

vrije Universiteit amsterdam



MASTER THESIS

RFID Guardian Back-end Security Protocol

Author:
Hongliang WANG

First Reader:
Bruno CRISPO
Second Reader:
Melanie REIBACK

Department of Computer Science
Vrije Universiteit, Amsterdam
The Netherlands

February 17, 2008

Abstract

Radio Frequency Identification (RFID) systems have become popular for automated identification and supply chain applications. Due to resource constraints, RFID tags have potential security and privacy problems. However, rather than relying on public RFID readers to enforce privacy protection, consumers might instead carry their own privacy-enforce devices for RFID, i.e. RFID Guardian. RFID Guardian acts as a kind of personal RFID firewall. As a high-powered device with substantive computing power, a Guardian can implement sophisticated privacy policies and can use channels other than RF (e.g. GPS or Internet connections) to supplement ambient data.

In this thesis, I focus on the interaction between RFID Guardian and RFID readers and will describe the back-end security protocol of RFID Guardian, including:

- Access Control List
- Authentication Protocols, notably asymmetric and symmetric key protocols

After that, I will show the result of 35 test cases on the RFID Guardian software to the readers.

Acknowledgements

I would like to thank *dr.* Bruno Crispo as my supervisor and *drs.* Melanie Rieback for her direct supervision. I would also like to appreciate *dr.* Rutger Hoffman for his help as the scientific programmer.

Table of Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 1.1 | Radio Frequency Identification | 2 |
| 1.2 | RFID Guardian | 3 |
| 1.3 | Organization | 4 |
| 2 | RFID Guardian Usage | 5 |
| 2.1 | General System Model | 5 |
| 2.2 | RFID Guardian Access Control List | 7 |
| 2.2.1 | Language | 7 |
| 2.2.2 | Example ACL | 9 |
| 2.3 | List of Usage Scenarios | 15 |
| 2.3.1 | Contact-less Smart Card | 16 |
| 2.3.2 | Supermarket | 21 |
| 2.3.3 | Library | 26 |
| 2.3.4 | E-Passport | 35 |
| 2.3.5 | Car Key-less Entry Systems | 40 |
| 2.3.6 | Home Appliance | 44 |
| 2.3.7 | Animal Identification | 50 |
| 2.3.8 | Supply Chain | 54 |
| 2.3.9 | Casino | 64 |
| 2.3.10 | Waste Management | 68 |
| 2.4 | Summary | 72 |

| | | |
|----------|---|------------|
| 3 | Authentication | 75 |
| 3.1 | Introduction | 76 |
| 3.1.1 | Asymmetric Key Protocols | 76 |
| 3.1.2 | Symmetric Key Protocols | 79 |
| 3.2 | Technique Discussion | 81 |
| 3.2.1 | Asymmetric vs Symmetric Key Protocols for RFID Guardians | 81 |
| 3.2.2 | Key Generation | 82 |
| 3.2.3 | Certificate Authority in RFID Systems | 84 |
| 3.2.4 | Certificate Validation in RFID Guardian | 85 |
| 3.2.5 | Certificate Revocation in RFID Guardian | 86 |
| 3.3 | Authentication in our Usage Scenarios | 87 |
| 3.3.1 | Contactless Smart Card | 87 |
| 3.3.2 | Supermarket | 88 |
| 3.3.3 | Library | 88 |
| 3.3.4 | E-passport | 88 |
| 3.3.5 | Car Keyless Entry System | 88 |
| 3.3.6 | Home Appliance | 89 |
| 3.3.7 | Animal Identification | 89 |
| 3.3.8 | Military and Commercial Supply Chains | 89 |
| 3.4 | Conceptual User Interface | 90 |
| 3.5 | Conclusion | 96 |
| 4 | Software Regression Test | 97 |
| 4.1 | Introduction | 97 |
| 4.2 | Correctness and Robustness | 98 |
| 4.2.1 | Framework | 98 |
| 4.2.2 | Test Cases | 100 |
| 4.2.3 | How to add a new test? | 106 |
| 4.3 | Efficiency | 107 |
| 4.4 | Summary | 109 |
| 5 | Conclusion | 111 |

List of Figures

| | | |
|------|--|----|
| 1.1 | An RFID tag used for Wal-Mart | 2 |
| 1.2 | RFID Reader | 2 |
| 1.3 | RFID Guardian Prototype | 4 |
| 2.1 | UML Object Model: RFID Guardian | 6 |
| 2.2 | UML Sequence Model: RFID Guardian | 7 |
| 2.3 | RFID Guardian Reader File Example | 11 |
| 2.4 | RFID Guardian Tag File Example | 11 |
| 2.5 | RFID Guardian Reader File Correct Form | 11 |
| 2.6 | UML Object Model: Credit Card | 17 |
| 2.7 | UML Sequence Model: Credit Card | 17 |
| 2.8 | RFID Guardian Reader File: Credit Card | 19 |
| 2.9 | RFID Guardian Tag File: Credit Card | 19 |
| 2.10 | UML Object Model: Supermarket | 21 |
| 2.11 | UML Sequence Model: Supermarket | 22 |
| 2.12 | RFID Guardian Reader File: Supermarket | 23 |
| 2.13 | RFID Guardian Tag File: Supermarket | 24 |
| 2.14 | Library Inventory | 27 |
| 2.15 | Auto Return Device | 27 |
| 2.16 | Automated Conveyor System | 28 |
| 2.17 | Example Anti-collision Inventory Request | 28 |
| 2.18 | UML Object Model: Library | 29 |
| 2.19 | UML Sequence Model: Library | 30 |
| 2.20 | RFID Guardian ACL: Library | 31 |
| 2.21 | RFID Guardian Reader File: Library | 32 |

| | | |
|------|--|----|
| 2.22 | RFID Guardian Tag File: Library | 32 |
| 2.23 | UML Object Model: E-Passport | 36 |
| 2.24 | UML Sequence Model: Extended Access Control | 36 |
| 2.25 | RFID Guardian ACL: E-Passport | 37 |
| 2.26 | RFID Guardian Reader File: E-Passport | 38 |
| 2.27 | RFID Guardian Tag File: E-Passport | 38 |
| 2.28 | UML Object Model: Car Key-less Entry System | 41 |
| 2.29 | UML Sequence Model: Car Key-less Entry System | 41 |
| 2.30 | RFID Guardian ACL Rule: Car Key-less Entry System | 42 |
| 2.31 | RFID Guardian Reader File: Car Key-less Entry System | 43 |
| 2.32 | RFID Guardian Tag File: Car Key-less Entry System | 43 |
| 2.33 | UML Object Model: Home Appliance | 46 |
| 2.34 | UML Sequence Model: Home Appliance | 46 |
| 2.35 | RFID Guardian ACL Rule: Home Appliance | 47 |
| 2.36 | RFID Guardian Reader File: Home Appliance | 48 |
| 2.37 | RFID Guardian Tag File: Home Appliance | 48 |
| 2.38 | UML Object Model: Animal Identification | 51 |
| 2.39 | UML Sequence Model: Animal Identification | 51 |
| 2.40 | RFID Guardian ACL Rule: Animal Identification | 52 |
| 2.41 | RFID Guardian Reader File: Animal Identification | 52 |
| 2.42 | RFID Guardian Tag File: Animal Identification | 53 |
| 2.43 | Military Tag | 55 |
| 2.44 | Military Container | 55 |
| 2.45 | UML Object Model: Military Container Shipment | 55 |
| 2.46 | UML Sequence Model: Military Container Shipment | 56 |
| 2.47 | RFID Guardian ACL Rule: Military Container Shipment | 57 |
| 2.48 | RFID Guardian Reader File: Military Container Shipment | 58 |
| 2.49 | RFID Guardian Tag File: Military Container Shipment | 58 |
| 2.50 | UML Object Model: Business Usage | 60 |
| 2.51 | UML Object Model: Business Usage | 60 |
| 2.52 | RFID Guardian Reader File: Business Usage | 62 |
| 2.53 | RFID Guardian Tag File: Business Usage | 62 |
| 2.54 | UML Object Model: Casino | 65 |

| | | |
|------|--|-----|
| 2.55 | UML Sequence Model: Casino | 65 |
| 2.56 | RFID Guardian ACL Rule: Casino | 66 |
| 2.57 | UML Object Model: Waste Management | 69 |
| 2.58 | UML Object Model: Waste Management | 69 |
| 2.59 | RFID Guardian ACL Rule: Waste Management | 70 |
| 2.60 | Range of a personal Guardian | 73 |
| 2.61 | Range of Guardians for a truck | 73 |
| 3.1 | UML Sequence Model Sample: Credit Card | 75 |
| 3.2 | Centralized CA Sample: Smart Card | 84 |
| 3.3 | Local CA Sample: Library | 85 |
| 4.1 | Comparison Result: Preprocessing vs. Non-Preprocessing . . | 108 |

List of Tables

| | | |
|------|---|-----|
| 1.1 | RFID tag and communication range | 3 |
| 2.1 | RFID Guardian Query Result Example | 14 |
| 2.2 | Different Context will decide different result | 15 |
| 2.3 | RFID Guardian Query Result: Credit Card | 20 |
| 2.4 | RFID Guardian Query Result: Supermarket | 26 |
| 2.5 | RFID Guardian Query Result: Library (Lending Phase) . . . | 34 |
| 2.6 | RFID Guardian Query Result: Library (Return Phase) . . . | 35 |
| 2.7 | RFID Guardian Query Result: E-Passport | 40 |
| 2.8 | RFID Guardian Query Result: Car Key-less Entry System . . | 44 |
| 2.9 | RFID Guardian Query Result: Home | 50 |
| 2.10 | RFID Guardian Query Result: Animal Identification | 54 |
| 2.11 | RFID Guardian Query Result: Military Container Shipment . | 59 |
| 2.12 | RFID Guardian Query Result: Business Usage | 64 |
| 2.13 | RFID Guardian Query Result: Casino | 68 |
| 2.14 | RFID Guardian Query Result: Waste Management | 71 |
| 4.1 | Testing Environment | 98 |
| 4.2 | Software Parameter and Usage | 99 |
| 4.3 | Test Suites | 100 |
| 4.4 | Comparison Result: Preprocessing vs. Non-preprocessing . . | 108 |

Chapter 1

Introduction

Radio Frequency Identification (RFID) is a technology used for automatic identification of persons and objects. It is considered to be the next generation for bar-codes. RFID tags are becoming popular tools for identification of products in various areas, i.e. supply chain management. In November 2003, Phillips Corporation announced that it has shipped more than a billion RFID devices worldwide.

However, more and more people are concerning about the privacy problems of RFID. To be precise, RFID tags may leak its holder's position, which is a *tracking* threat. RFID tags respond to reader interrogation without alerting their owners or bearers. Thus, where read range permits, clandestine scanning of tags is a plausible threat. Since most RFID tags emit unique identifiers, a person carrying an RFID tag effectively broadcasts a fixed serial number to nearby readers.[3]

Moreover, the authentication protocol between RFID tags and readers is not strong because of the resource constraints. Typically, RFID tags can only store hundreds of bits and have only 5K - 10K logic gates. Within this number of gates, only between 250 and 3000 gates can be devoted to security functions. It is interesting to recall that for a standard implementation of Advanced Encryption Standard (AES), 20K - 30K gates are needed.[3] The result of this weak authentication scheme is that tags may send signals to illegitimate readers.

