A DTN-Based RFID Protocol

by

Xin Deng

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for
Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario, Canada
Fall 2012

Your file Votre référence
ISBN: 978-0-494-94277-2

Our file Notre référence
ISBN: 978-0-494-94277-2

Abstract

In this thesis, we propose a lightweight Radio-frequency identification ($RFID$) mutual authentication protocol without permanent backend server connections. Only pseudo-random number generator ($PRNG$) and XOR operations are used in our protocol. Mutual authentication is achieved by exchanging random numbers drawn from the synchronized $PRNG$ functions of the readers and tags. A Delay Tolerant Network (DTN) is implemented by readers to communicate and synchronize their state. The performance is evaluated using the ONE simulator.

Acknowledgments

I would like to give special thanks to Michel Barbeau, professor at Carleton University School of Computer Science, for his guidance and help.

Table of Contents

List of Tables

List of Figures

Glossary

ACK	Acknowledgment
CRC	Cyclic Redundancy Check
EPC	Electronic Product Code
EPC C1G2	EPC Class 1 Gen2 Standard
EPC HF C1G2	EPC High Frequency Class 1 Generation 2 Standard
EPC UHF C1G2	EPC Ultra High Frequency Class 1 Generation 2 Standard
HF	High Frequency
ID	Identity
IDS	Index-pseudonym Number
LF	Low Frequency
MDID	Tag Mask Designer Identifier
NAK	Negative Acknowledgment
ONE	Opportunistic Network Environment
PRNG	Pseudo-random Number Generator

RD	Reader Identity
RFID	Radio-frequency Identification
RN16	16-bit Random Number
RND16	16-bit Random Number
RND	Random Number
seed_{cur}	Current *PRNG* Seed
seed_{pre}	Previous *PRNG* Seed
seed_{temp}	Temporary *PRNG* Seed
TD	Tag Identity
TDS	Tag Pseudorandom Identity Number
TDS_{cur}	Current Tag Pseudorandom Identity Number
TDS_{pre}	Previous Tag Pseudorandom Identity Number
TID	Tag Identification
TTL	Message Time To Live
UHF	Ultra High Frequency
URI	Uniform Resource Identifier
XTID	Extended Tag Identification

Chapter 1

Introduction

1.1 Background

1.1.1 Radio-frequency identification (RFID) technology

The RFID (Radio-Frequency Identification) technology is widely used in daily transactions. It is slowly replacing the use of barcodes [4]. The implementation of a RFID tag consists of a processor, memory and a transceiver. It can be powered by a reader or use its own batteries. Each RFID tag has an Electronic Product Code (*EPC*). The *EPC* can be extracted by a portable reader unit.

Currently, there are three radio spectrum segments utilized by RFID: the Low Frequency (*LF*) range (124 to 135 KHz), the High Frequency (*HF*) range (13.56 MHz) and the Ultra High Frequency (*UHF*) range (868 MHz in Europe, 915 MHz in North America or 950 MHz in Japan) [2]. Chien classifies RFID tags into three types, based on supported operations: *fullfledged*, *lightweight* and *ultralight* [7]. *Fullfledged* tags are capable of conducting conventional cryptography. They support random number generators and hash functions. *Fullfledged* tags are high-cost. It means that they require more computation and memory resources to function. *Lightweight* tags implement random number generators and Cyclic Redundancy Code (*CRC*) checksum calculation. *Ultralight* tags only support

bitwise calculation operations like *XOR*, *AND* and *OR*. *Lightweight* tags and *ultralight* tags are low-cost. It means that they require less computation and memory resources. It is a big challenge to secure low-cost tags due to their restricted capabilities and widely distributed use in daily activities.

The *EPCglobal* standard [15] promotes the implementation of the RFID technology. It supports *lightweight* tags. It specifies an on-chip 16-bit Pseudo Random Number Generator (*PRNG*) and a 16-bit Cyclic Redundancy Code calculator. *EPCglobal* [10] has defined a detailed standard architecture for system interoperability. This architecture has three parts: *EPC* Data Exchange standards, Data Exchange Infrastructure for Data Capture and Physical Object Exchange standards. Recently, the *EPC* Physical Object Exchange standard has been heavily studied due to the lack of a mutual verification scheme in the inventory protocol. The *EPC* Physical Object Exchange standard enables participant mutual identification. They define two interface standards: *EPC UHF* tag protocol [11] and *EPC HF* protocol [12]. They also define two data standards: tag data standard [13] and tag data translation standard [14]. The interface standards define detail commands that a reader can use to connect with tags. The tag data standards specify tag memory deployment and tag functionalities (e.g., *PRNG*). In this thesis, modifications are made to the inventory protocol in accordance with the requirements from the tag data standard and tag data translation standard.

1.2 Problem description

The inventory protocol is part of the interface standards. It specifies the start of a communication between a reader and a tag and helps the reader to identify a tag. Because of the absence of mutual verification in the inventory protocol, tags can be read without reader identity verification. The inventory protocol is presented in Figure 1.

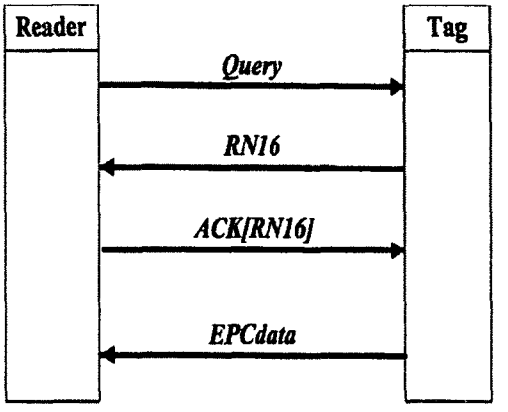

Figure 1: Inventory protocol.

Within the inventory protocol, a reader sends a query to a tag. The tag replies with a 16-bit random number: $RN16$. A tag accepts a reader when it receives the acknowledgement of the $RN16$ from the reader. It sends back its identification $EPCdata$ back to the reader. Thus, as long as a reader is capable of sending a query and an acknowledgement of a random number, it can read any tag. Moreover, as long as a tag is equipped with a 16-bit random number generator, and can receive the query command, it can start a communication with a reader. On the other hand, adversaries can read and write a reader or forge a tag's identity and provide information to a reader on behalf the real tag. There is no mutual verification in the inventory protocol. If the tag is used as a chip on a credit card, then important personal information can be leaked to the adversary. Adversaries can also use a fake chip to complete transactions on behalf of a real credit card. Mutual verification is needed for the inventory protocol.

1.3 Contributions

Several new protocols have been proposed to solve the lack of mutual authentication in the *EPCglobal* inventory protocol defined in [11] and [12]. In Chapter 3, we propose a new

mutual verification protocol - the DTN based RFID protocol. In the DTN based RFID protocol, readers and tags keep a synchronized *PRNG* state, which includes the same *PRNG* function and same *PRNG* seed for mutual verification. Each tag is assigned a pseudonym number. It is used to communicate with the reader instead of using the its real identity. Both the *PRNG* seed and tag's pseudonym number are updated after every successful inventory. The DTN based RFID protocol enables RFID tags and readers to verify each other. It helps RFID tags to hide their own real identity. It achieves untraceability, mutual verification and forward and backward security for lightweight RFID tags. If there is more than one reader working with a set of tags, reader states need to be synchronized using a permanent connection with a backend server or a network. We study two scenarios where the RFID technology is combined with Delay Tolerant Networks (*DTN*s) to avoid permanent connections with backend servers. All our works follow the requirement of the *EPCglobal* interface standard. The DTN based RFID protocol is fully compatible with the current *EPCglobal* standard.

1.4 Document structure

In this thesis, previous research is reviewed and summarized. The DTN based RFID protocol and two related strategies are introduced. The security of the protocol and its compatibility with the *EPCglobal* standards are discussed. A simulation is conducted to evaluate RFID readers' performance in the DTN. The following is the structure of this thesis.

Chapter 2, Recent RFID security studies. This chapter discusses three previous research works to enhance the *EPCglobal* inventory protocol security.

Chapter 3, The DTN based RFID protocol. This chapter presents our proposed protocol and strategies for handling a de-synchronization situation and a collision situation.

Chapter 4, Analysis and delay tolerant network. This chapter discusses the security of the proposed protocol and its compatibility with the *EPCglobal* standard. At the end of this chapter, the concept of *DTN* and the strategy RFID readers use to communicate in a DTN are introduced.

Chapter 5, Simulation. Simulation is conducted in this chapter to evaluate RFID readers performance in two scenarios.

Chapter 6, Conclusion. This chapter summarizes the overall work and compares it with two previous proposed protocols.

Chapter 2

Recent RFID security studies

EPCglobal standards aim at improving the RFID tag operationalability and balancing its cost and functionality. Basic reliability is guaranteed by the use of *PRNG* and *CRC* functions. Based on their standards, a reader needs to conduct three different protocol sessions to communicate with a tag: *select*, *inventory*, *access*. In the *select* session, reader selects a group of tags. The *inventory* session helps the reader to identify one tag from the selected tag group. The reader starts to update the chosen tag in the *access* session. In the *inventory* session, the RFID tag sends its identification to the reader for further communications. No mutual verification in the *inventory* session leads to the exposure of a tag identity. It is vulnerable to tag identity theft and reader impersonation.

Studies on the *EPCglobal* inventory protocol can be roughly divided into two categories. In the first category, new protocols use private and public keys with hash functions for mutual verification. Theoretically, they prove that hash functions can secure the *EPCglobal* inventory protocol. But, low-cost RFID tags do not support hash functions. The second category of studies focuses on using the current *EPCglobal* standard to design lightweight RFID protocols. Among the second category, there are two branches of popular protocols. One is the family of ultralight weight protocols: the first is M^2AP [26], followed by *EMAP* [24] and *LMAP* [25]. This branch of protocols do not expose any tag or reader information in the environment. They completely follow the *EPCglobal* standard

by using bitwise operations: *OR*, *AND* and *XOR*. The reader identifies tags by using tag pseudonym numbers instead of their own real identities. In 2009, authors of those protocols proposed a new one called Gossamer [28]. This new ultralight protocol has some ideas similar with the ones of the *SASI* protocol [7]. They both store the previous inventory verification information to recover from de-synchronization between a reader and a tag.

2.1 The Gossamer protocol

There are three groups of participants in the Gossamer Protocol [28]: the server, readers and tags. Readers are used to communicate with tags while the server supports all calculations and verifications. Instead of using a tag's real identity, an index-pseudonym number *IDS* is used for each tag. The *IDS* is updated after every successful interrogation. The tag real identification is kept the same. Tags and the server shares two private keys: k_1, k_2 and two security random number: n_1, n_2. Both the server and tags keep *ID*, *IDS*, k_1, k_2, n_1 and n_2 locally. Every time a reader receives an *IDS*, it communicates with the server. The server matches the *IDS* in the database and returns the corresponding k_1, k_2, n_1 and n_2 to the reader. The Gossamer protocol uses not only the *EPCglobal* standard defined operations: bitwise *XOR*, bitwise *AND*, bitwise *OR* but also new functions: $ROT(x,y)$ and $MIXBITS(x,y)$. These two new functions are used to calculate verification messages and update local variables. Function $ROT(x,y)$ is defined to conduct a circular shift for the bit sequence x, (y mod 96) positions to the left. Function $MIXBIX(x,y)$ uses bitwise right shift (>>) and additions to compute an output Z:

$$Z = x;$$

$$for(i = 0; i < 32; i++)$$

$$Z = (Z>>1) + Z + Z + Y;$$

The Gossamer protocol includes four steps and is illustrated in Figure 2:

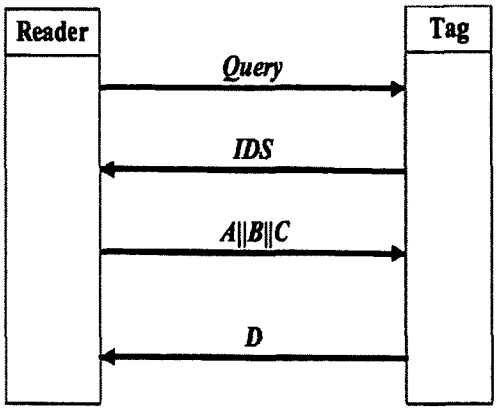

Figure 2: The Gossamer protocol.

Tag identification. A reader sends the *query* command to a tag. The tag replies with the *IDS*. The reader then sends the *IDS* to the server and gets the related k_1, k_2.

Mutual authentication. A reader generates A, B and C separately and sends them together to the tag. The function *ROT* is used twice with five parameters to generate A and B according to the following equations 1 and 2:

$$A = ROT(ROT(IDS + k_1 + \pi + n_1, k_2) + k_1, k_1) \tag{1}$$

$$B = ROT(ROT(IDS + k_2 + \pi + n_2, k_1) + k_2, k_2) \tag{2}$$

Next, the function call $MIXBITS(n_1, n_2)$ is used to generate n_3. The function call $MIXBIT(n_3, n_2)$ is used to generate a new n_1 which is stored in the server as n_1'. The k_1 and k_2 are updated by using the *ROT* function to k_1' and k_2' in equations 3 and 4:

$$k_1' = ROT(ROT(n_2 + k_1 + \pi + n_3, n_2) \oplus n_3, n_1) \oplus n_3 \tag{3}$$

$$k_2' = ROT(ROT(n_1 + k_2 + \pi + n_3, n_1) \oplus n_3, n_2) \oplus n_3 \tag{4}$$

Then the C is generated as:

$$C = ROT(ROT(n_3 + k_1' + \pi + n_1', n_3) + k_2' \oplus n_1', n_2) \oplus n_1' \tag{5}$$

The reader now concatenates A, B, C together and sends the result to the tag. The tag extracts n_1 from A and n_2 from B by using its local IDS, k_1, k_2 to generate C' with the function 5. If $C = C'$, the tag generates D in the equation 6:

$$D = ROT(ROT(n_2 + k_2' + ID + n_1', n_2) + k_1' + n_1', n_3) + n_1' \tag{6}$$

and the tag sends D to the reader for verification.

