Proceedings on Computer Security Foundations Workshop*, IEEE, 1997, pp. 18–30.

[105] T.-C. Yeh, Y.-J. Wang, T.-C. Kuo, and S.-S. Wang, “Securing RFID systems conforming to EPC Class 1 Generation 2 standard,” *Expert Systems with Applications*, vol. 37, no. 12, pp. 7678–7683, 2010.

[106] A. Juels, “Yoking-Proofs for RFID tags”, in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, IEEE, 2004, pp. 138–143.

[107] A. Juels, R. Pappu, B. Parno, “Unidirectional key distribution across time and space with applications to RFID security”, in *Proceedings of the 17th conference on Security symposium*, USENIX Association, 2008, pp. 75–90.

[108] M. Langheinrich, R. Marti, “Practical minimalist cryptography for RFID privacy”, in *IEEE Systems Journal*, vol. 1, no. 2, pp. 115–128, 2007.

[109] R. McEliece, D. Sarwate, “On Sharing Secrets and Reed-Solomon Codes”, in *ACM Commun.*, vol. 24, no. 9, pp. 583–584, 1981.

[110] T. Dimitriou, "A Lightweight RFID Protocol to protect against Traceability and Cloning attacks", in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks, (SecureComm 2005)*, Springer, 2005, pp. 59–66.

[111] K. Finkenzeller, "RFID Handbook: Radio-frequency identification fundamentals and applications", Wiley, 1999.

[112] J. Hernandez-Castro, P. Peris-Lopez, M. Safkhani, N. Bagheri, M. Naderi, "Another Fallen Hash-Based RFID Authentication Protocol", in *The 6th IFIP WG 11.2 International Workshop, Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems, (WISTP 2012)*, Springer, 2012, pp. 29–37.

[113] S. Cai, Y. Li, T. Li, R. Deng, "Attacks and improvements to an RFID mutual authentication protocol and its extensions", in *Proceedings of the second ACM conference on Wireless network security*, ACM, 2009, pp. 51–58.

[114] D. Duc, J. Park, H. Lee, K. Kim, "Enhancing security of EPCglobal Gen-2 RFID tag against traceability and cloning", in *CS-Conference Papeers*, Institute of Electronics, Information and Communication Engineers, 2006.

[115] D. Molnar, D. Wagner, "Privacy and security in library RFID: Issues, practices, and architectures", in *Proceedings of the 11th ACM conference on Computer and communications security*, ACM, 2004, pp. 210–219.

[116] C. Ma, J. Lin, Y. Wang, M. Shang, "Offline RFID grouping proofs with trusted timestamps", in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2012, pp. 674–681.

[117] E. Yoon, "Improvement of the securing RFID systems conforming to EPC Class 1 Generation 2 Standard," *Expert Systems with Applications*, vol. 39, no. 1, pp. 1589–1594, 2012.

[118] M. Safkhani, N. Bagheri, S. K. Sanadhya, and M. Naderi, "Cryptanalysis of improved Yeh et al.'s authentication protocol: An EPC Class-1 Generation-2 standard compliant protocol," *IACR Cryptology ePrint Archive*, vol. 2011, p. 426, 2011.

[119] G. Avoine and X. Carpent, "Yet another ultralightweight authentication protocol that is broken," in *Cryptology ePrint Archive, Report 2011/691. Also, in Workshop on RFID Security RFIDSec12, Nijmegen, Netherlands, June 2012*, 2012.

[120] G. Avoine, X. Carpent, and J. Hernandez-Castro, “Pitfalls in ultralightweight authentication protocol designs,” *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, 2015.

[121] 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 *Security in pervasive computing.* Springer, 2004, pp. 201–212.

[122] G. Poulopoulos, K. Markantonakis, and K. Mayes, “A Secure and Efficient Mutual Authentication Protocol for Low-Cost RFID Systems,” in *International Conference on Availability, Reliability and Security (ARES’09).* IEEE, 2009, pp. 706–711.

[123] J. Shi, S. M. Kywe, and Y. Li, “Batch clone detection in RFID-enabled supply chain,” in *IEEE International Conference on RFID (IEEE RFID)*, April 2014, pp. 118–125.

[124] P. Lopez, J. Hernandez-Castro, J. Estevez-Tapiador, A. Ribagorda, “Solving the simultaneous scanning problem anonymously: clumping proofs for RFID tags”, in *Third International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SECPerU 2007)*, IEEE, 2007, pp. 55-60.