In order to solve these problems several techniques have been proposed, including relabelling, killing and sleeping, etc. In 2004, Melanie Rieback from Vrije Universiteit proposed the idea of using an active device called RFID Guardian to secure the communicate between RFID readers and tags. As a high-powered device with substantive computing power a Guardian can implement sophisticated privacy policies, and can use channels other than RF (e.g., GPS or Internet connections) to supplement ambient data.[6]

In this thesis, I will investigate the possibilities of using RFID Guardian in real life. I will convince the readers with proof from both theory and practice that RFID Guardian is indeed a successful solution to their privacy problems. I will explain how Access Control List works, and also analyze how a Guardian can establish (mutual) authentication with RFID readers — RFID Guardian ensures only the correct RFID readers query the correct RFID tags at correct time. Then I will tell the readers how the software in RFID Guardian can implement this behavior.

This chapter is organized as follows: I will review RFID and RFID Guardian technology respectively in section 1.1 and 1.2. After that I will present the organization of the whole thesis in section 1.3.

1.1 Radio Frequency Identification

RFID was first used during the World War II in “Identification Friend or Foe” systems onboard military aircraft by British Royal Air Force. Soon after that, Harry Stockman demonstrated a system energized completely by reflected power. The first Electronic Article Surveillance anti-theft systems were commercialized in 1960s. In the 1970s, the US Department of Energy investigated the technology’s potential to safeguard materials at nuclear weapons sites.

Modern RFID tags represent a culmination of the evolution toward wireless infrastructure and low-cost embedded computers. An RFID system basically consists of transponders (tags), readers (scanners) and application systems for further processing of the acquired data.

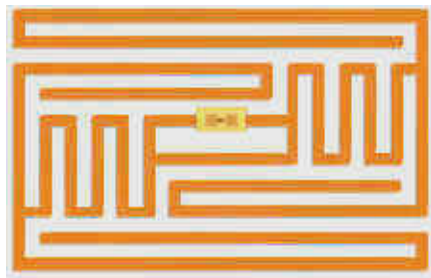


Figure 1.1: An RFID tag used for Wal-Mart



Figure 1.2: RFID Reader

RFID tags (Figure: 1.1) are now of size of a grain of rice and have built-in logic (microchip or state machine), a coupling element (analog front end with

| Type | Frequency | Range (meter) |
|----------------------|-----------------------------|---------------|
| Low Frequency | 124 kHz - 135 kHz | 0 - 1 |
| High Frequency | 13.56 MHz | 1 - 10 |
| Ultra High Frequency | 860 MHz - 960 MHz, 2.45 GHz | 10 - 100 |
| Active | - | > 100 |

Table 1.1: RFID tag and communication range

antenna) and memory (pre-masked or electronically erasable-programmable read-only memory). The coil-on-chip technology allows very small tags with only $6mm$ diameter and $1.5mm$ thickness. These tags can be passive, active or semi-active. Passive tags do not possess an internal energy source while semi-active tags carry a battery only for internal calculation. Instead, both obtain power from the reader's electromagnetic field. Therefore, they are only able to emit radio signals after receiving request from readers. Active tags, on the other hand, can send out signals without external help from readers, because they carry battery power.[1] [2] The communication range of passive tags depends on the frequency it is using, which are shown Table 1.1. The price of an RFID tag depends on its type and ranges from 1 to 20 US cents. Generally speaking, tags with a price under 5 cents are called low-cost tags and will not carry power supply. In this paper, I only address low-cost tags.

RFID readers (Figure 1.2) are generally composed of a computer and a radio. The computer manages communications with the network, allowing tag data to be communicated to back-end application systems. The radio controls communication with the tag, typically using a language dictated by a published protocol such as EPC Class 1 specification. These readers could be either fixed or in-hand. The communication range of a reader is expected to be as long as 100 meters, much larger than that of a tag.

Application systems are made up of back-end database and application-specific software. The back-end database contains information of RFID tags and readers, i.e. cryptographic keys and identifiers. Readers may use these information to contact and identify tags. The software is used to facilitate users, for instance an ERP system.

1.2 RFID Guardian

RFID Guardian is a portable battery-powered device that mediates interaction between RFID readers and RFID tags. As can be seen from the picture 1.3 it is almost as large as a PDA, so users can easily carry a Guardian on his belt. Therefore RFID Guardian could serve as a "personal RFID firewall" by a user walking on the street, shopping in a supermarket, etc.

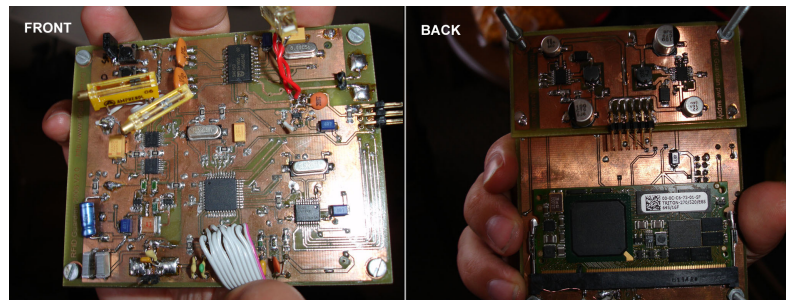


Figure 1.3: RFID Guardian Prototype

In terms of hardware, RFID Guardian can be regarded as a fully-fledged portable computer. In order to perform computation tasks, it has an Intel XScale PXA270 processor, with 64M SDRAM and 16M flash memory. In order to talk to both RFID readers and RFID tags, RFID Guardian also has a reader transmitter/receiver and a tag transmitter/receiver.

In terms of software, RFID Guardian has a device driver, a protocol stack, data storage libraries, high-level system tasks, and application libraries. The operating system is e-Cos Real-Time Operating System (RTOS).

In terms of functionality, RFID Guardian could reads and executes Access Control List, provide off-tag authentication, audits tags within its proximity, tracks RFID tags ownership and switches between different contexts.

To summarize a RFID Guardian is a solution to RFID security and privacy.

1.3 Organization

This thesis is organized as follows: Chapter 2 describes Access Control List policies for RFID Guardian. In chapter 3, the authentication scheme is discussed, and the software regression test is presented in chapter 4. Finally, the author will conclude in chapter 5.

Chapter 2

RFID Guardian Usage

Radio Frequency Identification (RFID) is one of today's fastest growing techniques. It is believed to be the next generation of bar-code, for its contact-less feature and does not require "line of sight". In this chapter I will investigate and describe several applications of RFID Guardian. This is useful because although RFID technology is already widely used, little has been done with RFID Guardian to protect privacy.

This chapter is organized as follows: first a general system model will be given in section 2.1, then I will describe the grammar of Guardian Access Control List in section 2.2. After that I will provide several applications of RFID Guardians in section 2.3. The conclusion is in section 2.4.

2.1 General System Model

Generally speaking RFID readers are divided into two categories — Guardian-aware and non-Guardian-aware. The difference between these two kinds is that the former one can do (mutual) authentication with RFID Guardian and exchange keys with RFID Guardian, while the latter can not. In this paper, the focus is mainly on the first kind. But note that occasionally the Guardian will also deal with non-Guardian-aware readers and the result depends on the Guardian's "default ACL Rule" for anonymous readers. Most work relates to RFID tags that support ISO Standard 15693/14443, because this is de facto standard.

This chapter also assumes that RFID Reader will use Public Key Infrastructure to establish an SSL connection with RFID Guardian, although PKI and symmetric key system is compared in more detail in the next chapter. This is because normally RFID readers are owned by big companies and will be placed in public areas, which means that they will query hundreds or even thousands of RFID tags in one day. If some customers carry an RFID

Guardian along with their RFID tag, then one reader could meet hundreds of RFID Guardians; and that Guardian might come across several readers in one day. For example, a user may go to a bank, then enter a supermarket, and then lend a book from a library. This means that the user's Guardian will have to communicate with bank readers, supermarket readers and library readers in one day. It is therefore impossible to use symmetric key authentication in this case.

In most application scenarios, there are four parties involved: RFID Reader, RFID Middleware, RFID Tag and RFID Guardian. The interaction between these parties is shown in UML format in Figure 2.1 and 2.2. It is better to explain their relationships with more concrete examples, so no abstract description is given here.

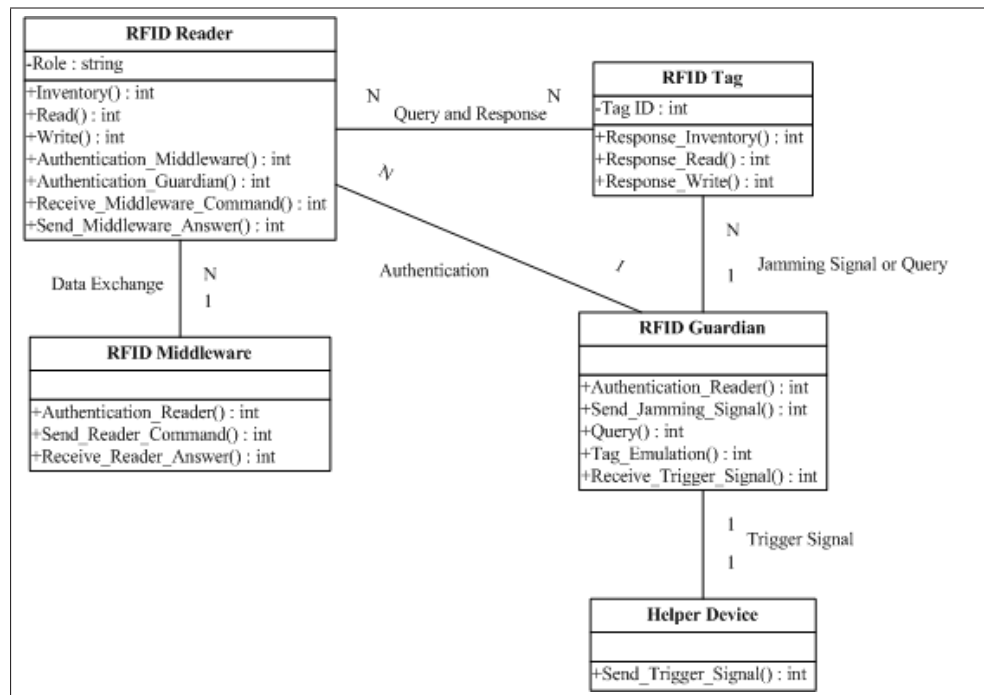


Figure 2.1: UML Object Model: RFID Guardian

Note that there is a "Helper Device" linked to the RFID Guardian in the Object Model (Figure: 2.1). This is because the RFID Guardian sometimes needs a trigger signal. For example, RFID Guardian may do queries on tags carried by the user, say every 5 minutes. This is in case an enemy puts a tag on the user's body while the user is walking on the street without the user noticing. Although RFID Guardian can protect known tags by sending jamming signals, usually it will leave the traffic alone if an unknown reader is querying an unknown tag. Therefore, the Guardian has to broadcast