Tag update and backend database update. A reader uses equation 6 to generate D' with its local variables. If $D = D'$, then the backend server and tag update all the local parameters for the next verification. The function $MIXBITS(n_1', n_3)$ is used to update n_2 and the IDS is updated in the equation 7:

$$IDS = ROT(ROT(n_1' + k_1' + IDS + n_2', n_1') + k_2' \oplus n_2', n_3) \oplus n_2' \tag{7}$$

k_1 and k_2 are updated by using equations 3 and 4 again. In the backend database, previous IDS, k_1 and k_2 are stored to recover from an eventual de-synchronization attack. An adversary can block the last transmission from the tag to the reader, so that the tag can update its information while the reader does not update its local records due to the lack of the verification message from the tag. In order to recover from this de-synchronization attack, tags have to save both current IDS, k_1, k_2 and previous ones. If the database can not identify the current IDS, the reader requires a tag to send its IDS again. The tag replies with the previous IDS to resynchronize each other again.

Progress and constrains. The Gossamer protocol achieves untraceability by using different *IDS* for each inventory execution. The *IDS* is updated after each successful inventory to look random to the environment. However, it uses two defined functions that are not supported by the current *EPCglobal* standard, i.e. *ROT* and *MIXBIT*. RFID tags need to be re-designed to implement the protocol. Moreover, an adversary can eavesdrop on the previous value of $A||B||C$ and the tag's previous *IDS* to fake a de-synchronization scenario to communicate with a tag. In our proposed protocol, the *IDS* is also used to achieve untraceability. Only the *EPCglobal* standard compatible operations are used. And also, if a tag and a reader are de-synchronized, they need to recover from the de-synchronization and then verify each other again before starting a new communication.

2.2 The CD protocol

In the second branch, the *PRNG* and *CRC* functions are used to secure the inventory protocol. They are compatible with the *EPCglobal* standard. In the CD protocol [6], the *CRC* is used to encrypt the verification data. It does not need any permanent connection with a backend server but requires readers and tags to register to the server before they are put into use. Each tag shares three private parameters with the back-end server: a key k, a nonce N and an unique identifier *EPC*. The CD protocol is shown in Figure 3.

Their inventory protocol includes three steps:

Tag identification. A reader inquiries a tag with three messages: a request: M_{req}, a *CRC* function result $CRC(N \oplus RND)$ and a random number *RND*.

Reader verification. A tag verifies the $CRC(N \oplus RND)$ by calculating the result with its local nonce N, the received *RND* and the *CRC* function. If this *CRC* result is verified, a tag generates a new random number (RND_{new}) and computes tag-verification information:

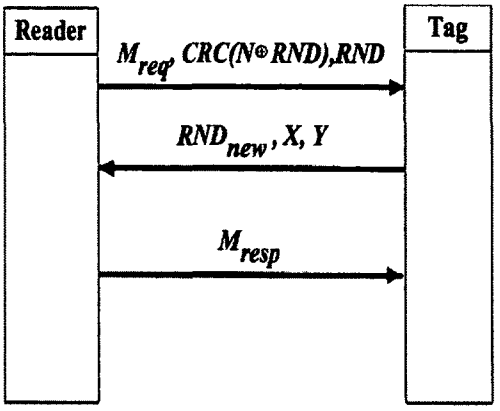

Figure 3: The CD protocol.

X and Y with its identification information EPC, local key k and nonce N. Calculations are shown in equations 8 and 9. A tag sends X, Y and RND_{new} to the reader.

$$X = (k \oplus EPC \oplus RND_{new}) \tag{8}$$

and

$$Y = CRC(RND_{new} \oplus N \oplus X) \tag{9}$$

Tag verification. A reader verifies the received X and Y by using equations 8 and 9 with its local variables and the RND_{new}. If the tag is verified, the reader sends a response to the tag.

Progress and constrains. The use of the CRC function for the verification is proven insecure. Indeed, an adversary can successfully respond a reader's query after the eavesdropping of one interrogation session between the reader and a tag [22]. Although the CD protocol is proven insecure, it is the first one to propose an inventory protocol without a permanent backend server connection. The DTN based RFID protocol includes two stages as well. Readers and tags are configured with private secure parameters first before they

are put into use. Readers then use a DTN to communicate with each other, so that no permanent backend connection is needed.

2.3 The Flyweight protocol

In 2009, a new protocol called Flyweight [5] based on the use of the *PRNG* function was proposed. This protocol closely follows the *EPCglobal* standard and prevents leaking of any tags or reader identities during the interrogation. This protocol requires readers and tags to use a synchronized *PRNG* state (same *PRNG* algorithm and *PRNG* seed) to generate the identical random number sequence. They then use different parts of the random number sequence for mutual verification. Similarly to the Gossamer protocol, a permanent server connection is needed.

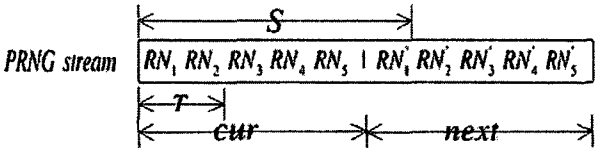

Figure 4: A synchronized PRNG stream.

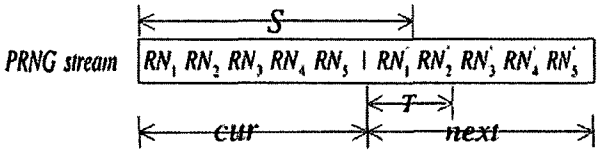

Figure 5: A de-synchronized PRNG stream.

The Flyweight protocol uses random number sequences generated by a *PRNG* function with a 32-bit seed. Every random number sequence is divided into 5 random numbers: (RN_1, RN_2, RN_3, RN_4, RN_5); each number is 16 bits long. As shown in Figures 4 and 5,

the S stands for the server and the T stands for the tag, there are two continuous random number sequences: the *current five random* numbers (RN_1 to RN_5) and the *next five random numbers* (RN'_1 to RN'_5) . A reader is connected with a database that always holds six numbers. They include the first random number of the current sequence RN_1 to the first number of the next sequence RN'_1. A tag holds the *current* RN_1, RN_2 if it is synchronized with the reader (Figure 4) or holds the *next* RN'_1, RN'_2 if it is not (Figure 5). The de-synchronization occurs when a tag updates its random sequence to the next one while the reader still holds the current random sequence. If the reader and the tag are synchronized, RN_1, RN_2, RN_3 from the same sequence are then used for mutual verification as it is shown in Figure 6. If it is a de-synchronization situation, RN'_1 RN'_2,RN'_3, RN'_4, RN'_5 from *next* sequences are used for resynchronization and verification as it is shown in Figure 7.

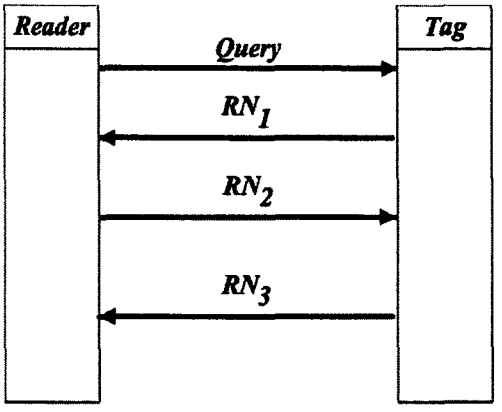

Figure 6: The synchronized Flyweight protocol.

A reader first sends a query to a tag, the tag then replies with RN_1. If the received RN_1 is the *current* RN_1 then RN_2 is replied to the tag. If the received RN_2 equals to the local RN_2, the tag generates the random sequence from RN_3 to RN'_2. The tag replaces RN_1 and RN_2 with RN'_1 and RN'_2 in its memory. The tag sends the RN_3 back to the reader for verification. If the RN_3 is valid, the reader updates random number sequences and this mutual verification session is completed.

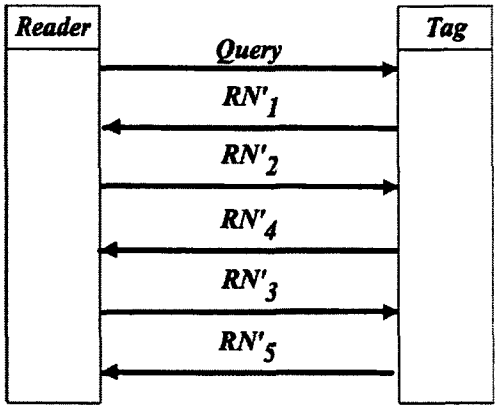

Figure 7: The resynchronized Flyweight protocol.

In Figure 7, if the reader and the tag are de-synchronized, the reader receives the *next* RN'_1. The reader then updates its random number sequence and replies RN'_2 with a flag to alert the de-synchronization. The tag then updates its random number sequence and replies with the RN'_4 to the reader. To acknowledge that RN'_4 has been received, the reader sends back RN'_3 and updates its random number sequences but saves the old RN'_5. The tag then receives RN'_3, if it equals to the local one, the tag sends back RN'_5. If the received RN'_5 equals to the old RN'_5 of the reader, then the tag is accepted. This process is shown in Figure 6. Overall, the tag updates its random sequence once, while the reader updates its local sequence twice.

Progress and constrains. With the use of the *PRNG* function, readers and tags keep less information for mutual verification compared to the Gossamer protocol. A *PRNG* seed needs to be updated for readers and tags. The function and the frequency to update the *PRNG* seed is not specified in their paper. The DTN based RFID protocol uses random number sequences for mutual verification, and after every successful inventory, the *PRNG* seed is updated.

Chapter 3

The DTN based RFID protocol

3.1 Drawbacks of previous protocols

The Gossamer protocol uses two functions: *MIXBITS* and *ROT* which are not supported by current lightweight *RFID* tags. The Flyweight protocol [5] implements a *PRNG* function for mutual verification, but the *PRNG* seed update frequency and update function are not specified. Furthermore, the *EPCglobal* standard does not define connections with a backend database in the inventory protocol. Both the Gossamer and Flyweight protocols, however, require permanent backend server connections. The Gossamer protocol has to use a database to match the tag, generate verification messages and update private keys. A reader is required to connect with the sever all the time. In the Flyweight protocol, the server not only needs to process the data as the Gossamer protocol, but it also has to update the *PRNG* seed.

3.1.1 Design requirement

New designed RFID protocols should achieve compatibility, security and prevent permanent server connections. For the compatibility, the protocol must follow *EPCglobal* standards. There are four requirements for the protocol security. Firstly, both the tag and reader

need to verify each other before conduct further communications. Secondly, adversaries can not link any messages to a tag based on previous ones, which they eavesdrop from the environment. All the tags are untraceable. Thirdly, Previous messages confidentiality is ensured, even if the tag's current local parameters are exposed to an adversary. And fourthly, to an adversary who has access to a tag's current local parameters, future messages keep confidential to the adversary after the tag updates its current parameters.

The DTN based RFID protocol is designed to provide mutual verification without permanent backend server connections. Particularly, it focuses on the following design requirements:

Compatibility. The DTN based RFID protocol only uses functions and operations that are supported by RFID lightweight tags including the *PRNG* function and XOR operation. The Flyweight protocol [5] and studies from [24], [26], [28] have shown that the *PRNG* function can be used in accordance with the requirements from the *EPCglobal* standard.

No server connections. In the DTN based RFID protocol, the initial set of verification parameters are configured within the readers and tags. A *DTN* is used by readers to exchange new information. Each reader in the DTN carries, stores and broadcasts new messages to other readers.

Security. The DTN based RFID protocol protocol is designed to meet mutual verification, untraceability, forward security, backward security.

3.2 The DTN based RFID protocol

The DTN based RFID protocol focuses on mutual verification without permanent backend server connections. It includes two stages: configuration and communication. During the configuration stage, readers and tags obtain the initial set of private secure parameters. In

Table 1: Tag local constants and variables.

TD	RD	TDS	$seed$

the communication stage, the DTN based RFID protocol conducts mutual verification and uses a DTN to connect all readers.

3.2.1 Configuration

The configuration stage is conducted in a secure environment. A tag obtains its identity TD, its pseudonym number TDS, a reader identity RD and an initial $PRNG$ function $seed$ as it is shown Table 1. TD and RD are constants. TD is for tag identification and RD is used to verify readers. TDS and $seed$ are variables and updated after every successful inventory. A tag only has one record of the type of Table 1. A reader needs to use the valid RD to access it.