[125] K. Toyoda and I. Sasase, “Secret sharing based unidirectional key distribution with dummy tags in Gen2v2 RFID-enabled supply chains,” in *IEEE International Conference on RFID (IEEE RFID)*, April 2015, pp. 63–69.

[126] S. Abughazalah, K. Markantonakis, and K. Mayes, “Enhancing the key distribution model in the RFID-enabled supply chains,” in *The 28th International Conference onAdvanced Information Networking and Applications Workshops (WAINA)*, May 2014, pp. 871–878.

[127] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[128] C. Tan, B. Sheng, and Q. Li, “Secure and serverless RFID authentication and search protocols,” *Wireless Communications, IEEE Transactions on*, vol. 7, no. 4, pp. 1400–1407, 2008.

[129] J. Landt, “The history of RFID”, in *IEEE Potentials*, vol. 24, no. 4, pp. 8–11, 2005.

[130] W. Xie, L. Xie, C. Zhang, Q. Zhang, and C. Tang, "Cloud-based RFID authentication," in *RFID (RFID), 2013 IEEE International Conference on.* IEEE, 2013, pp. 168–175.

[131] Z.-W. Yuan and Q. Li, "Research on Data Processing of RFID Middleware Based on Cloud Computing", in *5th International Conference on Rough Set and Knowledge Technology (RSKT 2010).* Springer, 2010, pp. 663–671. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16248-0_90

[132] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 843–859, Second 2013.

[133] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating timing channels in compute clouds," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop.* ACM, 2010, pp. 103–108.

[134] B. D. Payne, M. De Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual.* IEEE, 2007, pp. 385–397.

[135] C. Lee, H. Chien, C. Laih, "Server-less RFID authentication and searching protocol with enhanced security", in *International Journal of Communication Systems*, vol. 25, no. 3, pp. 376–385, 2012.

[136] J. Li, Y. Wang, B. Jiao, Y. Xu, "An authentication protocol for secure and efficient RFID communication", in *International Conference on Logistics Systems and Intelligent Management*, IEEE, 2010, pp. 1648–1651.

[137] T. Li, Y. Li, G. Wang, "Secure and Practical Key Distribution for RFID-Enabled Supply Chains", in *Security and Privacy in Communication Networks*, Springer, 2012, vol. 96, pp. 356–372.

[138] P. Mell, T. Grance, "The NIST definition of cloud computing", in *NIST Special Publication*, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

[139] J. Liu, Y. Xiao, S. Li, W. Liang, C. Chen, "Cyber security and privacy issues in smart grids", in *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 981–997, 2012.

[140] X. Zhifeng, X. Yang, "Security and Privacy in Cloud Computing", in *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[141] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security.* ACM, 2009, pp. 199–212.

[142] S. Sundaresan, R. Doss, S. Piramuthu, and W. Zhou, "A robust grouping proof protocol for RFID EPC C1G2 tags," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 6, pp. 961–975, June 2014.

[143] D. Moriyama, "A provably secure offline RFID yoking-proof protocol with anonymity," in *Lightweight Cryptography for Security and Privacy.* Springer, 2015, pp. 155–167.

[144] J. Saito and K. Sakurai, "Grouping proof for RFID tags," in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 2. IEEE, 2005, pp. 621–624.

[145] S. Piramuthu, "On existence proofs for multiple RFID tags," in *Pervasive Services, 2006 ACS/IEEE International Conference on.* IEEE, 2006, pp. 317–320.

[146] P. Peris-Lopez, A. Orfila, J. C. Hernandez-Castro, and J. C. Van der Lubbe, "Flaws on RFID grouping-proofs. Guidelines for future sound protocols," *Journal of Network and Computer Applications*, vol. 34, no. 3, pp. 833–845, 2011.

[147] H.-Y. Chien, C.-C. Yang, T.-C. Wu, and C.-F. Lee, "Two RFID-based solutions to enhance inpatient medication safety," *Journal of Medical Systems*, vol. 35, no. 3, pp. 369–375, 2011.

[148] L. Bolotnyy and G. Robins, "Generalized Yoking-Proofs for a group of RFID tags," in *The Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services.* IEEE, 2006, pp. 1–4.

[149] M. Burmester, B. de Medeiros, and R. Motta, *Provably Secure Grouping-Proofs for RFID Tags.* Springer Berlin Heidelberg, 2008, pp. 176–190.