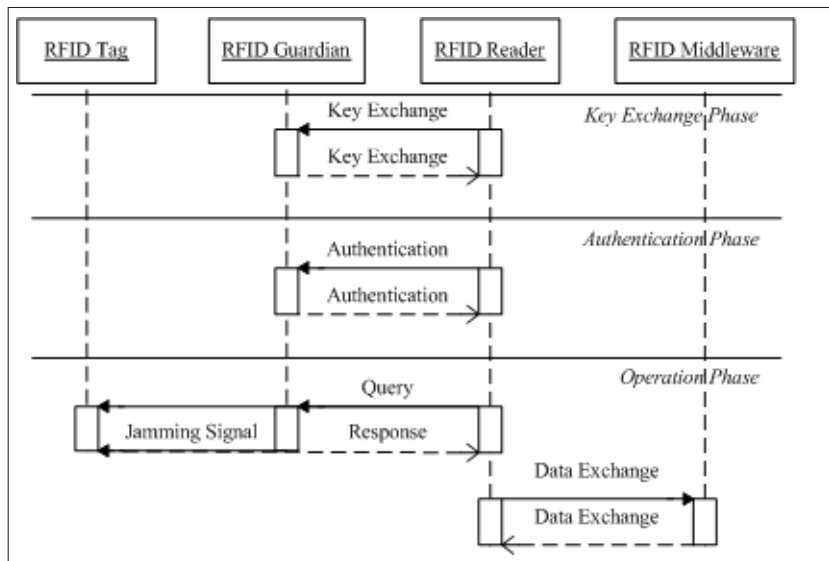


Figure 2.2: UML Sequence Model: RFID Guardian

an inventory signal every five minutes to find if some “bad” tags have been attached to the user. The user must set a timer to regularly send a triggering signal to Guardian. This timer can be set inside the Guardian itself or it can be some external devices, i.e. a blue-tooth mobile phone. Such devices do exist. But such a device is not a mandatory part of RFID Guardian, so it will be omitted from the UML Object Model in following sections.

Note that techniques used in contact-less credit card and E-Passports should be more precisely called “Contact-less smart card” instead of Radio Frequency Identification. The Smart Card Alliance even published a summary talking about the differences between these two techniques. [15] In short, the smart contact-less cards typically have more powerful CPUs and cryptographic capabilities. But from the point view of RFID Guardians, they are also just a device sending out radio signals. Therefore, from the point of view of RFID Guardians, they are just counted as RFID tags.

2.2 RFID Guardian Access Control List

2.2.1 Language

RFID Guardian Access Control List defines the actions of RFID Guardian. Users have the freedom to modify, add or remove any rules as they like. The grammar is simple.

1. Contents after a hash mark (#) are comments.
2. A rule starts with `rule $protocol ACCEPT` or `rule $protocol DENY`. The *protocol* variable should be P15693 or P14443, etc. which means the ISO standard used during communication. *ACCEPT* or *DENY* defines different actions that the user expects RFID Guardian to take.
3. The body of the rule is enclosed in a pair of braces: { and }.
4. The keyword `context` describes different modes of RFID Guardian. Users can make Guardian switch between different modes as the environment changes without editing the ACL again. For example, a user may allow an RFID reader to query her credit card when she is in a supermarket. But if she is walking on the street, she would not want the same RFID reader to query her credit card again. To achieve this, the user can simply send a signal to RFID Guardian to make it switch from “trusted” mode to “not-trusted” mode. Users must declare supported context types either at the beginning of ACL or in a separate “context file”.
5. The keyword `role` means a group of readers, so `role = PAYMENT` means the readers of PAYMENT group. The user defines these groups.
6. The keyword `tags` means a group of tags, so `tags = @MYTAGS` means the group of tags named MYTAGS. Note that the name of the tag group should start with a @ mark, but the name of a reader group should not. The aim is to make it more clearly for the user to distinguish reader groups from tag groups. So the following expressions are not correct.
 - `role = @PAYMENT` # *Reader groups never start with “@”*
 - `@role = PAYMENT` # *“role” keyword never starts with “@”*
 - `@tags= MYTAGS` # *“tags” keyword never starts with “@”*
 - `tags = MYTAGS` # *Tag groups always start with “@”*
7. Potential query content can be specified in the `query` field. For example, `command = INVENTORY` means the send command Inventory. Other content includes *mask_len* and *mask*, which is explained later.
8. Wildcard (*) means any.
9. Every line in the RULE body must end with a semicolon ;.

2.2.2 Example ACL

```
context trusted;
context home;
context private;

# Normally leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = {command = *;};
};

# Block all queries to our tags
rule P15693 DENY
{
    role = *;
    tags = @MY_TAGS;
    query = {command = *;};
};

# Allow friendly readers to do inventory queries on our tags
rule P15693 ACCEPT
{
    context = trusted;
    role = FRIENDLY_READERS;
    tags = @MY_TAGS;
    query = { command = INVENTORY; };
};

# Deny friendly readers to do queries on our secret tags
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @SECRET_TAGS;
};

# Allow trusted readers to do queries on our secret tags
rule P15693 ACCEPT
{
    context = {home, private, };
};
```

```
    role = TRUSTED_READERS;  
    tags = @SECRET_TAGS;  
};
```

The above example ACL illustrates the grammar. The document usually starts by declaring the context names, after which the main content begins. Generally speaking the access control list should start with an accept rule. This is because the RFID traffic should generally be left alone. Remember that RFID Guardian blocks queries by sending jamming signals to RFID tags, so if the first rule is to block all queries, RFID Guardian will be sending out signals whenever a reader is present, which would lead to a Denial of Service Attack (DOS Attack) on tags which are owned by other people nearby. The context mode is therefore left to “any” in the first rule.

The next rule is usually a deny rule to block unknown or illegitimate readers from accessing our own tags. And the context mode is usually set to any too.

The rest of the rules should be more specific about which kind of queries are allowed on which tags in which environment. When the context changes, legal readers may become illegal readers, so the UML Sequence Model may change because the reader can no longer retrieve information from the tags as the Guardian are sending jamming signals. But to make things simpler, only the most typical model for most UML Sequence Models are provided.

Below is an example reader file (Figure: 2.3) and tag file (Figure: 2.4), which match the ACL example above. The example shows how to group those RFID readers and tags into groups with reader files and tag files. Note that this is just an example and there is no special reason why the “Book Shelf Reader” is categorized into “Friendly Readers” but not “Trusted Readers”. For brevity the reader file does not strictly follow the grammar. The real reader file should declare readers in this way (Figure: 2.5), but it is better to delete those curly braces since there is no content inside them.

```
reader Book_Shelf;

reader Washing_Machine;

role FRIENDLY_READERS
{
    Book_Shelf,
};

role TRUSTED_READERS
{
    Washing_Machine,
};
```

Figure 2.3: RFID Guardian Reader File Example

```
tag P15693 Book
{
    tagid = 0xE0123456;
};

tag P15693 Bra
{
    tagid = 0xE0123457;
};

@tag P15693 MY_TAGS
{
    $Book,
};

@tag P15693 SECRET_TAGS
{
    $Bra,
};
```

Figure 2.4: RFID Guardian Tag File Example

```
reader READER_NAME
{
};
```

Figure 2.5: RFID Guardian Reader File Correct Form

In this example, there are two reader groups: *FRIENDLY_READERS* and *TRUSTED_READERS*, as well as two tag groups: *MY_TAGS* and *SECRET_TAGS*. Suppose the user requirements are:

- *FRIENDLY_READERS* can query *MY_TAGS*, but not *SECRET_TAGS*.
- *SECRET_TAGS* can be accessed by *TRUSTED_READERS*.

Then the above ACL file will meet these requirements.

In the UML Sequence Model, I showed that there will be a key exchange phase and an authentication phase before the reader can query the tag. These two phases are identical in almost all scenarios, so I will briefly describe them here and will not repeat this in the following sections.

In short, these two phases are done automatically during the SSL handshake phase. In order to exchange certificate files and key files, the reader and Guardian will send several packets to each other, in which they will write those files into the content of an RFID query command — *WRITE_MULTIPLE_BLOCK* or *READ_MULTIPLE_BLOCK*. Then they will use these files to establish an SSL connection. The reader can be viewed as a computer server executing *SSL_accept()*, while RFID Guardian can be regarded as a computer client executing *SSL_connect()*. After the connection is established, the messages between the reader and Guardian will use *SSL_read()* and *SSL_write()*. [20] The following list is an abstraction of messages between RFID Reader, RFID Guardian and RFID tags.

```
# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Now communication channel between Guardian and Reader
# is set up
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));
```

```
# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO));# Guardian → Reader

# SSL handshake using BIO over RFID
# Many read/write_multiple_request/respond messages
# Agree on cipher suite and generate session key etc.

# Option: Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# Now, on top of this SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (but not
# used in this example)

# Reader perform queries on normal RFID tags
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Since both Book tag and Bra tag reply, the reader
# starts anti-collision mechanism.

# Reader queries Book
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 1;
    mask = 0x0;
};

# Reader queries Bra
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 1;
    mask = 0x1;
};
```

```

# Guardian may send jamming signals according to its ACL
  Rule
GP_command(send_jamming_signal()); # Guardian → Reader

# Guardian disconnect
GP_commmmand(SSL_close());          # Guardian → Reader

```

Suppose the two readers in the above example now want to query the two RFID tags. The table below illustrates the messages and completes the ACL example. In this table, messages between the readers and Guardian are marked normally, while the messages between the readers and tags, which are the actual queries, are marked with italics.

| From | To | Goal | Result |
|------------------------|-----------------|------------------|-------------|
| Book_Shelf | Guardian | Inventory | ALLOW |
| Washing_Machine | Guardian | Inventory | ALLOW |
| Guardian | Book_Shelf | Start Connection | ALLOW |
| Guardian | Washing_Machine | Start Connection | ALLOW |
| Book_Shelf | Guardian | Hand shake | ALLOW |
| Guardian | Book_Shelf | Hand shake | ALLOW |
| Washing_Machine | Guardian | Hand Shake | ALLOW |
| Guardian | Washing_Machine | Hand Shake | ALLOW |
| Book_Shelf | Guardian | Authentication | ALLOW |
| Guardian | Book_Shelf | Authentication | ALLOW |
| Washing_Machine | Guardian | Authentication | ALLOW |
| Guardian | Washing_Machine | Authentication | ALLOW |
| <i>Book_Shelf</i> | <i>Book Tag</i> | <i>Query</i> | <i>DENY</i> |
| <i>Book_Shelf</i> | <i>Bra</i> | <i>Query</i> | <i>DENY</i> |
| <i>Washing_Machine</i> | <i>Book Tag</i> | <i>Query</i> | <i>DENY</i> |
| <i>Washing_Machine</i> | <i>Bra Tag</i> | <i>Query</i> | <i>DENY</i> |
| Guardian | Book_Shelf | Close Connection | ALLOW |
| Guardian | Washing_Machine | Close Connection | ALLOW |