A reader acquires its identity RD and a list of accessible tag information records. For each tag in the list, a reader stores its TD, previous session TDS (TDS_{pre}), current session TDS (TDS_{cur}), previous $PRNG$ seed ($seed_{pre}$) and current $PRNG$ seed ($seed_{cur}$) as it is shown in Table 2. In our proposed protocol, a list of tags is associated to one RD. Readers that share the same RD have access to the same list of tags. In the configuration stage, no tag previous $PRNG$ seed ($seed_{pre}$) and TDS (TDS_{pre}) are available for readers. Table 2 details one tag information record of a reader's local tag list. Several RDs can be assigned to a reader to access correlated tags. TDS_{pre}, TDS_{cur}, $seed_{pre}$ and $seed_{cur}$ are variables and are updated after a tag is read. The updated information is a new message. The message is sent to other readers to update their local record of the same tag. All parameters are defined in Table 3.

Readers and tags keep the same $PRNG$ state, which includes the same seed and $PRNG$ algorithm in order to generate an identical sequence of random bits. Different parts of the

Table 2: Reader local constants and variables of one tag.

TD	TDS_{pre}	TDS_{cur}	$seed_{pre}$	$seed_{cur}$	RD

Table 3: Constant and variable definitions.

TD	32-bit tag identifier
RD	32-bit reader identifier
TDS_{pre}	16-bit previous pseudonym number of a tag
TDS_{cur}	16-bit current pseudonym number of a tag
$seed_{pre}$	16-bit previous *PRNG* seed
$seed_{cur}$	16-bit current *PRNG* seed
$seed'$	16-bit *PRNG* seed
$RND16$	16-bit random number
$PRNG_{n:m}(seed)$	A random number that starts at bit n and finishes at bit m in a random number sequence generated by *PRNG* with a *seed*.

Table 4: FLAG values for different scenarios.

FLAG=0	FLAG=1	FLAG=2
synchronization	de-synchronization	collision

random sequence are segregated into several random numbers. Each number starts at bit n and finishes at bit m is defined as $PRNG_{n:m}(seed)$.

A tag's *PRNG* seed and TDS are updated every time after successful mutual verification, so that the verification information is different in every inventory. However, if an adversary blocks a certain transmission between the reader and tag, it can cause a de-synchronized *PRNG* state: The reader updates its local parameters while the tag still keeps the current one. This de-synchronization attack is discussed in detail in the sub-section 3.2.3. To solve this security issue, readers store previous verification information, including the previous *PRNG* seed ($seed_{pre}$) and the tag's previous pseudonym number (TDS_{pre}).

Another possible issue may occur after updating a TDS. Several tags can have the same TDS, which the reader cannot differentiate. The TDS collision problem is discussed in detail in the sub-section 3.2.4.

A FLAG parameter is used by the reader in the inventory protocol to inform a tag that a de-synchronization or a collision has occurred. The different values of the FLAG and their meanings are shown in Table 4.

3.2.2 Communication

The usage of a random number sequence

Two *PRNG* seeds are used in the DTN based RFID protocol: *seed* and $seed'$. The *seed* is originally obtained through configuration and the reader stores it as $seed_{cur}$. $seed'$ is dynamically generated in every inventory session. Readers and tags update their local *seed*

with the constant *RD*, the previous *seed* and a 32-bit random number after every successful mutual verification. A 16-bit number drawn from it is to prove that the tag holds the valid *RD*. The *seed'* is generated with the constant *TD*, the current *seed* and a 32-bit random number. A 16-bit number drawn from it is to show the reader has access to the tag with the identity *TD*.

Table 5: The random number sequence generated from *seed*.

$\mathbf{PRNG}_{0:15}(seed)$	$\mathbf{PRNG}_{16:47}(seed)$	$\mathbf{PRNG}_{48:79}(seed)$	$\mathbf{PRNG}_{80:111}(seed)$
16-bit number to prove a reader identity	32-bit number for *seed* update (First 16 bits are for TDS update)	32-bit number for *seed'* generation	32-bit number for resynchronization

Table 6: The random number sequence generated from *seed'*.

$\mathbf{PRNG}_{0:15}(seed')$	$\mathbf{PRNG}_{16:31}(seed')$	$\mathbf{PRNG}_{32:47}(seed')$ and $\mathbf{PRNG}_{48:63}(seed')$
16-bit number to prove a tag identity	16-bit number for *TDS* update	two 16-bit numbers for *TDS* collision resolution

Two random bit sequences are divided into four numbers each. The first random number sequence that is generated from the *seed* is shown in Table 5. It is 112 bit long. The first 16 bits of the sequence are used to prove a reader's identity. The 16th bit to the 47th bit in the sequence are used as a 32-bit random number to update the $seed_{cur}$ for a reader or

the *seed* for a tag. The first 16 bits of this 32-bit random number are used to update a tag's *TDS*. From the 48th bit to the 79th bit is another 32-bit random number to generate the *seed'*. The last 32 bits are used as a random number to update the $seed_{cur}$ for a reader and *seed* for a tag when a de-synchronization occurs.

Table 6 details the second random number sequence that is generated from the *seed'*. it is 64 bit long. The first 16 bits of the random sequence are used to prove a tag's identity. From the 16th bit to the 31st bit of this sequence is a 16-bit random number to update a tag's *TDS*. The 32nd bit to the 47th bit and the 48th bit to the 63rd are segregated into two 16-bit random numbers to resolve a collision.

The DTN based RFID protocol

This protocol includes four steps as it is shown in Figure 8.

Tag identification. First, a reader sends a *Query* to a tag as outlined in step 1. The tag replies its *TDS* back to the reader in step 2. If the reader and tag are synchronized with each other, the received *TDS* is a TDS_{cur} in a reader's local record. The reader sets FLAG to 0 and backs up the $seed_{cur}$ as $seed_{pre}$. A new $seed_{cur}$ is generated to draw a 16-bit random number (*RND*16) that is sent to the tag for reader verification. The *seed'* is generated. The TDS_{pre} and TDS_{cur} are updated. The reader sends the 16-bit random number (*RND*16) and FLAG to the tag.

Reader verification. After the tag receives the *RND*16 and FLAG from the reader as outlined in step 3, it updates its *seed* temporarily as $seed_{temp}$. A 16-bit random number is generated from the temporary seed to verify the received *RND*16. Only if the received *RND*16 is verified, the reader is considered valid and the tag saves the temporary seed $seed_{temp}$ as *seed* and generate the *seed'*. The *seed'* then is used to draw a 16-bit random number (*RND*16) to prove the tag identity to the reader. Before the tag sends the 16-bit

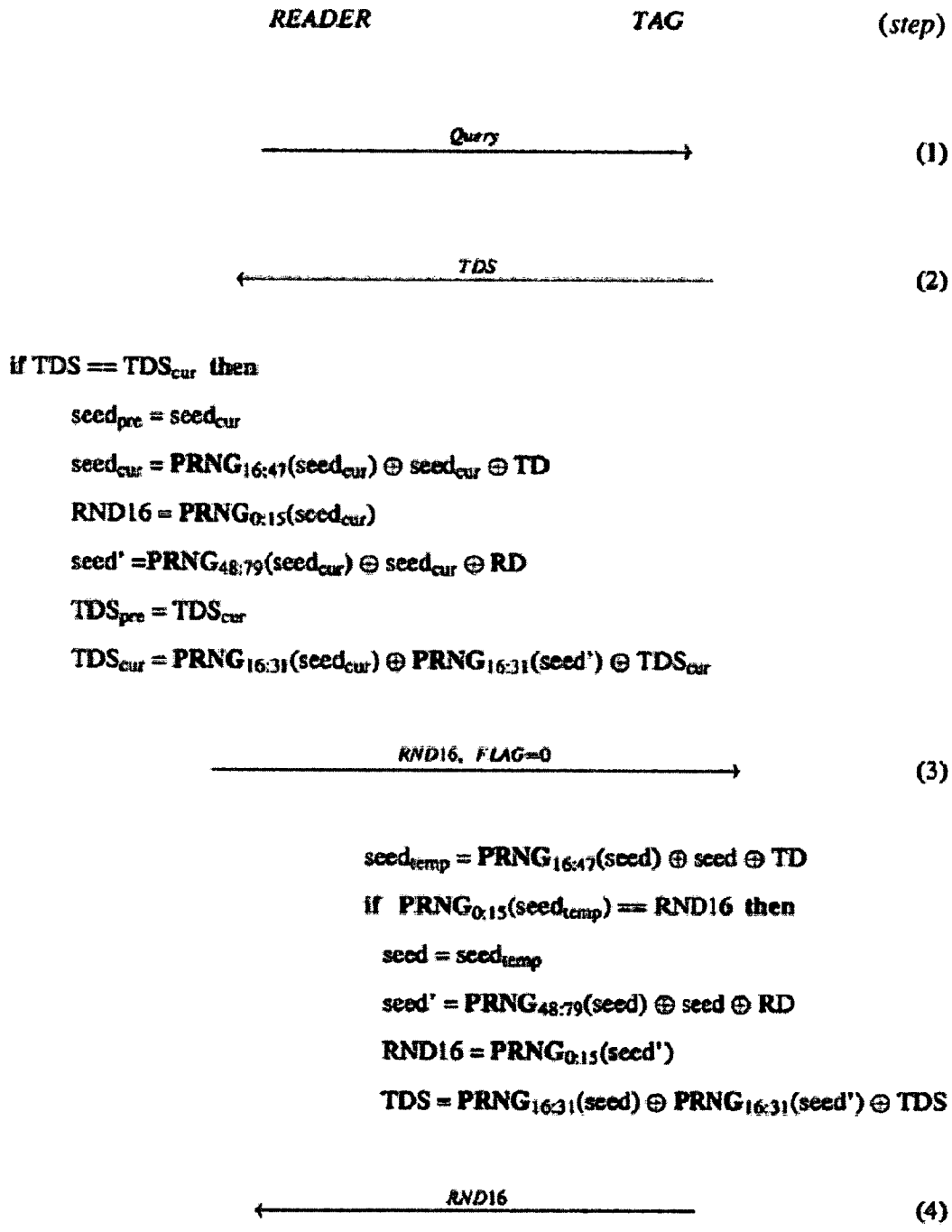

Figure 8: The DTN based RFID protocol.

random number back to the reader, the tag also updates its TDS.

Tag verification. In step 4, the reader receives the 16-bit random number ($RND16$) from the tag. It then uses the $seed'$ to draw another 16-bit number. If it is equal to the received number, the tag is valid. The reader accepts the tag.

3.2.3 Resynchronization

Since the reader updates its local variables when the TDS received from the tag is valid and the tag updates its local variables after it verified the reader, they do not update their local variables at the same time. An adversary can block the third transmission when the reader sends a 16-bit number ($RND16$) to the tag. The reader updates its local variable but the tag remains in its current state and waits for a message from the reader. Thus, de-synchronization occurs.

The following strategy is used to resynchronize the reader and tag. It requires one interrogation to complete resynchronization. It also needs three steps as it is shown in Figure 9.

Tag and de-synchronization identification. In step 2, if a reader and a tag are de-synchronized, the received TDS equals the TDS_{pre} of a reader's local record. The reader sets the FLAG to 1 to alert de-synchronization. The reader generates a temporary seed: $seed_{temp}$ to draw a 16-bit random number ($RND16$) for the tag to verify the reader.

Reader verification and resynchronization. The tag receives FLAG=1 and 16-bit random number ($RND16$) in step 3, the tag also generates a temporary seed: $seed_{temp}$. The temporary seed is used to draw a 16-bit random number to verify $RND16$. If they are equal, the reader is considered valid. The tag updates its $seed$ to generate the $seed'$. A 16-bit random number is drawn from $seed'$ to be sent to the reader. Before the tag sends

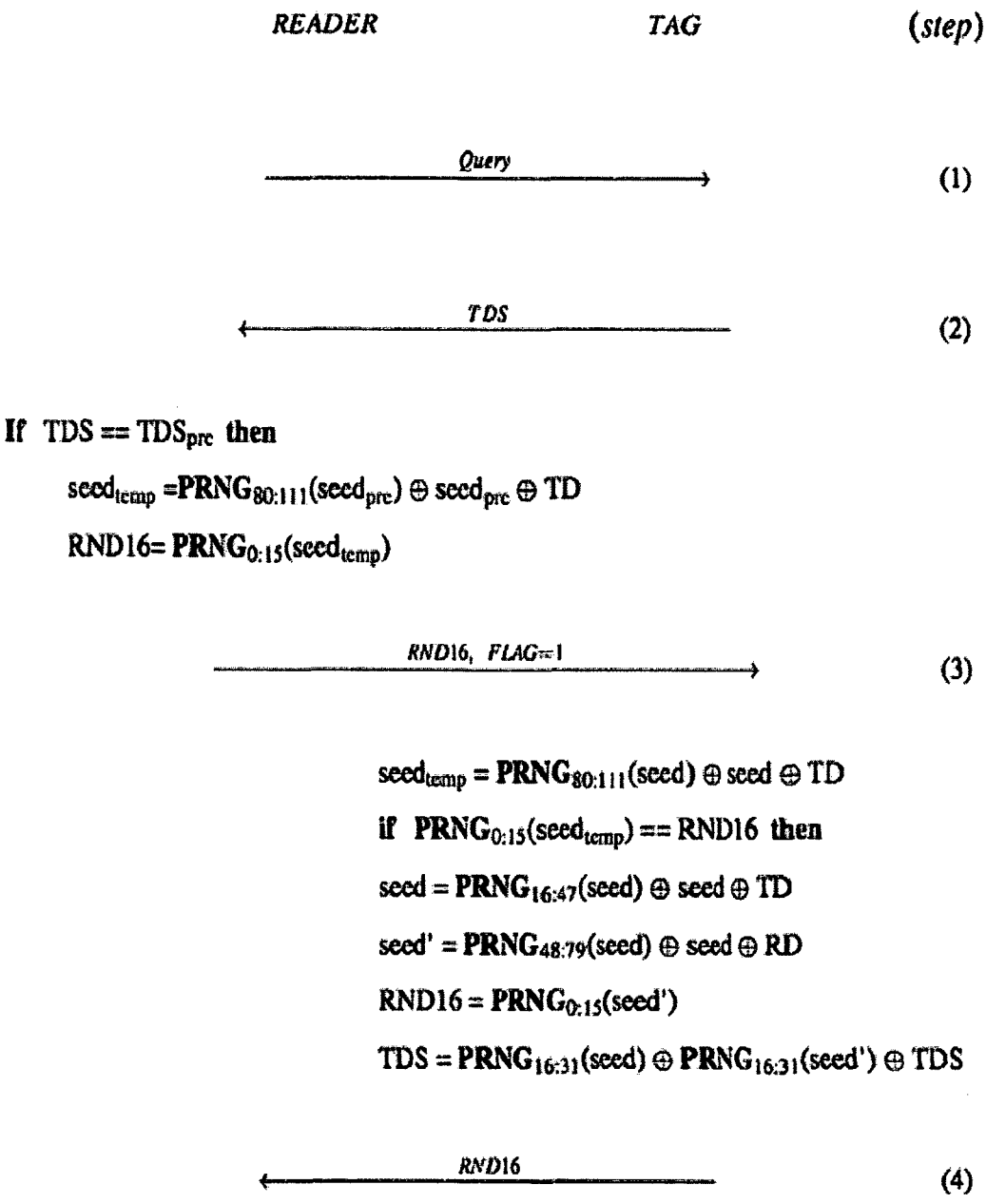

Figure 9: The DTN based RFID protocol resynchronization strategy.

back the random number, it updates its TDS.