[150] S. Abughazalah, K. Markantonakis, and K. Mayes, "A formally verified mutual authentication protocol for low-cost RFID tags," *International Journal of RFID Security and Cryptography*, vol. 3, no. 2, pp. 156–169, 2014.

[151] S. Abughazalah, K. Markantonakis, K. Mayes, "Two rounds RFID grouping-proof protocol," in *The 10th IEEE Radio Frequency Identification International conference (IEEE RFID)*, IEEE, 2016, pp. 1–14.

[152] S. Abughazalah, K. Markantonakis, K. Mayes, "Secure mobile payment on NFC-enabled mobile phones formally analysed using CasperFDR," in *IEEE 13th International Conference onTrust, Security and Privacy in Computing and Communications (TrustCom)*, Sept 2014, pp. 422–431.

[153] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on.* IEEE, 2001, pp. 136–145.

[154] D. W. Nance, T. L. Naps, *Introduction to Computer Science: Programming, Problem Solving and Data Structures.* West Publishing Co., 1992.

[155] R. Anderson and M. Bond, "The man-in-the-middle defence," in *Security Protocols*, ser. Lecture Notes in Computer Science, B. Christianson, B. Crispo, J. Malcolm, and M. Roe, Eds. Springer Berlin Heidelberg, 2009, vol. 5087, pp. 157–163. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04904-0_21

[156] A. Arbit, Y. Oren, and A. Wool, "Toward practical public key anti-counterfeiting for low-cost EPC tags," in *RFID (RFID), 2011 IEEE International Conference on.* IEEE, 2011, pp. 184–191.

[157] E. B. Kavun and T. Yalcin, "A lightweight implementation of Keccak hash function for radio-frequency identification applications," in *Radio frequency identification: security and privacy issues.* Springer, 2010, pp. 258–269.

[158] K. Chiew, Y. Li, T. Li, R. H. Deng, and M. Aigner, "Time cost evaluation for executing RFID authentication protocols," in *Internet of Things (IOT), 2010.* IEEE, 2010, pp. 1–8.

[159] C. Dabas and J. Gupta, "A cloud computing architecture framework for scalable RFID," in *International Multi-Conference of Engineering and Computer Scientists*, vol. 1. Citeseer, 2010.

[160] *Identification Cards - Contactless Integrated Circuit Cards*, Online, International Standard Organization / International Electrotechnical Commission Std Std., December 2006, http://www.iso.org/iso/catalogue_detail.htm?csnumber=39695.

[161] "EPC radio frequency identification protocols Class 1 Generation 2 UHF RFID," Online, January 2005, http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/uhfc1g2_1_0_9-standard-20050126.pdf.

Appendix A

Mutual Authentication Protocol for Low-Cost RFID Tags Formal Analysis Scripts

A.1 CasperFDR Script

```
-- Remove the reader entity to avoid state space explosion

#Free variables
T : Agent
S : Server
R1, R2 : Nonce
ID, K : Data
h: HashFunction
InverseKeys= (h,h)
#Protocol description
0. -> S : T
1. S -> T : R1
2a. T -> S : R2,h(R2 (+) ID)
2b. T -> S : h(K, R1, R2)
4. S -> T : h(ID, K, R1, R2)
#Processes
RESPONDER(T, S, R2, K, ID)
SERVER (S, T, R1, K, ID)
#Actual variables
```

```
Tag, Mallory : Agent
ServerDB : Server
Rr1, Rr2, R3 : Nonce
IDentityT, KeyTag : Data
#Specification
Aliveness(S, T)
Secret(T, K, [S])
Secret(T, ID, [S])
Agreement(T, S, [ID,K])
#System
RESPONDER(Tag,ServerDB,Rr2,KeyTag,IDentityT)
SERVER(ServerDB, Tag, Rr1, KeyTag, IDentityT)
#Intruder Information
Intruder = Mallory
IntruderKnowledge ={Tag,ServerDB,Mallory,Rr1, Rr2, R3}
```