Table 2.1: RFID Guardian Query Result Example

In the above example, all queries from readers to tags are denied because if the user didn't explicitly provide a context parameter, RFID Guardian will by default take the context as *ANY*. If we omit those authentication and key exchange issues, and focus only on the impact of different contexts we can compare the result of different context in Table: 2.2. So even if the user does not change the ACL rule but only switches the RFID Guardian switch "private" context to "trusted" context, then Guardian can deny the "Washing_Machine" queries by sending jamming signals! This also applies in the following sections, so if different results are generated the Guardian

ends up set to an illegal context. But for simplicity, let's assume that the Guardian already has the correct context that can make "LEGAL readers" successfully query RFID tags.

| Context | Reader | Tag | Result |
|---------|-----------------|------|--------|
| ANY | Book_Shelf | Book | DENY |
| | Washing_Machine | Book | DENY |
| | Book_Shelf | Bra | DENY |
| | Washing_Machine | Bra | DENY |
| trusted | Book_Shelf | Book | ALLOW |
| | Washing_Machine | Book | DENY |
| | Book_Shelf | Bra | DENY |
| | Washing_Machine | Bra | DENY |
| private | Book_Shelf | Book | DENY |
| | Washing_Machine | Book | DENY |
| | Book_Shelf | Bra | DENY |
| | Washing_Machine | Bra | ALLOW |
| home | Book_Shelf | Book | DENY |
| | Washing_Machine | Book | DENY |
| | Book_Shelf | Bra | DENY |
| | Washing_Machine | Bra | ALLOW |

Table 2.2: Different Context will decide different result

Note that this is an example ACL and users can add more rules to refine things as they like. We will show how to combine these rules with user cases in following sections. Actually the command from the readers arrives at the Guardian and RFID tags simultaneously, but it is difficult to describe this parallelism, so the description instead uses consecutive messages. Only ACCEPT scenarios are described in following user cases, meaning that all the readers in UML Sequence Model are legal readers and the Guardian will not send jamming signals. In real life, it is possible to group all the tags into a single but large group (maybe called "My.Tag") and simply denies all queries to this group. After that we can define specially which readers can query which tags. In the following sections, we will provide an ACL Rule, a reader file and a tag file. The tag/reader lists are likely to vary between application scenarios, whereas the ACL should vary as little as possible.

2.3 List of Usage Scenarios

This section describes RFID Guardian usage scenarios. These scenarios include: Contact-less Smart Card, Supermarket, Library, E-Passport, Car Key-less Entry System, Home appliance, Animal Identification, Supply Chain, Casino and Waste Management.

2.3.1 Contact-less Smart Card

Contact-less smart cards differ from traditional credit cards because they do not require the magnetic stripe on the back that comes into physical contact with a special reader. By the end of 2006, it is estimated that between 35 and 50 million credit and debit cards will be contact-less and available for use in 25,000 - 50,000 merchant locations in the United States alone. Many experts are anticipating that this new technology could eventually make the magnetic stripe obsolete, at which point all of the world's electronic payments would be contact-less.[4]

What gives contact-less credit cards their edge is that they are faster than traditional magnetic stripe cards and cash. According to Visa, the average cash transaction takes 34 seconds and magnetic stripe credit card transactions take 24 seconds. Conversely, contact-less credit card transactions take a mere 15 seconds.

For security reasons, the protocols of commercial contact-less credit cards are never published. In [4], the author described some security features of this kind of card. The contact-less payment cards feature 128-bit and triple DES encryption that would make any stolen data useless to a potential thief. The cards also have built-in sensors that would disable the chip should anyone attempt to retrieve any personal data from a stolen card. Furthermore, just like regular credit and debit cards, holders of contact-less cards are not liable for any fraudulent charges. In [5], however, researchers observed that RFID-enabled credit cards are only more secure than their traditional counterparts against certain types of attacks. For example, since the card never leaves the holder's hands, it will never leave an physical image of the card to a potentially adversarial merchant or clerk. However, by eavesdropping and carefully studying the communication content between reader and tags, it is still possible to deploy certain types of attacks to RFID-enabled credit cards, i.e. Replay with Race Condition. The detail of attacks is beyond the scope of this thesis, so interested readers please refer to [5].

Because we have no idea of the details of communication content between RFID tag and reader, we leave the command of legitimate readers in the ACL to be *any* instead of just an *INVENTORY* command, because the reader may require the tag to write some data in its memory, i.e. transaction number.

The relationships for credit card are shown in Figure 2.6 and 2.7. Because the authentication messages between RFID reader and tag in the UML Sequence Model (Figure: 2.7) are not specified, an *Authentication* message is used instead.

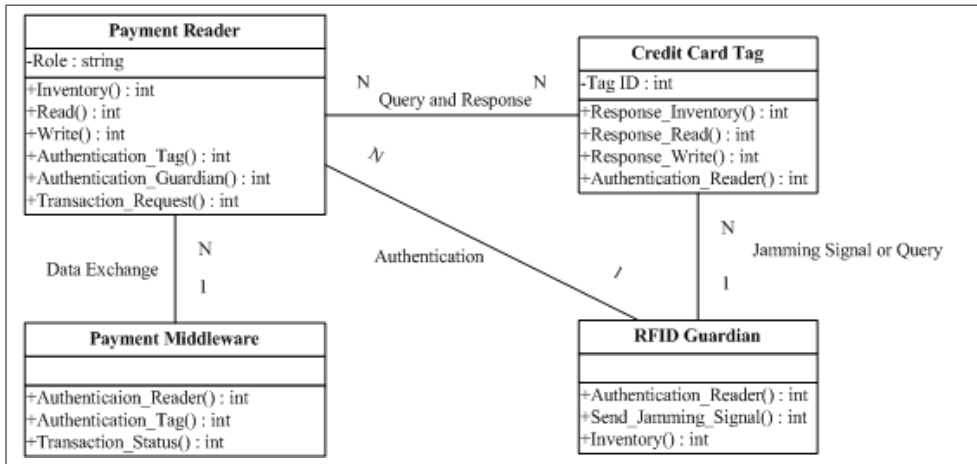


Figure 2.6: UML Object Model: Credit Card

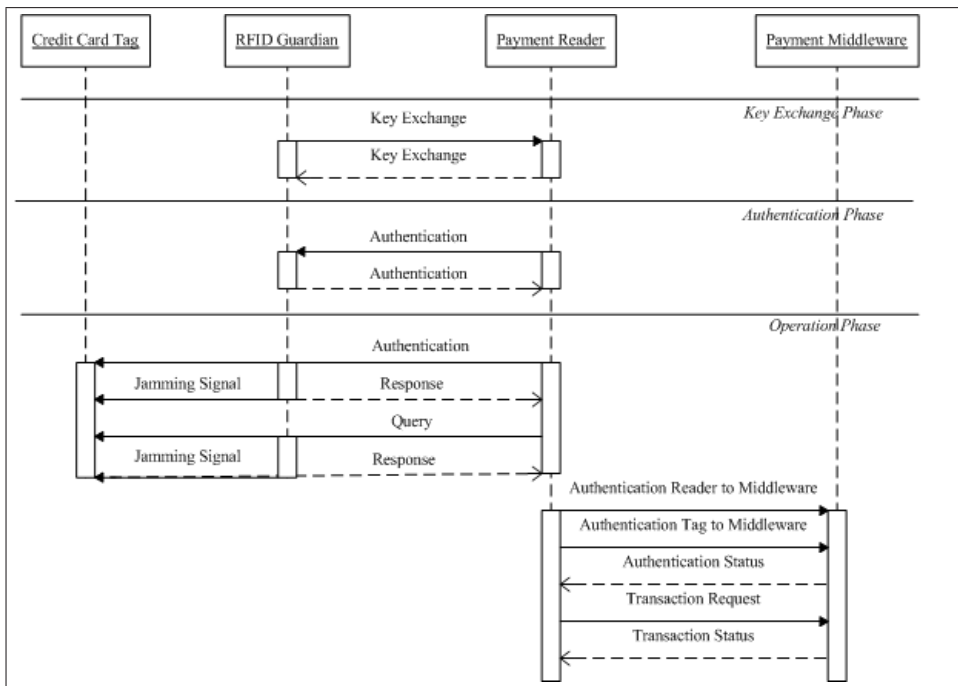


Figure 2.7: UML Sequence Model: Credit Card

Below is an example ACL file that could be used in this application scenario.

```
#####
```

```
# Credit Card Rules
#####

context trusted;
context home;

# By default, we leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Deny unknown readers to access our cards
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
    query = { command = *; };
};

# But allow payment readers
rule P15693 ACCEPT
{
    context = {trusted, home, };
    role = LEGAL_READER;
    tags = @MY_TAGS;
    query = { command = *; };
};
```

I also provide an example reader file (Figure: 2.8) and tag file (Figure: 2.9) below. In these files, I assume that the user is carrying an American Express Credit Card and there is a corresponding reader from the same company named `AMERICAN_EXPRESS_READER`.

```

reader AMERICAN_EXPRESS_READER;

role LEGAL_READER
{
    AMERICAN_EXPRESS_READER,
};

```

Figure 2.8: RFID Guardian Reader File: Credit Card

```

tag P15693 American_Express
{
    tagid = 0xE0123456;
};

@tag P16693 MY_TAGS
{
    $American_Express,
};

```

Figure 2.9: RFID Guardian Tag File: Credit Card

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader prove discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Now communication channel between Guardian and Reader
# is set up
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

```

```

# SSL handshake using BIO over RFID
# Many read/write_multiple_request/respond messages
# Agree on cipher suite and generate session key etc.

# Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# Now, on top of this ssl connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (but not
# used in this example)

# Connection setup, doing normal queries
# RFID Reader Query Credit Card Tag
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian disconnect
GP_commmand(SSL_close());          # Guardian → Reader

```

If the user executes the queries in the above figure, the following result will occur,

| From | To | Goal | Result |
|-------------|-----------------|----------------|--------|
| RFID READER | Guardian | Inventory | ALLOW |
| Guardian | RFID READER | Spoof UID | ALLOW |
| RFID READER | Guardian | HAND SHAKE | ALLOW |
| Guardian | RFID READER | HAND SHAKE | ALLOW |
| Guardian | RFID READER | Connect | ALLOW |
| RFID READER | Guardian | Authentication | ALLOW |
| Guardian | RFID READER | Authentication | ALLOW |
| RFID READER | Credit Card Tag | Query | ALLOW |
| Guardian | RFID READER | Disconnect | ALLOW |

Table 2.3: RFID Guardian Query Result: Credit Card

2.3.2 Supermarket

In retail shops consumers could check out by rolling shopping carts past point-of-sale terminals. These terminals would automatically tally the items, compute the total cost, and perhaps even charge the consumer's RFID-enabled payment devices. RFID tags would act as indices into database payment records, and could also help retailers track defective or contaminated items.[6]

In this scenario the RFID reader is used for not only scanning goods but for payment transaction as well. In real life multi-standards readers are very common and they could serve in our example. But the contactless smart card is just an option in this scenario and we will not include it in the UML diagram. We assume that multiple Guardians are within the proximity of the check-out desk and that is why mutual authentication is necessary here. The reader scans tags attached to the shop items and then transfers the tag ids back to the shop middleware. Then the shop middleware system will calculate the total price for these items and contact some payment systems, if necessary. At the end of a successful transaction, the middleware system will tell the reader to transfer the ownership of those items to RFID Guardian, so that the Guardian knows which tags it is going to protect. The corresponding UML figure is 2.10 and 2.11

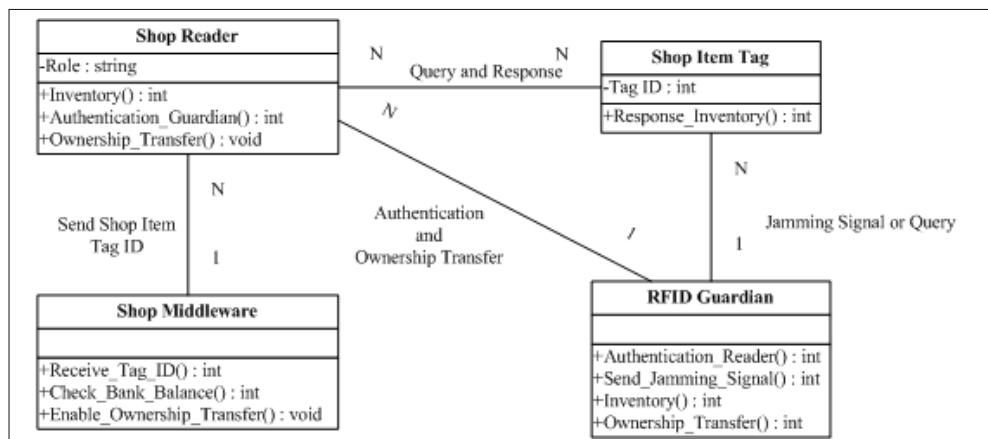


Figure 2.10: UML Object Model: Supermarket

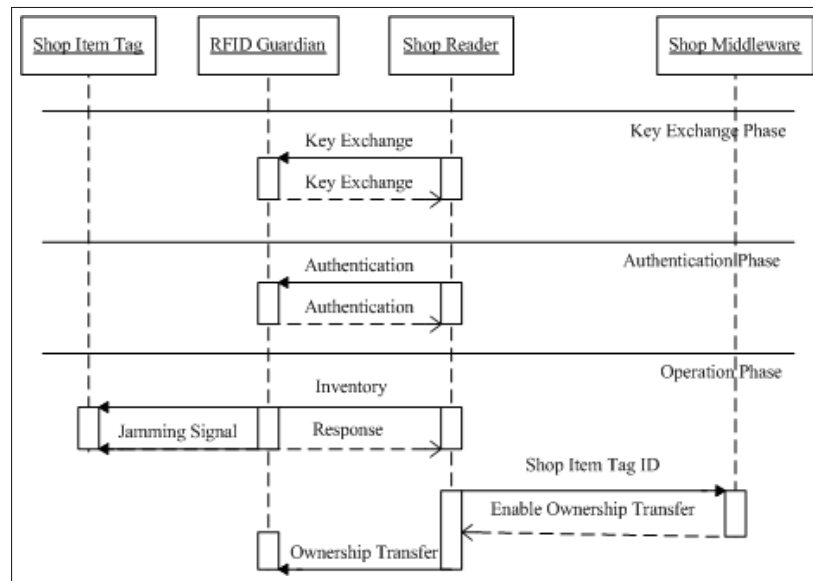


Figure 2.11: UML Sequence Model: Supermarket

The example ACL that could be used to in this scenario is shown below.

```
#####
# Supermarket Rules
#####

context trusted;
context home;

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Deny reading of credit card to readers
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
};
```

```
        query = { command = *; };
};

# Only allow payment reader to read card
rule P15693 ACCEPT
{
    context = {trusted, home, };
    role = LEGAL_READER;
    tags = @MY_CREDIT_CARD;
    query = { command = *; };
};
```

I also provide an example reader file (Figure: 2.12) and tag file (Figure: 2.13). In this example, I assume that the user is carrying an American Express Credit Card and the supermarket has a reader named DIRK_VAN_DER_BROOK (A supermarket in the Netherlands). Furthermore, I assume that the user is wearing a necklace, which belongs to the MY_TAG group. Since this necklace was not bought from the supermarket, the user would not like the supermarket reader to query it.

```
reader DIRK_VAN_DER_BROOK;

role LEGAL_READER
{
    DIRK_VAN_DER_BROOK,
};
```

Figure 2.12: RFID Guardian Reader File: Supermarket

```

tag P156693 American_Express
{
    tagid = 0xE0123456;
};

tag P15693 Necklace
{
    tagid = 0xE0123457;
};

tag P15693 Apple
{
    tagid = 0xE0123458;
};

@tag P15693 MY_CREDIT_CARD
{
    $American_Express,
};

@tag P15693 MY_TAGS
{
    $Necklace,
    @MY_CREDIT_CARD,
};

```

Figure 2.13: RFID Guardian Tag File: Supermarket

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

```

```
# Communication channel between Guardian and Reader
# is set up
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Many read/write_multiple_request/respond messages
# Agree on cipher suite and generate session key etc.

# Option: Authentication
GP_command(SSL_write(certificate)); # Reader → Guardian
GP_command(SSL_write(certificate)); # Guardian → Reader

# Now, on top of this SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not
# shown in this example)

GP_write(some_key.pem); # Reader → Guardian

# Query Credit Card
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 32;
    mask = 0xE0123456;
};

# Query Necklace
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 32;
    mask = 0xE0123457;
};

# Query Apple
query P15693
{
    flags = FL_INVENTORY;
```

```

        command = INVENTORY;
        masklen = 32;
        mask = 0xE0123458;
};

```

Note that in the last rule of the ACL, I specify that the super market reader can only query payment tag instead of the whole MY_TAGS group. So according to this last rule, when the necklace tag is going to respond, RFID Guardian will send out jamming signals, which means that RFID Guardian is now protecting the user's items! The interaction for a user buys apples from the reader above is shown in table 2.4. Note that the reader and Guardian can exchange messages once the SSL connection is established.

| From | To | Goal | Result |
|----------|------------------|------------------|--------|
| Reader | Guardian | Inventory | ALLOW |
| Guardian | Reader | Start Connection | ALLOW |
| Reader | Guardian | Hand shake | ALLOW |
| Guardian | Reader | Hand shake | ALLOW |
| Reader | Guardian | Authentication | ALLOW |
| Guardian | Reader | Authentication | ALLOW |
| Reader | American_Express | Query | ALLOW |
| Reader | Necklace | Query | DENY |
| Reader | Apple | Query | ALLOW |

Table 2.4: RFID Guardian Query Result: Supermarket

2.3.3 Library

RFID technology can also be deployed in library. In the US, there are about 60 libraries with approximately 10 million books using this technology. [11]. At least three procedures will be accelerated with the help of RFID: check-out, inventory and return.

- Checkout is easy once a person is identified as a library patron. When the user card is waved in front of a lending reader, the user is identified and the account is opened. The books' tags are now read by the reader and the chip are reset to status "checked out". When these books are carried through the sensor gate at the exit there will be no alarm. This procedure could be done either by a librarian or through a "self-check-out" station by the patron .
- Making an inventory can also be made automated (See Figure 2.14). The librarian only needs to connect the wand to a peripheral device (preferably a laptop computer), wave the wand across library collection

and let the reader wand and software do the rest — collect inventory data, search for missing items and identify mis-shelved materials. Data gathered during the inventory process is then uploaded into library's database.

- There are three ways to automate book returns. The first one is to have a librarian manually read the patron's library card and the books' RFID tag; the second is to deploy a self-check return station do the same thing. And a third possibility is an Auto Return Device (See Figure 2.15). As the items slide down the return chute, an RFID antenna reads the item's tag and checks the item back into the system. Note that the returning reader doesn't have to read the patron's tag, because the book may be returned by another person. The library does not care who returns the book, only the fact that each book is finally returned. This means that Auto Return Decive can be combined with a self-return station.



Figure 2.14: Library Inventory Figure 2.15: Auto Return Device

The “after return” procedure can also be accelerated (See Figure: 2.16). For example, some companies have developed an automated conveyor system to accept returned items, reactivate item security, separate materials that are on reserve, and assign the remaining items to their appropriate locations. As items enter the system, they are identified, checked in, and sorted based on library defined criteria. Returned items are immediately ready for reshelving, already sorted into bins by subject matter or other criteria. The system can also alert staff when a bin is full and should be emptied. The automatic features of the system allow a library to experience a dramatic reduction in average returns processing time. The returns system, configured with or without a sorter will automatically print a “holds” ticket for each item that is on reserve.



Figure 2.16: Automated Conveyor System

The following is a brief description of the inventory process. The RFID reader first broadcasts *inventory* requests and all tags within the reader's proximity respond with their tag IDs. If the reader receives more than one messages at the same time it will get confused, so an *anti-collision* process is used. The anti-collision process differs from the normal inventory process because the *mask length* and *mask value* fields are added to the *inventory* command. The inventory request then looks like this (Figure: 2.17).

```
query 15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 12;
    mask = 0xEEF;
};
```

Figure 2.17: Example Anti-collision Inventory Request

Mask length and mask value are used to distinguish between tags. Tag ids are usually 8-byte strings, so the anti-collision rule is that only tags with an ID whose final bits match the mask may reply. For example if tag A has ID 0xE0EEEEEF and tag B has ID 0xE0123456 and both hear the above query, then only tag A will respond because the end of its tag matches the mask. More details about the anti-collision process could be found in [16].

The UML is in Figure 2.18 and 2.19 and ACL is in Figure 2.20. Note that the “Key Exchange Phase” and “Authentication Phase” occur in both lending and return phases. For brevity only the lending phase is illustrated.