Tag verification In step 4, the reader generates its $seed'$ to draw a 16-bit random number. If the received number and generated number are equal then the tag is resynchronized with the reader. The resynchronization session is complete. The reader then starts another inventory session for mutual verification.

3.2.4 Collision resolution

Since the tag TDS is updated every time after an inventory is performed, it is possible that several tags simultaneously share the same TDS. When a TDS collision occurs, the reader tries to differentiate the tag first and then begins another inventory for mutual verification. This strategy is shown in Figure 10.

Tag and collision identification. If several reader's records have their TDS_{cur} equal to the received TDS in step 2, a TDS collision occurs. The reader sets the FLAG to 2 to alert the tag. The $seed'$ is generated to draw a 16-bit random number ($RND16$). $RND16$ is sent to the tag along with FLAG.

Reader verification. In step 3, after the tag receives the message from the reader, it generates the $seed'$ to draw the 16-bit random number. If the received 16-bit number and generated number are equal, the reader is considered valid. The tag then generates a temporary $seed_{temp}$ to calculate the $seed'$. The $seed'$ is used to draw a 16-bit random number to prove the tag identity to the reader.

Tag verification and collision resolution. Similarly, the reader generates the temporary $seed_{temp}$ and then calculates the $seed'$ to draw the 16-bit random number to verify the received one in step 4. If the received 16-bit number and generated number are equal, then

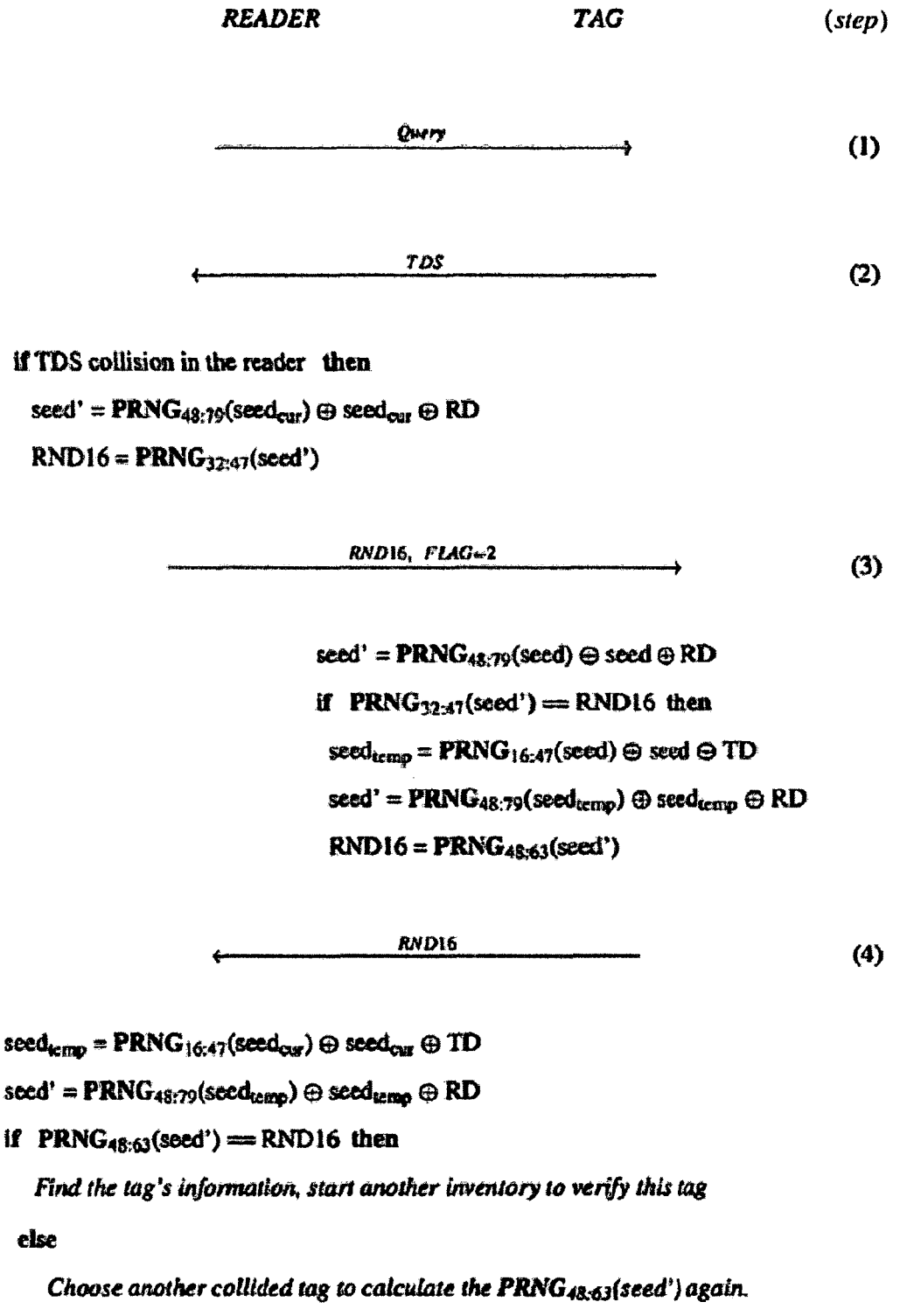

Figure 10: The DTN based RFID protocol collision resolution strategy.

the chosen tag is the correct one. If the generated 16-bit random number does not equal the received one, the reader chooses another collided tag to re-calculate the 16-bit number.

This strategy requires the tag to update the $seed_{cur}$ to generate a new $seed'$ temporarily. The $seed'$ is used to draw the 16-bit number used by the reader to differentiate the tag. Since the TD is uniquely assigned to each tag, by updating the $seed_{cur}$ to $seed_{temp}$ with the TD, the updated $seed_{temp}$ are different between collided tags. Thus, the $seed'$ that is generated from the $seed_{temp}$ and the 16-bit random number drawn from the $seed'$ can differentiate collided tags.

During a collision situation or a de-synchronization, readers and tags need to recover to normal state before beginning a new mutual verification session. An adversary cannot replay the collision message or the de-synchronization message for mutual verification. Once a tag is synchronized with a reader, the de-synchronization alert message becomes invalid. If an adversary re-sends the collision alert message to the tag, the tag replies with the same message it sent to the reader before. Furthermore, adversaries cannot alert a new de-synchronization or collision without obtaining a valid reader's *PRNG* seed, tags need to verify the reader before replying or changing their local state.

Chapter 4

Analysis and delay tolerant network

In this chapter, we first discuss available *PRNG* functions and data deployment strategies for our proposed protocol. Security is also analyzed against relevant attacks in the second part of this chapter. In the final part of this chapter, Delay Tolerant Network is studied as a message transmission media to synchronize RFID readers.

4.1 *PRNG* implementation

The *PRNG* function takes an initial set of values to generate a sequence of numbers having approximately random properties. The set of values is called a *PRNG* seed. The sequence of numbers is reproducible, when the same *PRNG* seed is provided. In our proposed protocol, readers and tags keep the same *PRNG* seeds locally to generate identical number sequences for mutual verification. *EPCglobal* standards provide specific requirements for the *PRNG* function without a specific algorithm for implementation. Several efficient *PRNG* function algorithms are proposed and discussed for lightweight RFID tags. The self-shrink *PRNG* [21] was developed from the shrinking *PRNG* [9]. It has been implemented for RFID tags. It is estimated to use 1435 logic gates and 64bytes of memory [20]. More recently, the *EPCglobal* standard compatible *PRNG* function LAMED-EPC [23] has been proposed. It requires 1566 logic gates and 64bytes of memory. Both of these two *PRNG*

Table 7: Inventory commands (they are all mandatory).

Command	Length (bits)
Query	22
QueryAdjust	9
QueryRep	4
Query_Reply(RND)	16
ACK	18
ACK_Reply	21 to 528
NAK	8

function can be utilized within our proposed protocol.

4.2 Compatibility analysis

4.2.1 Protocol compatibility

The *EPCglobal* standard specifies a communication between a reader and a tag into three processes:

- Select. This process is for a reader to select a tag population defined by user specified criteria. This command is mandatory before an inventory and access round.
- Inventory. This process is for a reader to identify a tag. Inventory commands include *Query*, *QueryAdjust*, *QueryRep*, *Query_Reply(RND)*, *ACK*, *ACK_Reply* and *NAK*. Our proposed protocol inherits those commands to achieve mutual verification without changing their original functions. Lengths of inventory commands are given in Table 7. All current commands are mandatory.
- Access Commands

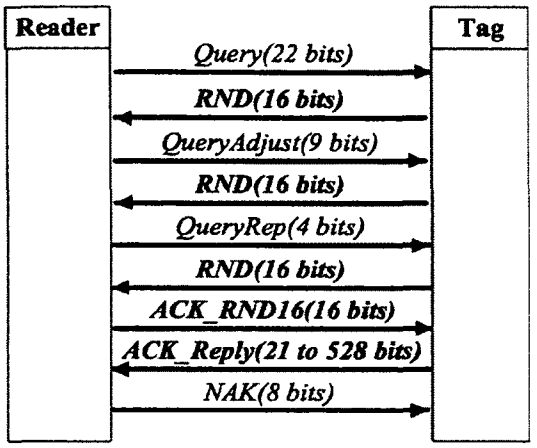

Figure 11: The *EPCglobal* standard inventory protocol.

Access Commands execute actual interactions between a reader and a tag. A tag can be read, rewritten, updated or killed. Access commands can only be used after an inventory session.

A full detailed *EPCglobal* inventory protocol is shown in Figure 11. Our proposed protocol is shown in Figure 12.

In the *EPCglobal* inventory protocol, the reader sends commands *Query*, *QueryAdjust*, *QueryRep* to conduct a random-slotted collision algorithm. First, the reader sends the *Query* that contains a parameter Q to the tag. The tag generates a random Q-bit number locally as a counter and a 16-bit random number *RND*. It replies the *RND* as an acknowledgement to the reader. Second, one or several *QueryAdjust*s can be sent from the reader to adjust the tag's counter value. Every time, the tag replies the same *RND* to the reader as an acknowledgement. Third, the tag begins to decrease its counter when it receives the *QueryRep*. The tag sends the same *RND* back to the reader when the counter reaches zero. The reader then sends back the same *RND* to the tag as *ACK*. If the tag does not receive a valid *ACK* due to a collision, then it returns to the initial state and repeats the process. Otherwise, the tag sends back an *ACK_Reply* including its *EPC* identification. At the end,

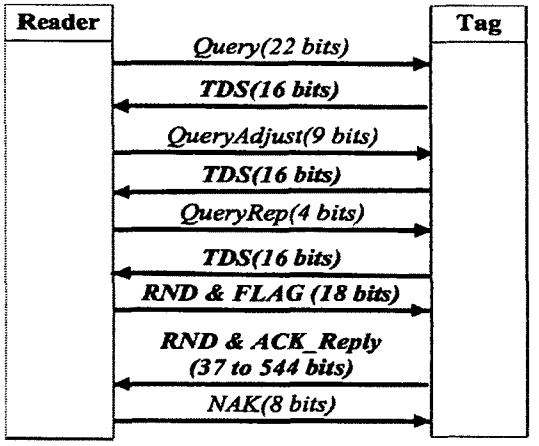

Figure 12: The DTN based RFID protocol.

the reader replies a *NAK* to complete the inventory round.