A.2 Scyther Script

```
usertype Data;
hashfunction H1;
const XOR: Function;

protocol MutualAuth (S,  R,  Ti){

role S{
fresh IDi: Data;
fresh Ki: Data;
var R1, R2: Nonce;

recv_3(R , S, R2, H1(Ki, R1, R2), H1(XOR(IDi, R2)));
send_4(S, R,  H1(IDi, Ki, R1, R2));

claim_S1(S, Secret, IDi);
claim_S2(S, Secret, Ki);
claim_S3(S, Niagree);
claim_S4(S, Alive);
}
role R{
fresh R1:  Nonce;
var X: Ticket;
var Y: Ticket;
var Z: Ticket;
var R2: Nonce;

send_1(R, Ti, R1);
recv_2(Ti, R, R2, X, Y);
send_3(R, S, R2, X, Y);
recv_4(S, R , Z);
send_5(R, Ti, Z);

}

role Ti{
var R1: Nonce;
fresh R2: Nonce;
fresh IDi: Data;
fresh Ki: Data;

recv_1(R, Ti, R1);
send_2(Ti, R, R2, H1(Ki, R1, R2), H1(XOR(IDi, R2)));
recv_5(R, Ti, H1(IDi, Ki, R1, R2));

claim_Ti1(Ti, Secret, IDi);
claim_Ti2(Ti, Secret, Ki);
claim_Ti3(Ti, Alive);
claim_Ti4(Ti, Niagree);
claim_Ti5(Ti, Nisynch);
}
}
```

Appendix B

Key Update Process in the Key Distribution Model Formal Analysis Scripts

B.1 CasperFDR Script

```
-- Secret key update protocol in CasperFDR
-- We assumed that the new values of (S, Tag-ID and C) are generated
by the reader
-- We remove the reader entity

#Free variables
S, Ti: Agent
R1, R2: Nonce
ID, Ci, IDProductnew, Snew, Cnew: Data
h: HashFunction
InverseKeys = (h,h)

#Protocol description
1. S    -> Ti : R1
2. Ti   -> S : R2
3b. S   -> Ti : IDProductnew, h(ID (+) IDProductnew (+) R1 (+) R2 (+)
Ci) (+) Snew
3c. S   -> Ti : h(Snew (+) ID (+) R1 (+) R2) (+) Cnew
```

```
3d. S   -> Ti : h(Snew (+) ID (+) R1 (+) R2 (+) Cnew)
4.  Ti  -> S : h(ID (+) Snew (+) Cnew (+) R1 (+) R2)

#Processes
Serveri(S, Ti, R1, ID, IDProductnew, Ci, Snew, Cnew)
Tagii(Ti, S, R2,  ID, Ci)

#Actual variables
Serveri, Tagi, Mallory: Agent
R11, R22, R33: Nonce
IDi, CTag, IDProductnew, SNew, CNew: Data

#Specification
Agreement (S,  Ti, [ID])
Secret (S, ID, [Ti])
Secret (S, Snew, [Ti])
Secret (S, Cnew, [Ti])

#System
Serveri(Serveri, Tagi, R11, IDi, IDProductnew, CTag, SNew, CNew)
Tagii(Tagi, Serveri,  R22,  IDi, CTag)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Serveri, Tagi, Mallory, R11, R22, R33,
CTag,  IDProductnew}
```

B.2 Syther Script

```
-- Secret key update protocol in Scyther
-- We assumed that the S, ID-case and C values are already generated
by the reader

usertype Data;
hashfunction H1;
const XOR: Function;
```

```
protocol key-update (S,  Ti){

 role S{

fresh ID: Data;
fresh Ci: Data;
fresh IDProductnew: Data;
fresh Snew: Data;
fresh Cnew: Data;
fresh R1 : Nonce;
var R2  : Nonce;

send_1(S, Ti, R1);
recv_2(Ti, S, R2);
send_3(S, Ti, IDProductnew,XOR(H1(ID, IDProductnew, R1, R2, Ci),Snew),
XOR(H1(Snew, ID, R1, R2) ,Cnew), H1(XOR(Snew, ID, R1, R2, Cnew)));
recv_2(Ti, S, H1(XOR(ID, Snew, Cnew, R1, R2)));


claim_S1(S, Secret, ID);
claim_S2(S, Secret, Snew);
claim_S3(S, Secret, Cnew);
claim_S4(S, Alive);
claim_S5(S, Niagree);
claim_S6(S, Nisynch);
}

 role Ti{

fresh R2: Nonce;
var R1: Nonce;
fresh ID: Data;
fresh Ci: Data;
var Snew: Data;
var Cnew: Data;
var IDProductnew: Data;
recv_1(S, Ti, R1);
send_2(Ti, S, R2);
recv_3(S, Ti, IDProductnew,XOR(H1(ID, IDProductnew, R1, R2, Ci),Snew),
XOR(H1(Snew, ID, R1, R2) ,Cnew),  H1(XOR(Snew, ID, R1, R2, Cnew)));
send_2(Ti, S, H1(XOR(ID, Snew, Cnew, R1, R2)));


claim_Ti1(Ti, Secret, ID);
claim_Ti2(Ti, Secret, Snew);
claim_Ti3(Ti, Secret, Cnew);
claim_Ti4(Ti, Alive);
claim_Ti5(Ti, Niagree);
claim_Ti6(Ti, Nisynch);
}
}
```