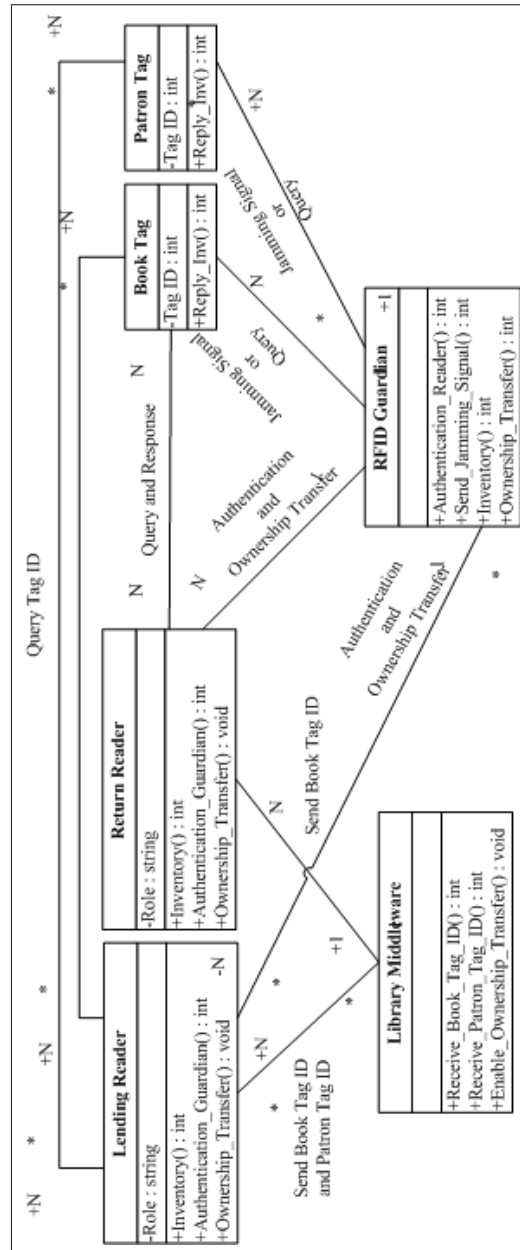


Figure 2.18: UML Object Model: Library

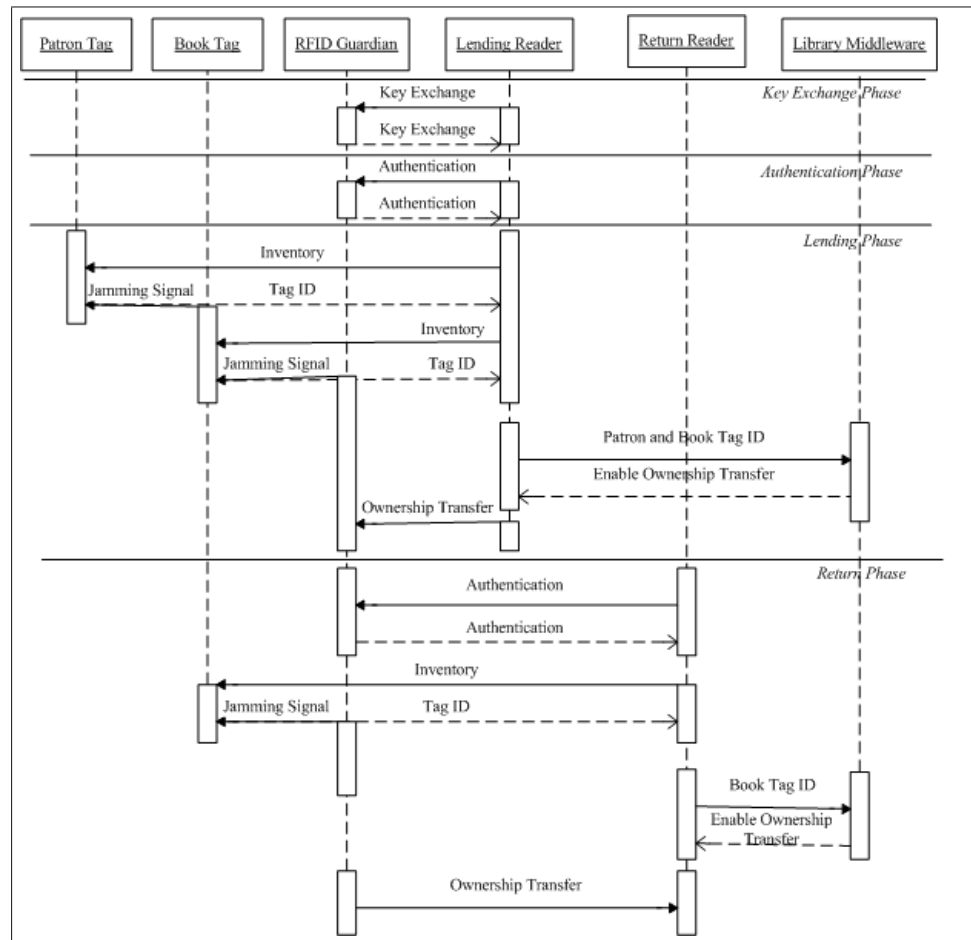


Figure 2.19: UML Sequence Model: Library

```
#####  
# Library rules  
#####  
  
context trusted;  
  
# By default, leave RFID traffic alone  
rule P15693 ACCEPT  
{  
    context = *;  
    role = *;  
    tags = *;  
    query = { command = *; };  
};  
  
# Block others from reading patron's books  
rule P15693 DENY  
{  
    context = *;  
    role = *;  
    tags = MY_TAGS;  
    query = { command = *; };  
}  
  
# Permit library readers to read library tags  
rule P15693 ACCEPT  
{  
    context = trusted;  
    role = LIBRARY;  
    tags = @LIBRARY_CARD;  
    query = { command = *; };  
};
```

Figure 2.20: RFID Guardian ACL: Library

```
reader LIBRARY_LEND_READER;

reader LIBRARY_RETURN_READER;

reader EXIT_GATE;

role LIBRARY
{
    LIBRARY_LEND_READER,
    LIBRARY_RETURN_READER,
    EXIT_GATE,
};
```

Figure 2.21: RFID Guardian Reader File: Library

```
tag P15693 BOOK_MINIX
{
    tagid = 0xE0123456;
};

tag P15693 PATRON
{
    tagid = 0xE0123457;
};

tag P15693 BOOK_LINUX
{
    tagid = 0xE0123458;
};

@tag P15693 MY_TAGS
{
    $BOOK_MINIX,
    $LIBRARY_CARD,
};

@tag P15693 LIBRARY_CARD
{
    $PATRON,
};
```

Figure 2.22: RFID Guardian Tag File: Library

The following is the interaction when a reader queries these tags.

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel established between Guardian and
  Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Many read/write_multiple_request/respond messages
# Agree on cipher suite and generate session key etc.

# Option: Authentication
GP_command(SSL_write(certificate)); # Reader → Guardian
GP_command(SSL_write(certificate)); # Guardian → Reader

# On top of SSL connection, Reader and Guardian
# exchange GP messages, e.g. exchanging keys (not shown)

# Actual actual anti-collision phase not shown.

# Query Minix Book
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 32;
    mask = 0xE0123456;
}

```

```

};

# Query Library Card
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 32;
    mask = 0xE0123457;
};

# Query Linux Book
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
    masklen = 32;
    mask = 0xE0123458;
};

```

Books that belong to the library will reply, while the user's own book will be protected by RFID Guardian (as shown in Table 2.5 and 2.6). This is because the library readers cannot query BOOK_LINUX after the lending phase because the ownership of that book is transferred to RFID Guardian.

| From | To | Goal | Result |
|---------------|------------------|------------------|--------|
| Group Library | Guardian | Inventory | ALLOW |
| Guardian | Group Library | Start Connection | ALLOW |
| Group Library | Guardian | Hand shake | ALLOW |
| Guardian | Group Library | Hand shake | ALLOW |
| Group Library | Guardian | Authentication | ALLOW |
| Guardian | Group Library | Authentication | ALLOW |
| Group Library | Minix Book Tag | Query | DENY |
| Group Library | Library Card Tag | Query | ALLOW |
| Group Library | Linux Book Tag | Query | DENY |
| Guardian | Group Library | Close Connection | ALLOW |

Table 2.5: RFID Guardian Query Result: Library (Lending Phase)

| From | To | Goal | Result |
|---------------|------------------|------------------|--------|
| Group Library | Guardian | Inventory | ALLOW |
| Guardian | Group Library | Start Connection | ALLOW |
| Group Library | Guardian | Hand shake | ALLOW |
| Guardian | Group Library | Hand shake | ALLOW |
| Group Library | Guardian | Authentication | ALLOW |
| Guardian | Group Library | Authentication | ALLOW |
| Group Library | Minix Book Tag | Query | DENY |
| Group Library | Library Card Tag | Query | ALLOW |
| Group Library | Linux Book Tag | Query | ALLOW |
| Guardian | Group Library | Close Connection | ALLOW |

Table 2.6: RFID Guardian Query Result: Library (Return Phase)

2.3.4 E-Passport

Many countries are planning to deploy RFID into their citizens' passports. This new kind of passport is often referred to as E-Passport or "Machine Readable Travel Document" (MRTD). The aim for adopting e-passport is to carry biometric information in RFID tags and to prevent passport forgery. The e-passports will contain digital signatures, so while cloning an e-passport is easy, modification its contents is largely infeasible. Since e-passports include biometric information such as facial images, copying does not permit impersonation, and therefore system security is not undermined.

The basic interaction between readers and tags is called Basic Access Control. In this scheme, the reader first acquires the Machine Readable Zone (MRZ) information from the data page of the passport, normally through a connected OCR scanner. MRZ is a standard for printing Optical Character Recognition (OCR) text. On a passport this information consists of the document holder's name, date of birth, gender, and the document's identification number and expiration date. The reader then computes session key from MRZ information. Finally the reader and the chip embedded in the passport generate and exchange random numbers to create a shared triple-DES session key.

The main problem with Basic Access Control is that the MRZ information could be read by any one who have access to it, so its keys are not secure enough. This is because besides customer officials there are many others who can touch passports such as hotel clerks. Even if customer officials at the airport are trustworthy, are hotel clerks?

Some researchers therefore propose an enhanced scheme called Extended Access Control. Extended Access Control consists of two phases, Chip Authentication followed by Terminal Authentication. Extended Access Control makes use of Diffie-Hellman key pair and requires the reader to authenticate

to the tag, thus ensures no sensitive information is leaked to illegal readers. Detailed information could be found in [9].

The corresponding UML is shown in figure 2.23 and 2.24 and ACL is in figure 2.25.

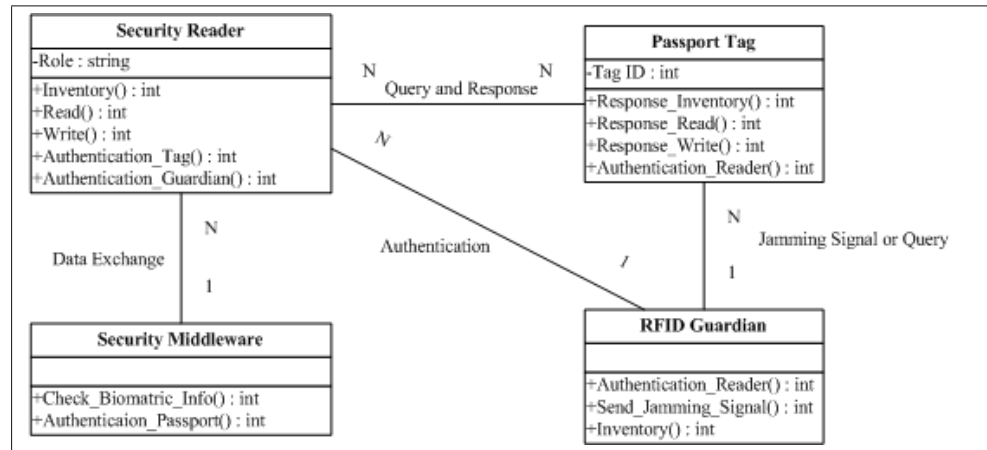


Figure 2.23: UML Object Model: E-Passport

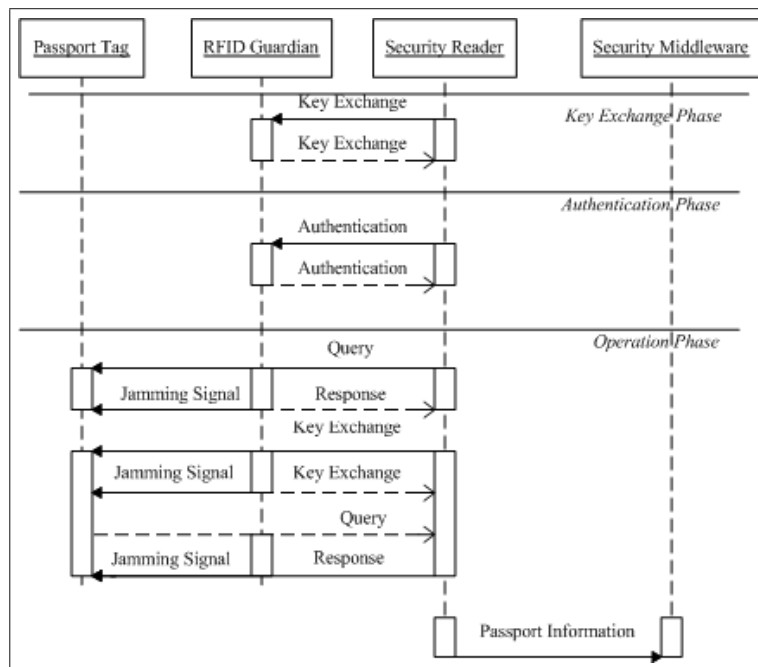


Figure 2.24: UML Sequence Model: Extended Access Control