As defined in Table 7, support of all *EPCglobal* inventory commands are mandatory. In our proposed protocol, these commands are adopted with minor modifications. *TDS* is used to replace the 16-bit random number (*RND*) sent from the tag to the reader to echo *Queries*. The *ACK* sent from the reader is replaced by a 18-bit number including a calculated 16-bit random number and a 2-bit FLAG. We also extend the *ACK_Reply* message to include a 16-bit random number for tag identity verification. The three *Queries* and the *NAK* commands are kept the same for the random-slotted collision algorithm.

4.2.2 Data compatibility

The *EPCglobal* standard specifies four data banks for RFID tags [13].

Reserved memory (bank00). This bank of data is reserved for the kill password and access password. The kill password is used for permanently terminating the use of one tag. The access password is used by the reader to perform privileged operations. Those two passwords are used by access commands.

EPC memory (bank 01). This data bank stores *EPCglobal* protocol-control information.

TID memory (bank 10). This data bank keeps the *TID* (Tag Identification). The *EPCglobal* standard outlines two schemes to identify a tag:

- 32 bits Short Tag Identification. It includes a 12-bit tag *MDID* (Tag Mask Designer Identifier), a 12-bit tag model number and a 8-bit class identifier.
- 128 bits Extended Tag Identification. The first 32 bits are the same as the short tag identification. The remaining 96 bits (Extended *TID*) are kept for end users and RFID applications. The Extended *TID* memory section can be partitioned into six segments. Each segment has16 bits.

User memory (bank 11). This data bank is optional. It provides user defined data storage for *EPCglobal* applications.

4.2.3 Data deployment

Based on the *EPCglobal* standard, the possible data memory deployment for our proposed protocol is as follow.

***RD* (32 bits), *TDS* (16 bits) and *seed* (32 bits).**

RD is an identity of readers. When it is configured into a tag, it is stored permanently. *TDS* is the pseudonym number. *seed* is a *PRNG* seed. Their initial value is configured into a tag and is updated after every inventory. These three 16-bit numbers can be stored in three segments of the extended *TID* in the TID memory (bank 10).

TD **(32 bits).**

The *TD* is an unique identity for a tag in the DTN based RFID protocol. According to the *EPCglobal* standard, the *TID* is different for tags with different custom commands and/or optional features. If each tag has unique features and/or RFID commands, then the DTN based RFID protocol can use the *TID* as *TD*. If tags have common commands or features (e.g., cell phones have same features or functions but they need to be treated individually for different customers), then the *EPCglobal* tag URI can be used. The *EPC URI* scheme is used within an information system to represent each physical object. In our proposed protocol, the URI scheme can be used to differentiate all tags. The URI format is defined as *urn:epc:id:scheme:component*1.*component*2. The *urn:epc:id:scheme* is the URI body. It indicates the URI scheme. The *component*1 and *component*2 are the remainders of the *scheme*. The *EPC URI* currently supports ten schemes. The *sgtin* scheme is typically used for trading item. Each object has an unique identity. The general syntax of *sgtin* scheme is *urn:epc:id:sgtin:CompanyPrefix.ItemReference.SerialNumber*. The *CompanyPrefix* represents a managing entity. The *ItemReference* is assigned by the managing entity to a particular object class. The particular object class gives each individual object an unique *SerialNumber* within its scope. Following this scheme, in an user specified criteria, tags can be divided into groups by assigning different 32-bit *CompanyPrefix* and *ItemReference*. Each tag is assigned an unique 32-bit *SerialNumber* within one group. The *SerialNumber* can be used as tags *TD*. The *ItemReference* and *CompanyPrefix* can be used as *RD*. Thus, tags in the same group share a *RD* but they have different *TD*s. Readers use the *RD* to access a group of tags and use a *TD* to verify the identity of an individual tag.

4.3 Security analysis

The DTN based RFID protocol meets the following security requirements: mutual verification, data confidentiality, tag anonymity and untraceability, forward security and backward security.

4.3.1 Mutual verification

Readers and tags need to verify each other before starting a communication. Mutual verification is achieved by using the same *PRNG* state: Readers and tags keep a same *PRNG* function and *PRNG* seeds to generate identical random number sequences for mutual verification. After every successful inventory, the *PRNG* seeds are updated for the next inventory. Thus, as long as the *PRNG* states remain synchronized, tags and readers can verify each other.

4.3.2 Data confidentiality

All transmission messages between tags and readers are random numbers. With a synchronized *PRNG* states, valid readers and tags can recognize those messages and verify each other. Since the *PRNG* seed is updated after every communication, the random number sequence is different in each inventory.

4.3.3 Tag anonymity and untraceability

In our proposed protocol, a tag sends a pseudonym number (TDS) instead of the real identity TD for identification. The TDS is updated every time after a successful inventory. Readers need to use a different TDS to identify the same tag in the next inventory. A tag's identification is hidden to the environment and its anonymity is guaranteed.

4.3.4 Forward security

Forward security is defined in [5] as: previous messages of a tag need to look random even if the tag is corrupted. In the DTN based RFID protocol, if a tag is compromised, its local parameters (*TD*, *RD*, *TDS*, *seed*) are exposed to the adversary. The *PRNG* seed is updated as followed:

$$seed_{cur} = TD \oplus seed_{cur} \oplus PRNG_{16:47}(seed_{cur}) \tag{10}$$

According to this equation, $seed_{cur}$ is updated using the previous $seed_{cur}$. The previous seed is erased. Adversaries require previous seeds to regenerate previous messages. Since tags do not store previous seeds, adversaries cannot generate previous messages and forward security is achieved.

4.3.5 Backward security

Backward Security is defined as: following the update of a tag by a reader, future communication messages look random to adversaries, even when they have access to a tag's current state [5]. In the DTN based RFID protocol, once a tag is updated, a new *PRNG* seed and a new *TDS* are generated for the upcoming verification. The previous *PRNG* state and *TDS* are invalid and adversaries cannot use them to decipher future messages.

4.4 Delay tolerant network

4.4.1 Overview

Most proposed RFID protocols use backend server connections to synchronize RFID readers. In some scenarios, it is not practical to use stationary readers with permanent backend server connections. For example, in a warehouse with many products on shelves, it is more

convenient for an employee to use a portable reader to check and update products information. In order to avoid permanent backend server connections, we use the Delay Tolerant Network (DTN) to connect readers.

Following each successful inventory, a tag updates its *PRNG* seeds and *TDS*, while the reader generates a new message and synchronizes with other readers. The new generated message includes the tag's new *TDS*, previous *TDS* and two *PRNG* seeds. Whenever two readers are within each other's communication range, they exchange new messages. Readers update local tag records according to received messages and keep a list of the latest tags' information to share with other readers. Each message is assigned a message Time To Live (TTL). A message is dropped when its TTL expires.

4.4.2 Delay-tolerant networking

The DTN [16] addresses the message transmission between different nodes in a network which lacks continuous connectivities. Routing protocols in a DTN need to *store* and *forward* messages in order to increase the dissemination of information. No routes are pre-established. A DTN has several regions. Each region has different nodes that share a unique region *ID*. A node name consists of two parts: entity *ID* and region *ID*. Nodes only communicate with those that have the same region *ID*. There are region gateways that can bridge different regions. Within one region, nodes use entity IDs to differentiate each other.

In our proposed protocol, readers are DTN nodes that are responsible to receive, store and transmit new messages. Messages that are generated by the readers are not given specific destination. They need to be sent to all nodes with the same delivery priority. Readers only keep the most recent messages and relay them to other nodes.

Tags can be divided into several regions by assigning different region *ID*s. A region *ID* is used as *RD* in our proposed protocol. A valid reader can check a group of tags and a tag can verify a reader with the region *ID*. Within one region, each reader also keeps an unique

entity *ID* to differentiate each other and transmit messages. When a new reader needs to join one existed region, it is configured with the region *ID* and an unique entity *ID*.

4.4.3 DTN message routing

All the readers in the same region need to update their local records when there is a new message generated. All readers are destinations for the message. The message routing protocol does not need to provide a one-to-one routing strategy. We assume that positions of portable readers change all the time. Therefore no fixed path can be determined. No movement patterns can be predicted. The epidemic routing protocol [31] is considered as a possible routing strategy for the DTN based RFID protocol. Packets are received at intermediate nodes and forwarded to all the node's neighbours except the one who sent the packet.

4.4.4 DTN and the DTN based RFID protocol

Our proposed protocol is conducted in the *EPCglobal* inventory session between the reader and the tag. A DTN is only used between RFID readers, RFID tags are not participants. Every time after a reader checked a tag, new TDS_{cur}, TDS_{pre}, $seed_{cur}$ and $seed_{pre}$ are generated and need to be synchronized to other readers to make the tag readable again. The DTN is a media between readers to exchange new messages and all DTN communications are assumed to be secure. Each RFID reader is a DTN node and can be used at random locations. After a RFID reader receives a message, it updates its local records to synchronize with the correlated RFID tag.

Chapter 5

Simulation

In this chapter, the simulator - *Opportunistic Network Environment (ONE)* [19] is used to emulate the RFID readers communications.

5.1 Introduction

In simulator *ONE*, three parts of a DTN are simulated: node compatibility, node mobility and message routing. The node compatibility enables nodes to act as a *store* and *carry* router using different routing algorithms. Certain parameters are provided to simulate: *network interface, persistence storage, message routing* and *energy consumption.*

Routing strategies, for messages in a DTN, include *epidemic routing* [31] and *spray and wait routing* [30].

ONE provides map-based mobility and working day movement model for node mobility simulation. For map-based mobility, nodes are only allowed to move on pre-defined routes with different movements strategies including random movement, shortest path movement and routed movement. The random movement mode allows nodes to walk randomly on the map. The shortest path movement randomly chose a destination from the map. Between the previous destination to the next destination, nodes take the shortest path

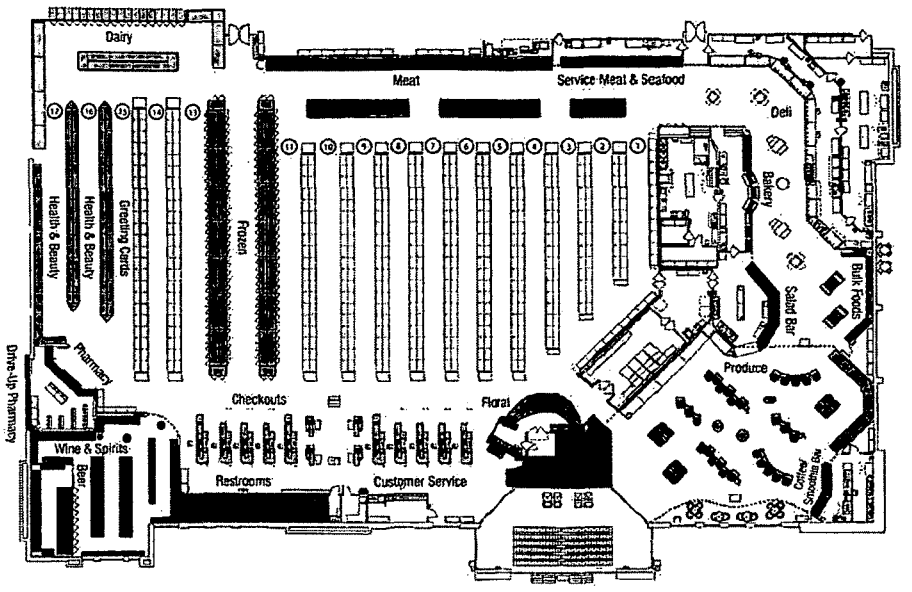

Figure 13: The map of a supermarket.

and stay at one destination for a random time interval. The routed movement uses predetermined routes for nodes. The working day movement model consists of several sub models that define daily routings at different time of a day for objects within a city.

5.2 Scenario

Readers communicate with each other within a DTN. Whenever a tag is read by a reader, a new message is generated. It is sent to other readers to make the tag readable again. We describe a supermarket scenario. There is a mall with goods and products that are equipped with RFID tags. Readers are used by workers to manage different product information. An example supermarket map is provided for this scenario in Figure 13.

Based on the map, the available routes are drawn as Figure 14 for reader holders to walk. The size of this supermarket is approximately 3,700 square meters. Each node has a 10 meter communication range. They can communicate with each other using a bluetooth

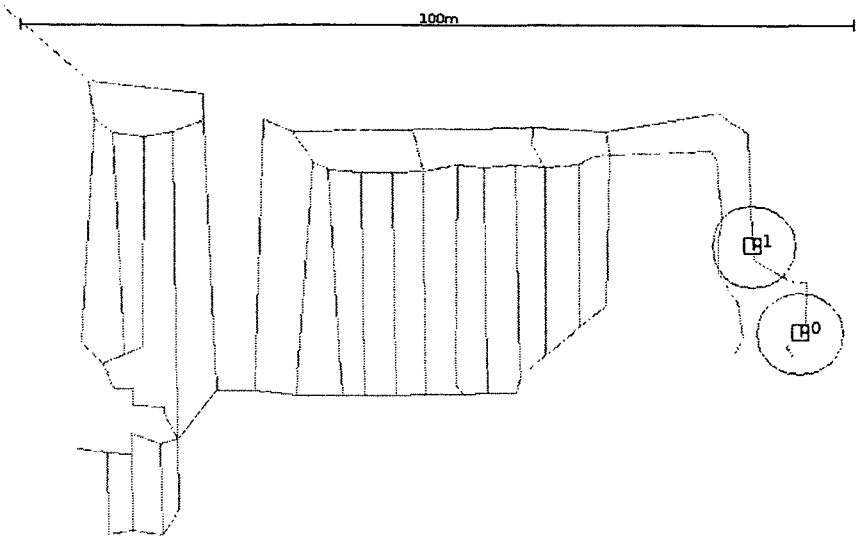

Figure 14: The route map.

network interface.