Appendix C

Improved Cloud-Based RFID Protocol Formal Analysis Scripts

C.1 CasperFDR Script (Xie protocol)

```
--The cloud-based RFID authentication  protocol
--We assume that the reader already knows the tag's data

#Free variables
T, R : Agent
Nr: Nonce
Nt: nonce
TID, RID, S, M: Data
h: HashFunction
InverseKeys=(h,h)

#Protocol description
0.    -> T : R
1.  T -> R : h(RID, TID, S)
2.  R -> T : Nr
3.  T -> R : h(RID, TID, Nr), Nt
4a.  R -> T : h(RID, TID, Nt) (+) M
4b.  R -> T : h(TID, RID, M)

#Processes
TAGG(T, R, TID, RID, S, Nt)
```

```
Reader(R,  T, TID, RID, S, M, Nr)

#Actual variables
T1, R1, Mallory : Agent
Nr1: Nonce
Nt1, NM: nonce
TID1, RID1, S1, M1: Data

#Specification
Agreement(T, R, [TID, RID])
Secret(R, M, [T])
Secret (T, S, [R])
Secret (T, TID, [R])
Secret(T, RID, [R])

#System
TAGG(T1, R1, TID1, RID1, S1, Nt1)
Reader(R1,  T1, TID1, RID1, S1, M1, Nr1)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {T1, R1, Mallory, Nr1, Nt1, NM}
```

C.2 CasperFDR Script (Our protocol)

```
--The improved cloud-based protocol in CasperFDR
--We assume that the reader already knows the tag's data

#Free variables
T, R : Agent
R1: Nonce
R2: nonce
ID, K: Data
h: HashFunction
InverseKeys=(h,h)
```

```
#Protocol description
0.       -> R : T
1.    R -> T : R1
2a.   T -> R : R2, h(ID (+) R1 (+) R2)
2b.   T -> R : h(ID, K, R1, R2), R2
4.    R -> T : h(h(ID), h(h(ID) (+) K), R1, R2)

#Processes
TAGG(T, R, ID, R2, K)
Reader(R, T, ID, R1, K)

#Actual variables
T1, R11, Mallory : Agent
Nr1, NM1: Nonce
Nt1: nonce
ID1, K1: Data

#Specification
Agreement(T, R, [ID, K])
Secret(T, ID, [R])
Secret (T, K, [R])

#System
TAGG(T1, R11, ID1, Nt1, K1)
Reader(R11, T1, ID1, Nr1, K1)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {T1, R11, Nr1, Nt1, NM1, Mallory}
```

C.3 Scyther Script

```
-- Improved cloud-based protocol in Scyther

usertype Data;
hashfunction H1;
```

```
const XOR: Function;

protocol cloud-RFID (R,  Ti){


role R{

fresh R1 : Nonce;
var R2  : Nonce;
fresh Ki: Data;
fresh IDi: Data;




send_1(R, Ti, R1);
recv_2(Ti, R, H1(XOR(IDi, R1, R2)), H1(IDi, Ki, R1, R2), R2);
send_3(R, Ti, H1(H1(XOR(H1(IDi), Ki)), R1, R2));

claim_R1(R, Secret, IDi);
claim_R2(R, Secret, Ki);
claim_R3(R, Alive);
claim_R4(R, Niagree);
claim_R5(R, Nisynch);
}



role Ti{

fresh R2: Nonce;
var R1: Nonce;
fresh IDi: Data;
fresh Ki: Data;


recv_1(R, Ti, R1);
send_2(Ti, R, H1(XOR(IDi, R1, R2)), H1(IDi, Ki, R1, R2), R2);
recv_3(R, Ti, H1(H1(XOR(H1(IDi), Ki)), R1, R2));

claim_Ti1(Ti, Secret, IDi);
claim_Ti2(Ti, Secret, Ki);
claim_Ti3(Ti, Alive);
claim_Ti4(Ti, Niagree);
claim_Ti5(Ti, Nisynch);
}
}
```