```

context trusted;

#####
# Passport Rules
#####

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Deny others permissions to read this passport
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
    query = { command = *; };
};

# Permit reading by customer officials and police
rule P15693 ACCEPT
{
    context = trusted;
    role = LEGAL_READER;
    tags = @MY_TAGS;
    query = { command = *; };
};

```

Figure 2.25: RFID Guardian ACL: E-Passport

Also provided is an example reader file (Figure: 2.26) and a tag file (Figure: 2.27). The reader file uses the Schiphol airport and Uilenstede Police Station as an example. The reader file has extra contents because in this example Schiphol Airport is using an RSA-based authentication protocol with RFID Guardian and the public key is stored in a file named “schipol-public.pem”. And the public key of Uilenstede Police Station is stored in “uilenstede-public.pem”. Both readers are members of “LEGAL_READER”, because airport customer officials and police stations are

both legitimate querier of passports.

```

reader SCHIPHOL
{
    key =
    {
        type = rsa_public;
        store = 'schipol-public.pem';
    };
};

reader Uilenstede
{
    key =
    {
        type = rsa_public;
        store = 'uilenstede-public.pem';
    };
};

role LEGAL_READER
{
    SCHIPHOL,
    Uilenstede,
};

```

Figure 2.26: RFID Guardian Reader File: E-Passport

```

tag P15693 Passport
{
    tagid = 0xE0123456;
};

@tag P15693 MY_TAGS
{
    $Passport,
};

```

Figure 2.27: RFID Guardian Tag File: E-Passport

The following is the interaction if a reader queries these tags;

```
# Reader broadcasts inventory request
```

```

query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel established between Guardian and
  Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Multiple read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate)); # Reader → Guardian
GP_command(SSL_write(certificate)); # Guardian → Reader

# On top of SSL connection, Reader and Guardian
# exchange GP messages, e.g. exchanging keys (not shown)

# Reader queries Passport
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian disconnect
GP_command(SSL_close()); # Guardian → Reader

```

The RFID Guardian will judge if appropriate permissions exist, as shown in Table 2.7.

| From | To | Goal | Result |
|------------|--------------|------------------|--------|
| Reader | Guardian | Inventory | ALLOW |
| Guardian | SCHIPHOL | Start Connection | ALLOW |
| Guardian | UILENSTEDE | Start Connection | ALLOW |
| SCHIPHOL | Guardian | Hand shake | ALLOW |
| Guardian | SCHIPHOL | Hand shake | ALLOW |
| UILENSTEDE | Guardian | Hand Shake | ALLOW |
| Guardian | UILENSTEDE | Hand Shake | ALLOW |
| SCHIPHOL | Guardian | Authentication | ALLOW |
| Guardian | SCHIPHOL | Authentication | ALLOW |
| UILENSTEDE | Guardian | Authentication | ALLOW |
| Guardian | UILENSTEDE | Authentication | ALLOW |
| SCHIPHOL | Passport Tag | Query | ALLOW |
| UILENSTEDE | Passport Tag | Query | ALLOW |
| Guardian | SCHIPHOL | Close Connection | ALLOW |
| Guardian | UILENSTEDE | Close Connection | ALLOW |

Table 2.7: RFID Guardian Query Result: E-Passport

2.3.5 Car Key-less Entry Systems

During 1993, the worldwide increases in automotive theft reached a level which was no longer acceptable for insurance companies. The German insurance companies forced the rapid introduction of a new security system — an immobilizer, which uses RFID technology. Since the beginning of 1995, nearly all models for the European market are equipped with OEM immobilizers. Thefts of vehicles with electronic immobilizers are a tenth of thefts of vehicles without immobilizer.

The basic operation of immobilizer is simple: the door or the engine of the car will not be opened or started unless the user provides its authentication, i.e. a key. However, the details are very complex. In this section, I will briefly describe how Texas Instruments DST works.

The basic idea is to use a challenge/response scheme and a crypto device called Digital Signature Transponder (DST) serves that functionality. During initialization, the vehicle security system and the transponder exchange a secret encryption key. The key is not transmitted, only the transponder response to a challenge sent by the transceiver can be over heard.

The response R is a function of the encryption key K_e , the challenge $RAND$ and the cryptographic algorithm F_c . This response is returned to the transceiver using Frequency Shift Keying (FSK). The car security system calculates the expected response using the same algorithm and the same encryption key and compares the response received from the transponder to the calculated one. The calculation of the expected response either can be

done in parallel with the communication between transponder and reader or after reception of the transponder response. If expected and calculated responses match, success information will be sent to the engine management computer. In time critical applications, the challenge and the response can be generated after immobilization and stored for the next cycle.

The UML is in figure 2.28 and 2.29 and ACL is in figure: 2.30.

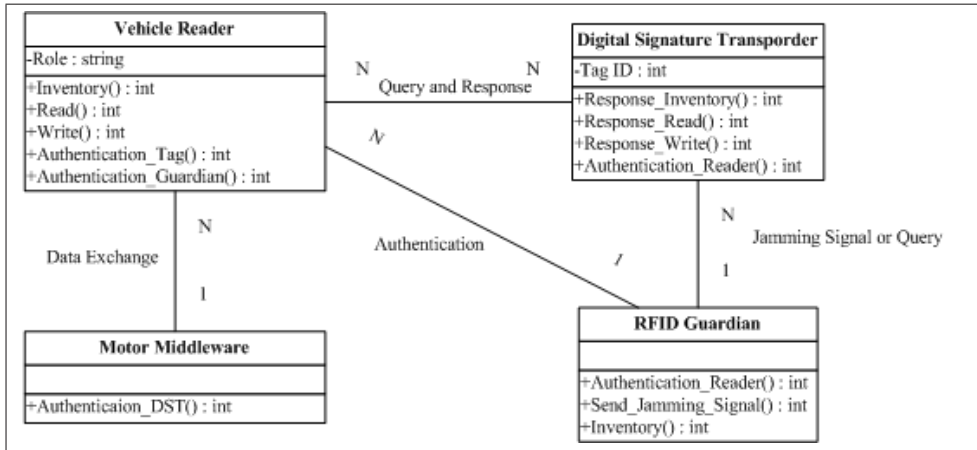


Figure 2.28: UML Object Model: Car Key-less Entry System

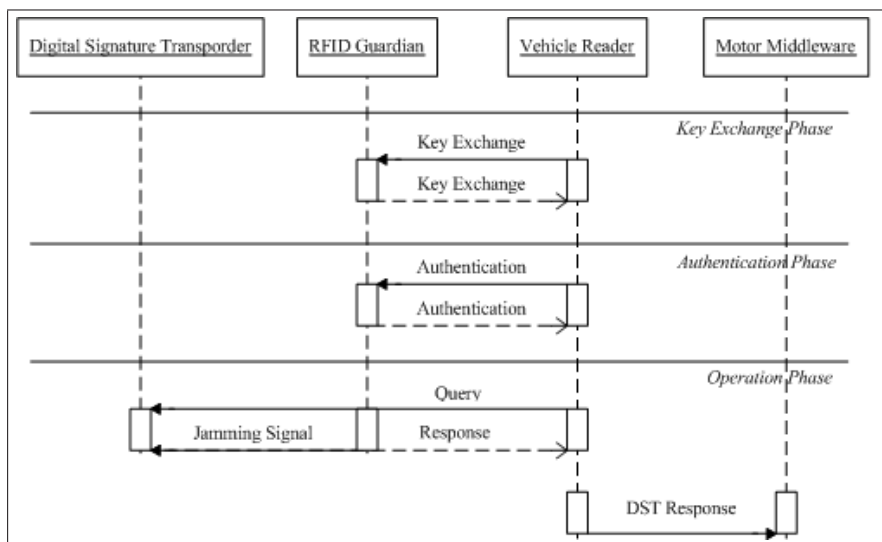


Figure 2.29: UML Sequence Model: Car Key-less Entry System

```
#####  
# Car Keyless Entry System Rules  
#####  
context trusted;  
  
# By default, leave RFID traffic alone  
rule P15693 ACCEPT  
{  
    context = *;  
    role = *;  
    tags = *;  
    query = {command = *};  
};  
  
# Block unknown readers from reading my keys  
rule P15693 DENY  
{  
    context = *;  
    role = *;  
    tags = @MY_TAGS;  
    query = { command = *; };  
};  
  
# Permit cars to read keys  
rule P15693 ACCEPT  
{  
    context = trusted;  
    role = LEGAL_READER;  
    tags = @MY_TAGS;  
    query = { command = *; };  
};
```

Figure 2.30: RFID Guardian ACL Rule: Car Key-less Entry System

```

reader BENZ;

role LEGAL_READER
{
    BENZ,
};

```

Figure 2.31: RFID Guardian Reader File: Car Key-less Entry System

```

tag P15693 BENZ_KEY
{
    tagid = 0xE0123456;
};

@tag P15693 MY_TAGS
{
    $BENZ_KEY,
};

```

Figure 2.32: RFID Guardian Tag File: Car Key-less Entry System

The following is the interaction when a reader queries this tag;

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel established between Guardian and
Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

```

```

# SSL handshake using BIO over RFID
# Multiple read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# On top of SSL connection Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not shown)

# Query DST
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

```

Then the result is shown below;

| From | To | Goal | Result |
|------------|------------|------------------|--------|
| Car Reader | Guardian | Inventory | ALLOW |
| Guardian | Car Reader | Start Connection | ALLOW |
| Car Reader | Guardian | Hand shake | ALLOW |
| Guardian | Car Reader | Hand shake | ALLOW |
| Car Reader | Guardian | Authentication | ALLOW |
| Guardian | Car Reader | Authentication | ALLOW |
| Car Reader | Key Tag | Query | ALLOW |
| Guardian | Car Reader | Close Connection | ALLOW |

Table 2.8: RFID Guardian Query Result: Car Key-less Entry System

2.3.6 Home Appliance

RFID technique could be used in the home. For example, RFID readers could be placed in washing machines and RFID tags in clothes such as socks. In this way washing machines could detect when clothes are put in them, and choose in the appropriate temperature and rotation speed. Other examples are frozen dinners which tell the microwave their cooking instructions, filing cabinets which know the location of tagged documents, medicine bottles which know drug interactions and location sensitive stamps for faster mail delivery.