All the reader holders walk randomly on the route defined in Figure 14. Once they reach a destination by taking the shortest path, they remain there for a random interval of time. Then, the next destination is randomly determined.

In this scenario, the main purpose of RFID reader holders is to check or update certain product information with a reader. Therefore, a new message is generated with the updated information regarding the tag every time a reader holder reaches a destination. Those new messages are broadcasted between different readers whenever they are within each others range. Each reader only updates new messages according to their local records and discards those messages whose TTL is expired.

All the readers in the simulation use a transmission speed of 250KBps. A reader's buffer size is 5M bytes, while a single updated tag message is 160 bits. Since all the messages are assigned a same TTL. They are discarded when their TTL is expired. It is assumed that readers have enough space to store all incoming messages until they become expired.

Each reader holder walks at a randomly chosen speed between 0.5 to 1.5 meters per second. Once a reader holder arrives at its destination, it stays there for a random time, between 0 to 200 seconds.

5.3 Performance analysis

Each message contains update information for a reader's local tag record. Reader needs to receive the message and update the tag to make it readable again. We say a message is synchronized, if all readers receive this message and update its local record within one DTN region. Two parameters are defined to evaluate the result of this simulation: *Message Coverage Rate* and *Message Update Success Rate*. The *Message Coverage Rate* indicates the average percentage of synchronized messages to the overall number of messages among all nodes, at a certain time point. The *Message Update Success Rate* shows the percentage of messages that have been sent to all nodes before they are dropped. Readers require a high *Message Coverage Rate* to stay synchronized. They also require a 100% *Message Update Success Rate* to ensure that all messages can be sent to all readers before they are dropped. In order to show the influence of the message traffic, the same simulation is conducted under two different situations: 100% message generation and 50% message generation. 100% message generation enables readers to generate a message every time they stop at a destination while the 50% message generation only gives 50% possibility to a reader to generate messages when it arrives at a destination.

Figure 15 and 16 show the total number of messages that are generated before they reach the TTL and begin to be dropped. The number of messages from scenarios with three, six and nine nodes are demonstrated together at different time from 300s to 2400s. The TTL is longer than 2400, no messages are discarded in these two figures. The number of message is growing as time goes by. Those two figures also show the difference of the number of messages from the 100% message generation In Figure 15 and the 50% message

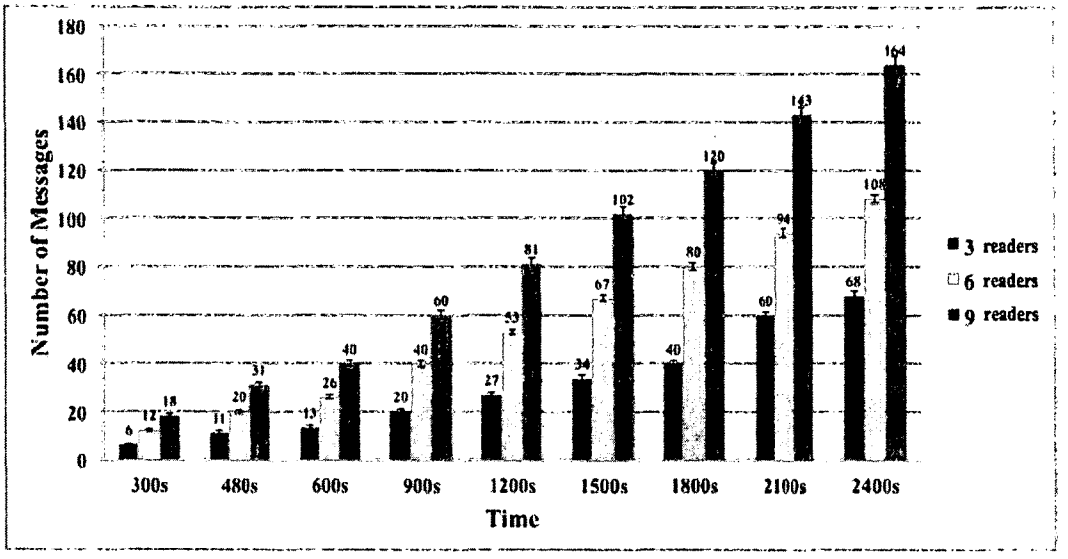

Figure 15: 100% message generation - The total number of generated messages with the 90% confidence interval.

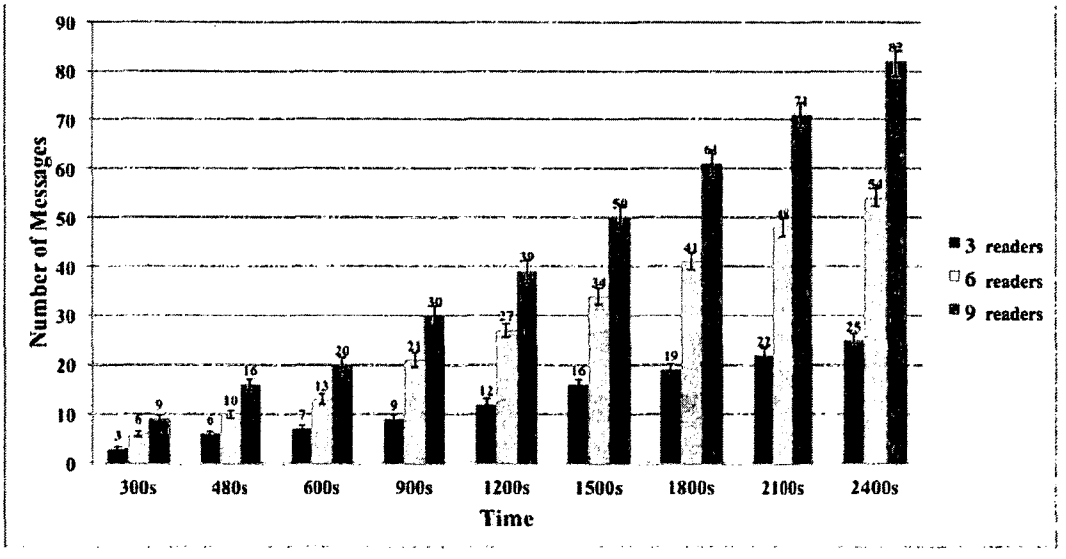

Figure 16: 50% message generation - The total number of generated messages with the 90% confidence interval.

generation, the formal one generates approximately twice number of messages of the later one.

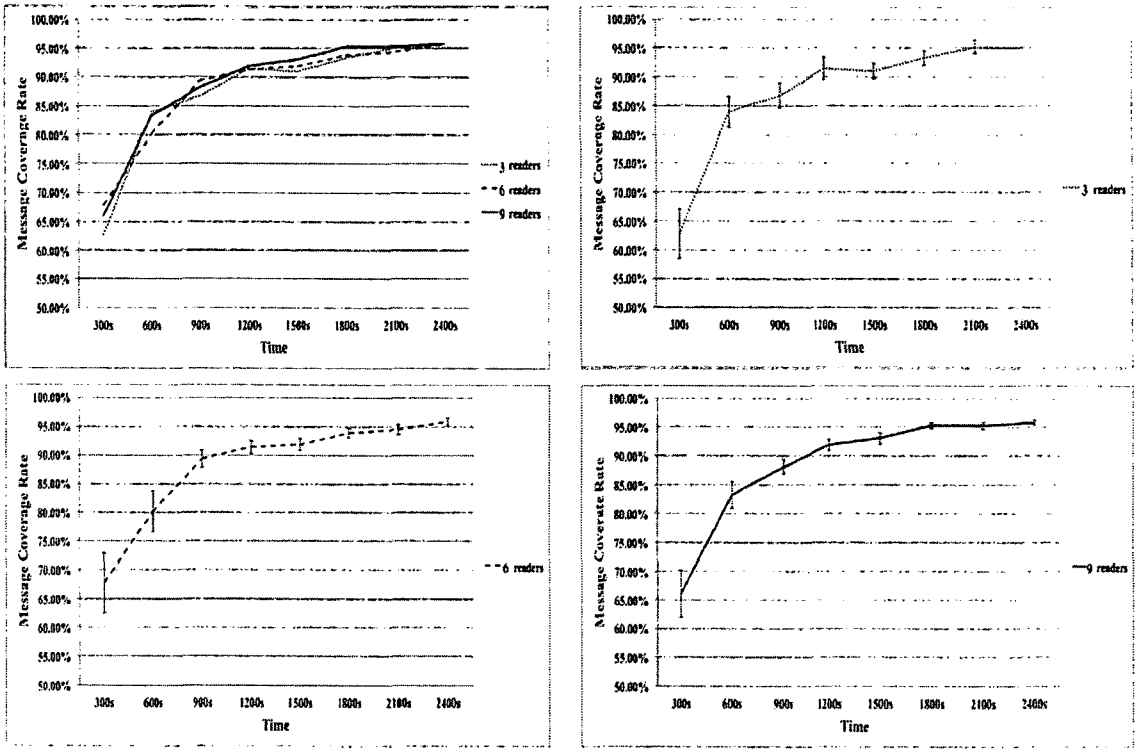

Figure 17: 100% message generation - Message Coverage Rate with the 90% confidence interval.

As it is shown in Figures 17 and 18, 90% of messages are synchronized among all readers after 20 to 30 minutes. Moreover, different numbers of reader holders in the environment have similar performance to each other. The high percentage of synchronized messages remains stable between 90% to 95% after 45 minutes.

Figure 19 and 20 show the influence on the *Message Update Success Rate* between different message TTLs, reader populations and message traffic. It turns out that when the TTL is set to 15 minutes, some reader messages fail to update reader's local records before those messages are dropped. This simulation shows that the higher the *Message Coverage Rate* is, the higher the *Message Update Success Rate* will be.

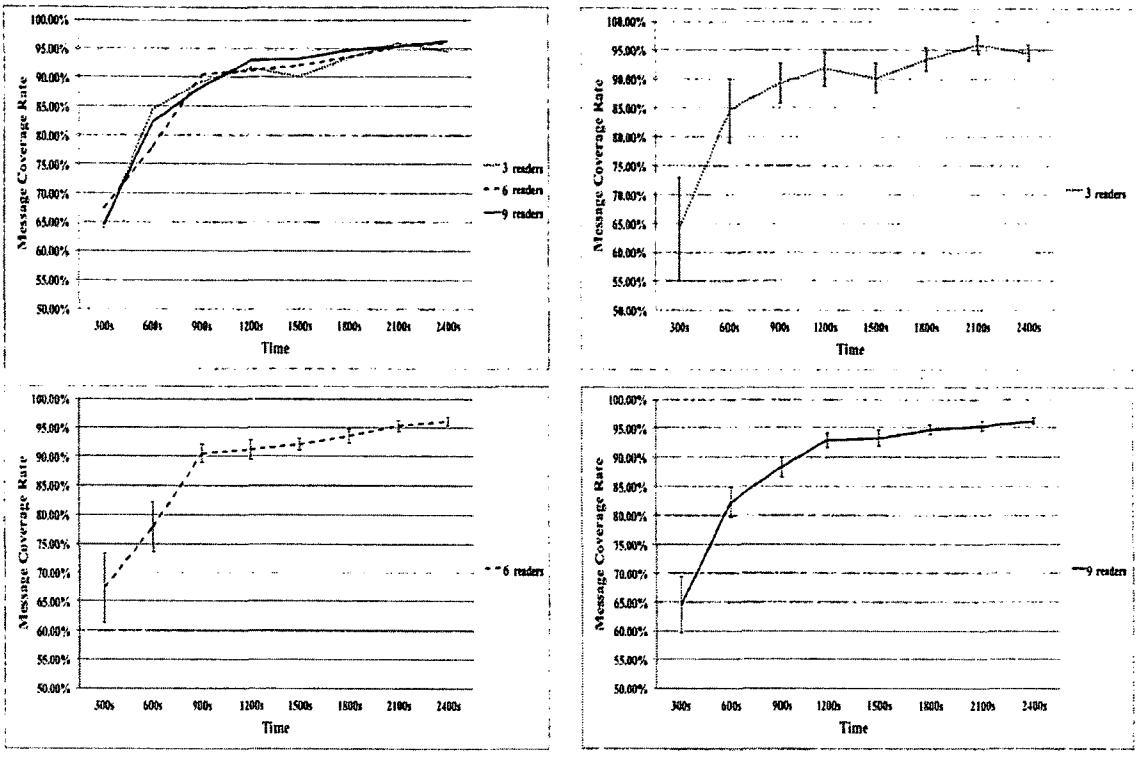

Figure 18: 50% message generation - Message Coverage Rate with the 90% confidence interval.