Appendix D

RFID Grouping-Proof Formal Analysis Scripts

D.1 CasperFDR Script

```
-- Group Proof PROTOCOL in CasperFDR
-- We omitted some data for simplicity and reducing
memory space in the compiling process


#Free variables
R, Ti: Agent
S: Server
rR, ri: Nonce
IDG, IDi, SK: Data
h: HashFunction
KRS: Key
t: TimeStamp
InverseKeys = (h,h), (KRS, KRS)

#Protocol description
1. S -> R: t, SK, h(IDi)
2. R    -> Ti : t, rR, h(SK, rR)
3a. Ti  -> R : ri , h(SK, ri, rR, h(IDi))
3b. Ti  -> R: h(IDi, ri, rR, t) % Mi
5. R    -> S : {ri, rR, t, SK, Mi % h(IDi, ri, rR, t)}{KRS}
```

```
#Processes
Reader1(R, Ti, S, rR, KRS)
Tagii(Ti, R, S, ri,  IDG, IDi)
ServerDB(S, R, Ti, IDi, IDG, SK, KRS)

#Actual variables
Reader, Tagi, Mallory: Agent
Server1: Server
Rr, Ri: Nonce
IDGroup, IDTag1, SK1: Data
KRSkey: Key
InverseKeys = (KRSkey, KRSkey)


#Specification
Agreement (R,  Ti, [SK])
Secret (S, IDi, [Ti])
Secret (R, SK, [Ti])

#System
Reader1(Reader, Tagi, Server1, Rr, KRSkey)
Tagii(Tagi, Reader, Server1, Ri,  IDGroup, IDTag1)
ServerDB(Server1, Reader, Tagi, IDTag1, IDGroup, SK1,KRSkey)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Server1, Reader, Tagi, Mallory, Rr, Ri}
```

D.2 Syther Script

```
--RFID Grouping-proof protocol in Scyther

usertype Timestamp;
usertype Data;
```

```
secret k: Function;
hashfunction H1;

protocol GroupProof (S,  R,  Ti){

role S{
fresh Time: Timestamp;
fresh IDi: Data;
fresh TSi: Data;
fresh IDG: Data;
fresh TSG: Data;
fresh SK: Data;
var ri, rR: Nonce;

send_!T1(S, S, Time);
send_1(S, R, Time);
recv_6(R, S, {rR, ri, Time, SK, H1(IDi, ri, rR, TSi,
Time)}k(R, S));

claim_S3(S, Secret, IDi);
claim_S4(S, Secret, TSi);
claim_S4(S, Secret, SK);
claim_S5(S, Niagree);
}


role R{

var Time: Timestamp;
var  X: Ticket;
fresh IDG: Data;
fresh IDi: Data;
fresh TSG: Data;
fresh rR : Nonce;
var ri  : Nonce;
var SK: Data;
```

```
recv_1(S, R, Time);
send_2(R, Ti, Time, rR, H(SK, rR);
recv_3(Ti, R, Ti, R, ri,  H1(SK, TSG, ri, rR, H(IDi)),
, X);
send_6(R, S, {rR, ri, Time, SK, X}k(R, S));

claim_R1(R, Secret, SK);
claim_R2(R, Niagree);
claim_R3(R, Nisynch);
}



role Ti{
var Time: Timestamp;
fresh ri: Nonce;
var rR: Nonce;
fresh TSG: Data;
fresh IDG: Data;
fresh IDi: Data;
fresh TSi: Data;
var SK: Data;


recv_2(R, Ti, Time, rR, H1(SK,rR);
send_3(Ti, R, ri,  H1(SK, TSG, ri, rR, H(IDi)),
H1(IDi, ri, rR, TSi, Time));

claim_Ti1(Ti, Secret, IDG);
claim_Ti2(Ti, Secret, TSG);
claim_Ti3(Ti, Secret, IDi);
claim_Ti4(Ti, Secret, TSi);
claim_Ti6(Ti, Alive);
claim_Ti7(Ti, Niagree);
claim_Ti8(Ti, Nisynch);
}}
```