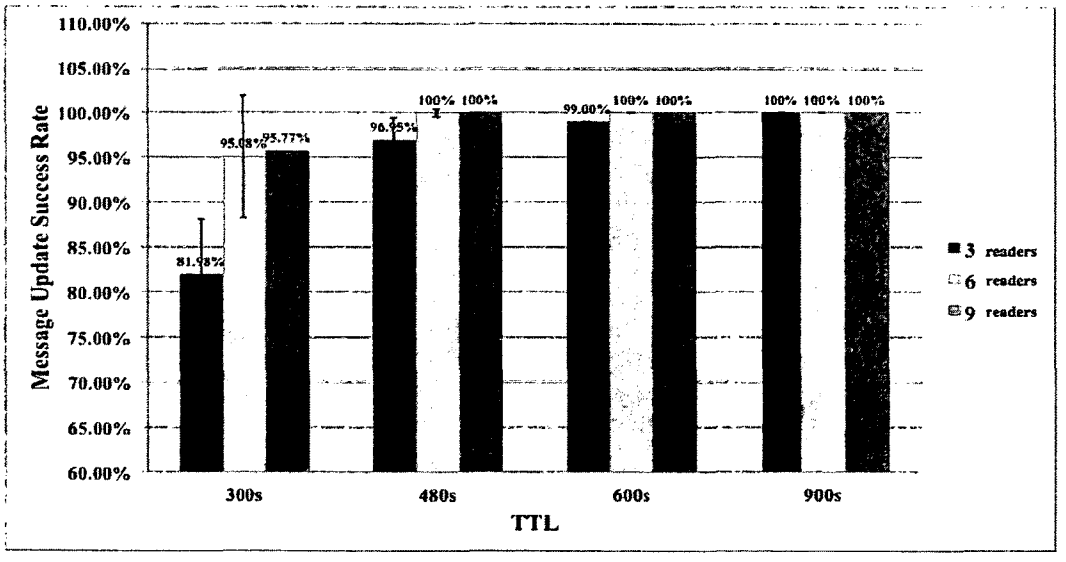

Figure 19: 100% message generation - Message Update Success Rate with the 90% confidence interval.

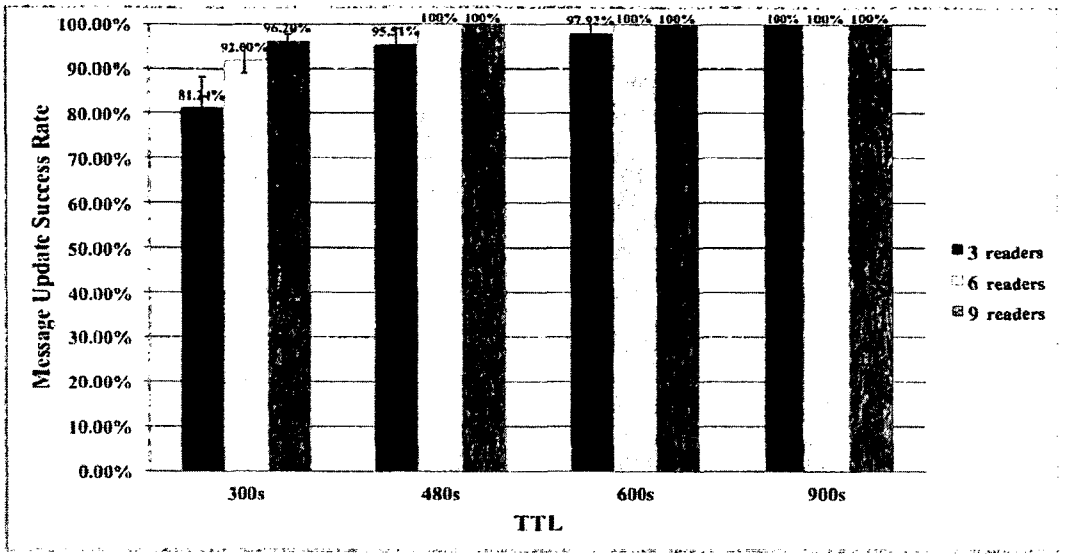

Figure 20: 50% message generation - Message Update Success Rate with the 90% confidence interval.

Based on our simulation results differences in the number of messages and reader holders do not significantly affect the *Message Update Success Rate* and *Message Coverage Rate*. The *Message Update Success Rate* mainly depends on the TTL. The longer the TTL, the higher the possibility that a message can be sent to all readers. Figures 19 and 20 indicate that it is safe to set the TTL to 30 minutes to ensure 100% success rate.

The next scenario uses a food warehouse to show the influence of different map sizes. Figure 21 shows a layout of a warehouse. Figure 22 shows the route that reader holders walk. The warehouse size is three times of the size of the supermarket. In Figure 22, a circle represents the transmission range of a node. Each square represents a reader.

Simulation parameters of this warehouse scenario are the same as the supermarket, with the only difference being the size and layout of the map. Figure 23 shows the total number of messages generated in three, six and nine readers scenarios.

Since the warehouse size is three times the one of the supermarket, readers have less

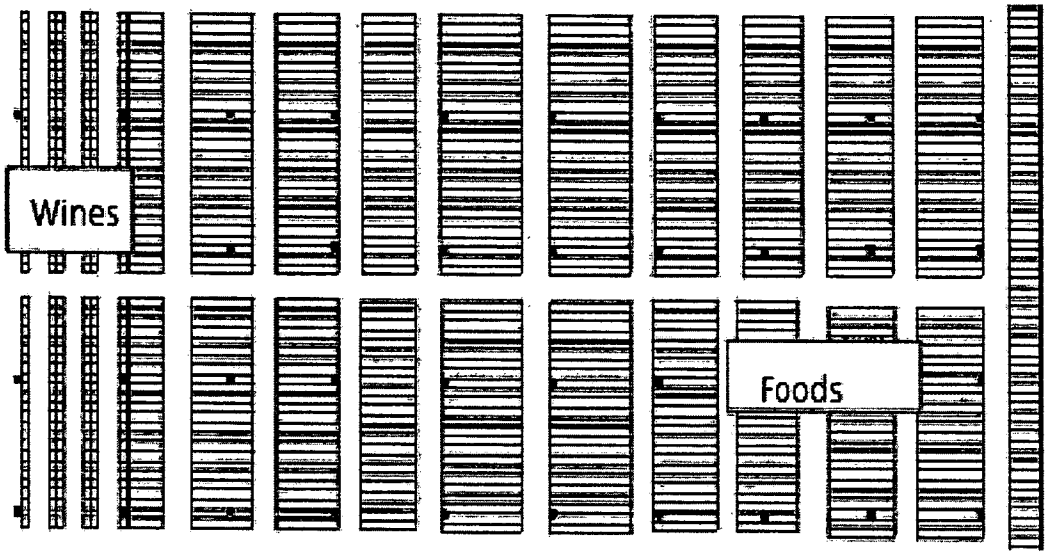

Figure 21: The map of a warehouse.

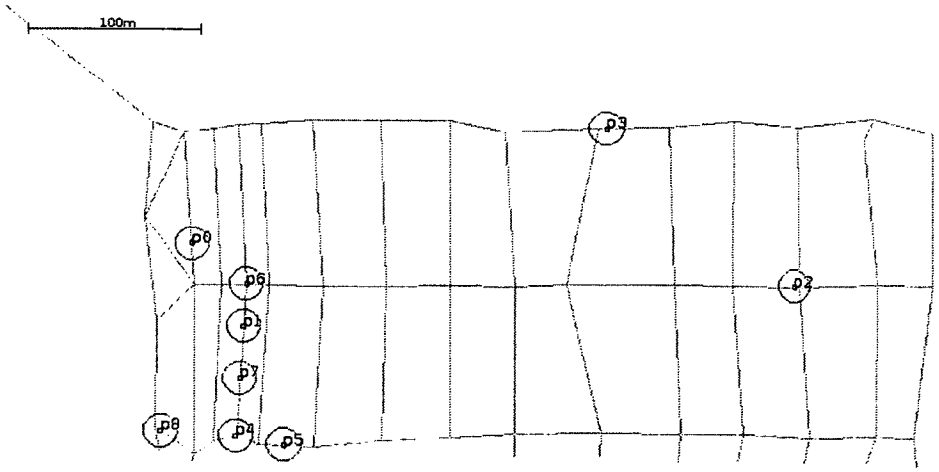

Figure 22: The route map.

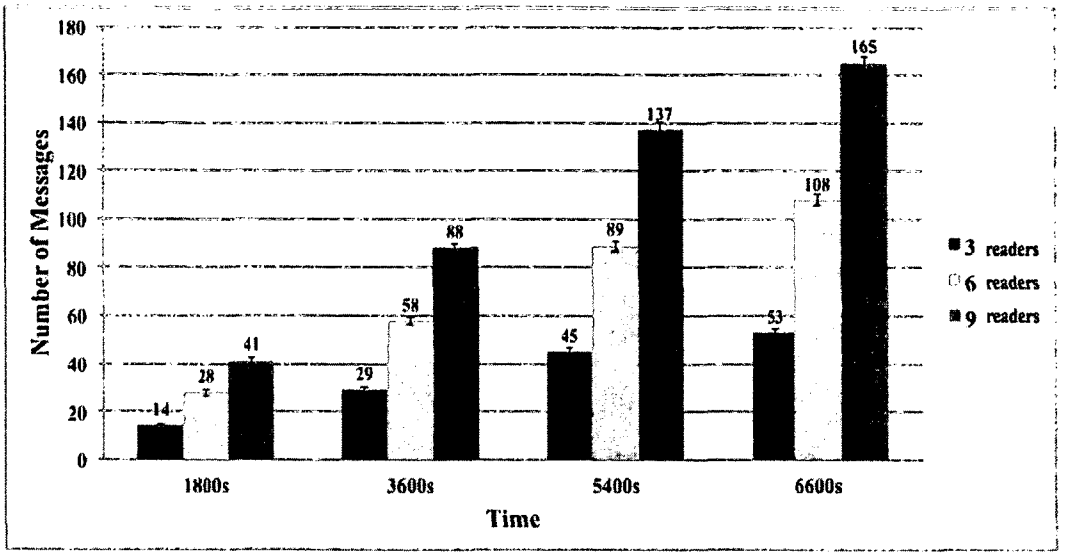

Figure 23: The total number of generated messages with the 90% confidence interval.

possibility to communicate with each other for a same time period and same node population. A longer time period is used in this simulation to show the result of *Message Coverage Rate* in Figure 24 and the result of *Message Update Success Rate* in Figure 25.

Similar to the supermarket simulation, the trend of the *Message Update Success Rate* and *Message Coverage Rate* starts with low rates and moves to higher rates. The larger map size slows the overall trend, therefore readers need a longer TTL to achieve the 100% *Message Update Success Rate*. If more readers are placed in the simulation, more possibilities are provided for communication. Therefore shorter TTLs are needed to achieve the 100% *Message Update Success Rate*. Figure 24 shows that the *Message Coverage Rate* in a nine node scenario is higher than in both six and three node scenarios; while the six node scenario *Message Coverage Rate* is higher than the three node scenario. Also from Figure 25, nine and six node scenarios can reach 100% *Message Update Success Rate* with a TTL of 3600 seconds while the three node scenario needs 5400 seconds.

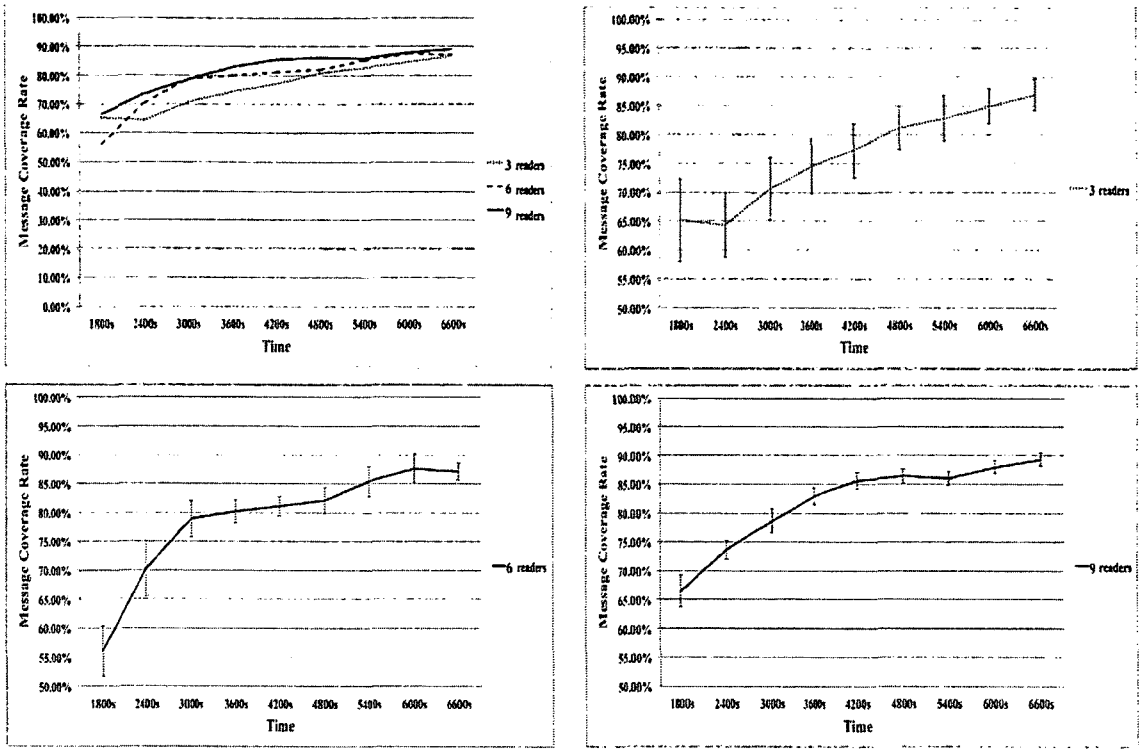

Figure 24: Message Coverage Rate with the 90% confidence interval.

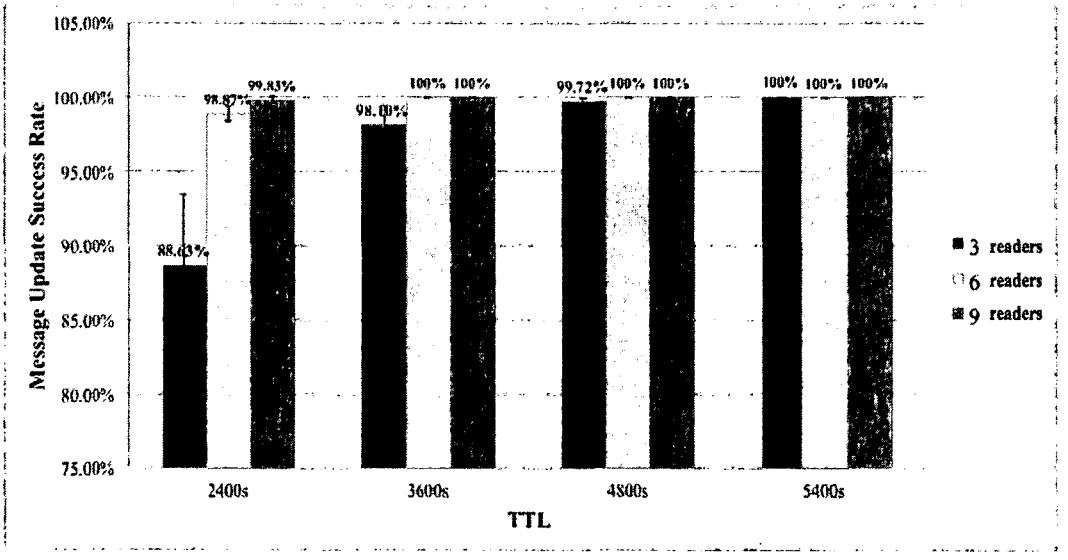

Figure 25: Message Update Success Rate with the 90% confidence interval.

5.4 Simulation conclusion

Readers in the DTN based RFID protocol can connect with each other and exchange messages in a DTN region. By assigning proper TTLs, it is possible to ensure that messages reach and update all readers before they are dropped. From the simulation, we may conclude that the longer the TTL, the higher the possibility that a message can be received and updates reader local records before it is dropped. Also, the more readers in the DTN, the shorter TTL is needed to arrive at a 100% *Message Update Success Rate*. In case of a smaller size scenario map, the number of readers does not influence the result significantly. For a larger size scenario map, more readers in the DTN provide more connection possibilities and therefore a higher *Message Coverage Rate* is achieved.

Chapter 6

Conclusion

In terms of ensuring the security of lightweight RFID tags, one of the difficulties is to meet security requirements within limited resources. Based on previous RFID studies, the DTN based RFID protocol has been proposed to ensure mutual verification, untraceability, forward security and backward security without permanent backend server connections. The *ONE* simulator has been used to evaluate the RFID reader performance in a DTN.

6.1 The DTN based RFID protocol

The DTN based RFID protocol enhances the security of the *EPCglobal* inventory protocol. Readers and tags share the same *PRNG* state which includes the same *PRNG* function and *PRNG* seed to generate an identical number sequence for mutual verification. In Chapter 2, strategies are provided to resolve de-synchronizations and collisions. When a de-synchronization occurs, the reader and the tag are resynchronized before starting another inventory round for mutual verification. To solve a collision situation when several tags have the same *TDS* simultaneously, the reader requires one of the collided tags to temporally update its state and send new identification back. Then the reader is able to differentiate collided tags and to start a new inventory with the correct tag for mutual verification.

A DTN is used to avoid permanent connections with a backend database or server. The *ONE* simulator is used to evaluate the DTN's performance for RFID readers. The simulator *ONE*, illustrating a supermarket and a warehouse, shows how workers check and update product information by using readers. Reader holders walk randomly through the store and warehouse. A message is generated to update readers local records every time a holder stops and scans a product. When two readers are within each other's transmission range, new messages are broadcasted. Readers local records are updated by new messages and TTL expired messages are discarded. The simulation results imply that a proper TTL value is important for all readers to synchronize new messages and update their local records before messages are dropped.

6.2 Performance comparison

Table 8 compares performances of the Gossamer, Flyweight and DTN based RFID protocol.

6.2.1 Computation cost

The DTN based RFID protocol only uses the XOR and $PRNG$ functions, which are compatible with the *EPCglobal* standard. The Gossamer protocol does not need to implement the $PRNG$ function to generate random numbers, but it does require tags to use new functions ROT and $MIXBITS$. As for the Flyweight protocol, the use of XOR operation and $PRNG$ function are supported but they still require a new secure function to update the $PRNG$ seed for both a reader and a tag.

Table 8: Performance comparison.

Performance	**Gossamer**	**Flyweight**	**DTN based RFID protocol**
Resynchronization	Yes	Yes	Yes
Data Confidentiality	Yes	Yes	Yes
Untraceability	Yes	Yes	Yes
Mutual Verification	Yes	Yes	Yes
Forward Security	Yes	Yes	Yes
Backward Security	NA	Yes	Yes
Communication Cost (bits)	*424*	*64* Or *80*	*66*
Memory Size on Tag (bits)	*21*32*	*3*32+1*	*2.5*32*
Memory Size for each Tag on Reader/Database (bits)	*12*32*	*6*32+1*	*5*32*
Functions and Operations	$\oplus$,+, *ROT*, *MIXBITS*	$\oplus$,*PRNG*, seed update function	$\oplus$,*PRNG*
Permanent backend server connections	Yes	Yes	No

6.2.2 Communication cost

For the cost of communication, the Gossamer protocol uses a total of 424 bits in four messages during the transmission for compatibility with the *EPCglobal* standard. The Flyweight protocol uses 64 bits in four messages in total when a tag and a reader are synchronized. If the tag and the reader are de-synchronized, then five messages are sent with a total of 80 bits. Moreover, another 32-bit random number is sent when the server updates a *PRNG* seed. In the DTN based RFID protocol, four messages are transmitted, 66 bits in total.

6.2.3 Storage cost

To store all the necessary information of tags and readers, the Gossamer protocol requires a total of 1,056 bits for a tag along with its correlated information which is stored in the server while the flyweight protocol requires 290 bits. In the DTN based RFID protocol, a reader keeps 160 bits for one tag while a tag stores 80 bits locally and uses TID as TD. Thus, 240 bits in total.

List of References

[1] Gildas Avoine, Xavier Carpent, and Benjamin Martin. Strong Authentication and Strong Integrity (SASI) is not that strong. In SiddikaBerna Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 50–64. Springer Berlin Heidelberg, Jun. 2010.

[2] Michel Barbeau. Passive RFID tag security. In *Haking pratical protection IT security magazine*, volume 6, pages 30–37. Software Press Sp. z o.o. SK 02-682 Warszawa, ul. Bokserska 1, Aug. 2011.

[3] Mike Burmester and Breno Medeiros. The Security of EPC Gen2 Compliant RFID Protocols. In StevenM. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 490–506. Springer Berlin Heidelberg, 2008.

[4] Mike Burmester, Breno Medeiros, Jorge Munilla, and Alberto Peinado. Secure EPC Gen2 Compliant Radio Frequency Identification. In PedroM. Ruiz and JoseJoaquin Garcia-Luna-Aceves, editors, *Ad-Hoc, Mobile and Wireless Networks*, volume 5793 of *Lecture Notes in Computer Science*, pages 227–240. Springer Berlin Heidelberg, 2009.

[5] Mike Burmester and Jorge Munilla. A Flyweight RFID Authentication Protocol. In *Workshop on RFID Security – RFIDSec'09*, Leuven, Belgium, Jul. 2009.

[6] Chin-Ling Chen and Yong-Yuan Deng. Conformation of EPC Class 1 Generation 2 standards RFID system with mutual authentication and privacy protection. *Engineering Applications of Artificial Intelligence*, 22(8):1284 – 1291, Dec. 2009.

[7] Hung-Yu Chien. SASI: A New Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity. *Dependable and Secure Computing, IEEE Transactions on*, 4(4):337 –340, Oct.–Dec. 2007.

[8] Hung-Yu Chien and Che-Hao Chen. Mutual authentication protocol for RFID conforming to EPC Class 1 Generation 2 standards. *Computer Standards Interfaces*, 29(2):254 – 259, Feb. 2007.

[9] Don Coppersmith, Hugo Krawczyk, and Yishay Mansour. The Shrinking Generator. In Stinson, DouglasR., editor, *Advances in Cryptology – CRYPTO' 93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer Berlin Heidelberg, 1994.

[10] EPC Global. EPCglobal Architecture Framework. `http://www.gs1.org/gsmp/kc/epcglobal/architecture`, 2010.

[11] EPC Global. Class 1 Generation 2 UHF Air Interface Protocol Standard. `http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2`, 2011.

[12] EPC Global. EPC Radio-Frequency Identity Protocols EPC Class-1 HF RFID Air Interface Protocol. `http://www.gs1.org/gsmp/kc/epcglobal/hf`, 2011.

[13] EPC Global. EPC Tag Data Standard (TDS). `http://www.gs1.org/gsmp/kc/epcglobal/tds/`, 2011.

[14] EPC Global. EPC Tag Data Translation Standard (TDT). `http://www.gs1.org/gsmp/kc/epcglobal/tdt/`, 2011.

[15] EPC Global. EPCglobal Standards. `http://www.gs1.org/gsmp/kc/epcglobal`, 2011.

[16] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[17] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, Aug. 1986.

[18] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Cryptanalytic Attacks on Pseudorandom Number Generators. In Vaudenay, Serge, editor, *Fast Software Encryption*, volume 1372 of *Lecture Notes in Computer Science*, pages 168–188. Springer Berlin Heidelberg, 1998.

[19] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Proceedings of the 2nd International Conference on Simulation Tools*

and Techniques, Simutools '09, pages 55:1–55:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[20] H.R. Lee and D.W. Hong. The tag authentication scheme using self–shrinking generator on RFID system. *Transactions on Engineering, Computing, and Technology*, 18:52–57, Dec. 2006.

[21] Willi Meier and Othmar Staffelbach. The self-shrinking generator. In Alfredo Santis, editor, *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer Berlin Heidelberg, 1995.

[22] Pedro Peris-Lopez, Julio C. Hernandez-Castro, JuanM Estevez-Tapiador, and Jan C.A. van der Lubbe. Cryptanalysis of an EPC class-1 generation-2 standard compliant authentication protocol. *Engineering Applications of Artificial Intelligence*, 24(6):1061–1069, Sep. 2011.

[23] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda. LAMED - A PRNG for EPC Class-1 Generation-2 RFID specification. *Comput. Stand. Interfaces*, 31(1):88–97, Jan. 2009.

[24] Pedro Peris-Lopez, JulioCesar Hernandez-Castro, JuanM. Estevez-Tapiador, and Arturo Ribagorda. EMAP: An efficient mutual-authentication protocol for low-cost RFID tags. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4277 of *Lecture Notes in Computer Science*, pages 352–361. Springer Berlin Heidelberg, 2006.

[25] Pedro Peris-Lopez, JulioCesar Hernandez-Castro, JuanM. Estevez-Tapiador, and Arturo Ribagorda. LMAP: A real lightweight mutual authentication protocol for low–cost RFID tags. In *In: Proc. of 2nd Workshop on RFID Security*, page 06. Ecrypt, Jul. 2006.

[26] Pedro Peris-Lopez, JulioCesar Hernandez-Castro, JuanM. Estevez-Tapiador, and Arturo Ribagorda. M^2AP: a minimalist mutual-authentication protocol for low-cost RFID tags. In Jianhua Ma, Hai Jin, LaurenceT. Yang, and JeffreyJ.-P. Tsai, editors, *Ubiquitous Intelligence and Computing*, volume 4159 of *Lecture Notes in Computer Science*, pages 912–923. Springer Berlin Heidelberg, 2006.

[27] Pedro Peris-Lopez, JulioCesar Hernandez-Castro, JuanM. Estevez-Tapiador, and Arturo Ribagorda. Cryptanalysis of a novel authentication protocol conforming to EPC-C1G2 standard. *Computer Standards Interfaces*, 31(2):372 - 380, Feb. 2009.

[28] Pedro Peris-Lopez, JulioCesar Hernandez-Castro, JuanM.E. Tapiador, and Arturo Ribagorda. Advances in ultralightweight cryptography for low-cost RFID tags: Gossamer protocol. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *Lecture Notes in Computer Science*, pages 56–68. Springer Berlin Heidelberg, 2009.

[29] R.C.-W. Phan. Cryptanalysis of a New Ultralightweight RFID Authentication Protocol x02014;SASI. *Dependable and Secure Computing, IEEE Transactions on*, 6(4):316 –320, Oct.–Dec. 2009.

[30] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, WDTN '05, pages 252–259, New York, NY, USA, 2005. ACM.

[31] A. Vahdat, D. Becker, et al. Epidemic routing for partially connected ad hoc networks. Technical report, Technical Report CS-2000-06, Duke University, 2000.