In the intelligent washing machine example, the RFID tag is encased in a soft plastic shell so it can be attached to any textile and be washed, ironed and pressed at least 100 times and survive (according to Midori Taniyama, of Fujitsu's RFID systems department [12]). These tags operate in the UHF band (at 952MHz to 955MHz) and can be read at distances of up to 1.2m, which means that the tags on all the items in a basket of laundry can be read in a few seconds. At each stage in the laundering process it's possible to place sensors to quickly determine the location of any item. Extending the sensor network beyond washing to the pick-up and delivery system creates a complete linen or garment management system.

In summary, the sequence of home appliance is:

1. Intelligent machine readers broadcast inventory queries.
2. Each tag responds with its unique IDs.
3. Legal readers collect IDs and send those IDs to RFID Middleware (or rather a back-end system), which performs further actions. For example, a washing machine may query an Internet-based database for controlling instructions.

The UML is in Figure: 2.33 and 2.34 and ACL is in Figure: 2.35.

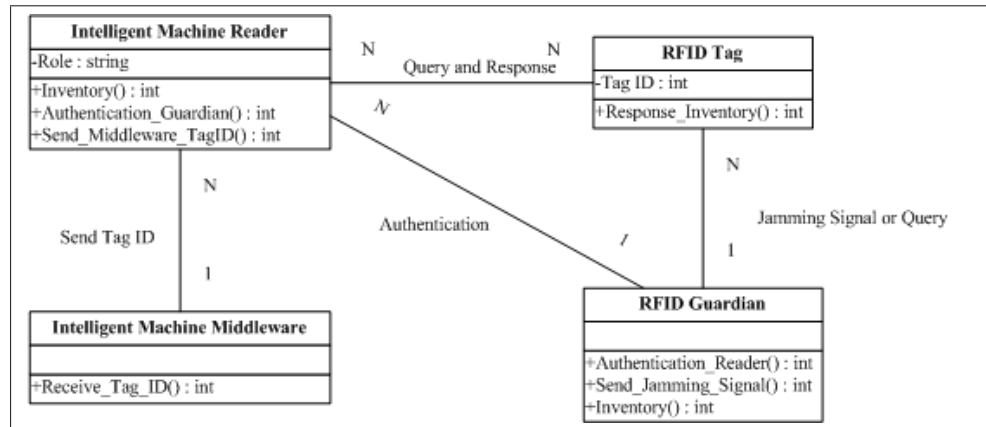


Figure 2.33: UML Object Model: Home Appliance

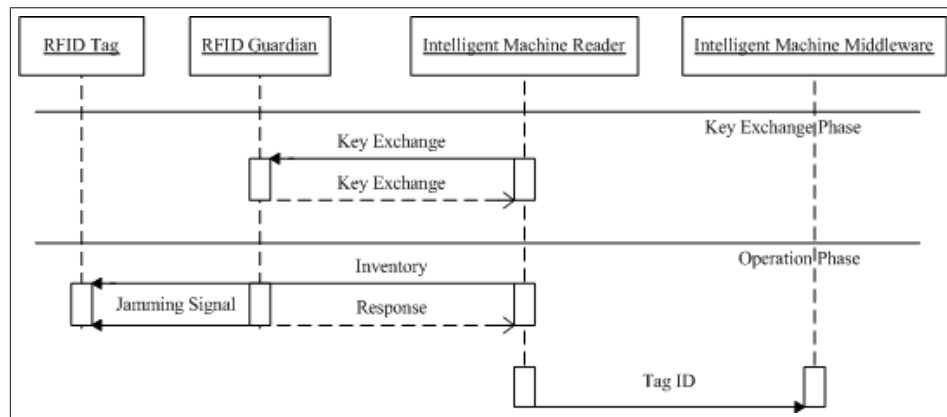


Figure 2.34: UML Sequence Model: Home Appliance

```
#####  
# HOME rules  
#####  
context home;  
context trusted;  
  
# By default, leave RFID traffic alone  
rule P15693 ACCEPT  
{  
    context = *;  
    role = *;  
    tags = *;  
    query = { command = *; };  
};  
  
# Block other readers from reading my tags  
rule P15693 DENY  
{  
    context = *;  
    role = *;  
    tags = @MY_TAGS;  
};  
  
# Permit home readers to read my tags  
# when in a trusted or home environment  
rule P15693 ACCEPT  
{  
    context = {home, trusted, };  
    role = LEGAL_READERS;  
    tags = @MY_TAGS;  
    query = { command = INVENTORY; };  
};
```

Figure 2.35: RFID Guardian ACL Rule: Home Appliance

```

reader WASHING_MACHINE;

reader FRIDGE;

role LEGAL_READER
{
    WASHING_MACHINE,
    FRIDGE,
};

```

Figure 2.36: RFID Guardian Reader File: Home Appliance

```

tag P15693 TSHIRT
{
    tagid = 0xE0123456;
};

tag P15693 ICE_CREAM
{
    tagid = 0xE0123457;
};

@tag 15693 MY_TAGS
{
    $TSHIRT,
    $ICE_CREAM,
};

```

Figure 2.37: RFID Guardian Tag File: Home Appliance

The interaction between reader and tags in the home would be as follows;

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian

```

```
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel established between Guardian and
  Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Many read/write_multiple_request/respond messages
# Agree on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate)); # Reader → Guardian
GP_command(SSL_write(certificate)); # Guardian → Reader

# On top of SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not shown)

# Query TSHIRT/ICECREAM Tags
# Omit anti-collision phase
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};
```

These tags will respond with their tag IDs, as shown in Table 2.9.

| From | To | Goal | Result |
|-----------------|-----------------|------------------|--------|
| FRIDGE | Guardian | Inventory | ALLOW |
| Washing_Machine | Guardian | Inventory | ALLOW |
| Guardian | FRIDGE | Start Connection | ALLOW |
| Guardian | Washing_Machine | Start Connection | ALLOW |
| FRIDGE | Guardian | Hand shake | ALLOW |
| Guardian | FRIDGE | Hand shake | ALLOW |
| Washing_Machine | Guardian | Hand Shake | ALLOW |
| Guardian | Washing_Machine | Hand Shake | ALLOW |
| FRIDGE | Guardian | Authentication | ALLOW |
| Guardian | FRIDGE | Authentication | ALLOW |
| Washing_Machine | Guardian | Authentication | ALLOW |
| Guardian | Washing_Machine | Authentication | ALLOW |
| FRIDGE | ICE.CREAM | Query | ALLOW |
| Washing_Machine | TSHIRT | Query | ALLOW |
| Guardian | FRIDGE | Close Connection | ALLOW |
| Guardian | Washing_Machine | Close Connection | ALLOW |

Table 2.9: RFID Guardian Query Result: Home

2.3.7 Animal Identification

It is estimated that some fifty million house pets around the world have RFID tags implanted in their bodies, to facilitate return to their owners should they become lost. The standard for animal identification is ISO 11784 and 11785. The first standard specifies the RFID code for use with animals and the second standard deals with activation of a transponder and the transfer of stored transponder information to a transceiver.

The animal identification system works as follows:

Pet owners bring their pets to a veterinarian, who plants a microchip under the skin of the pet. The microchip contains a unique identification number within biocompatible material. The microchip is the size of a grain of rice (approximately 12mm), and cannot be seen in the pet once it has been implanted. If the animal gets lost and is brought to a shelter, the clerk can use a reader to retrieve the tag ID from the RFID tag and use it to search for relevant information in a large database. If this search succeeds, the clerk will contact the owners of this animal. If the pet is not registered, the database will provide the name of the veterinarian who injected the microchip and the veterinarian will provide the owner's latest information.

The details for retrieving information from the tag between applications. For example AVID (one of the largest vendors for pet ID in USA) encrypts the data in its pet ID tags. If a lost pet implanted with such an encrypted AVID tag is brought to a shelter or vet, most readers will read the tag's

encrypted ID number but won't be able to decrypt it unless the reader has a special algorithm. This forces the shelter or vet to contact AVID to get it decrypted before they can look it up in an AVID-operated database to locate the pet's owner.[10]

The UML is in Figure 2.38 and 2.39 and ACL is in Figure 2.40.

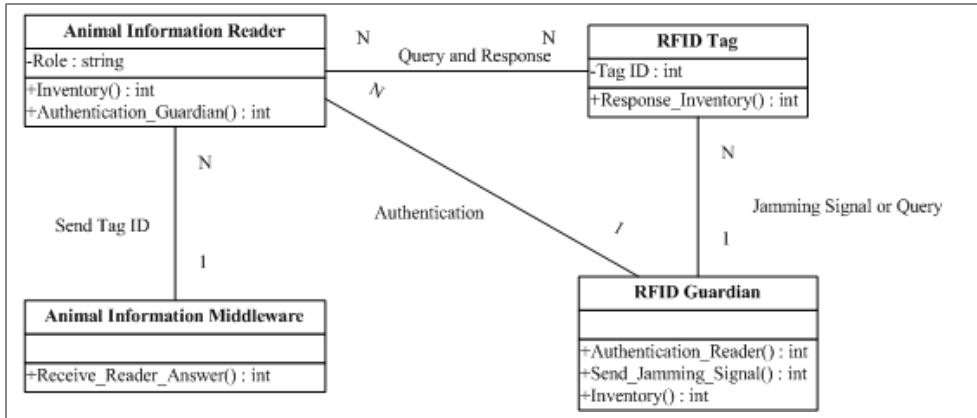


Figure 2.38: UML Object Model: Animal Identification

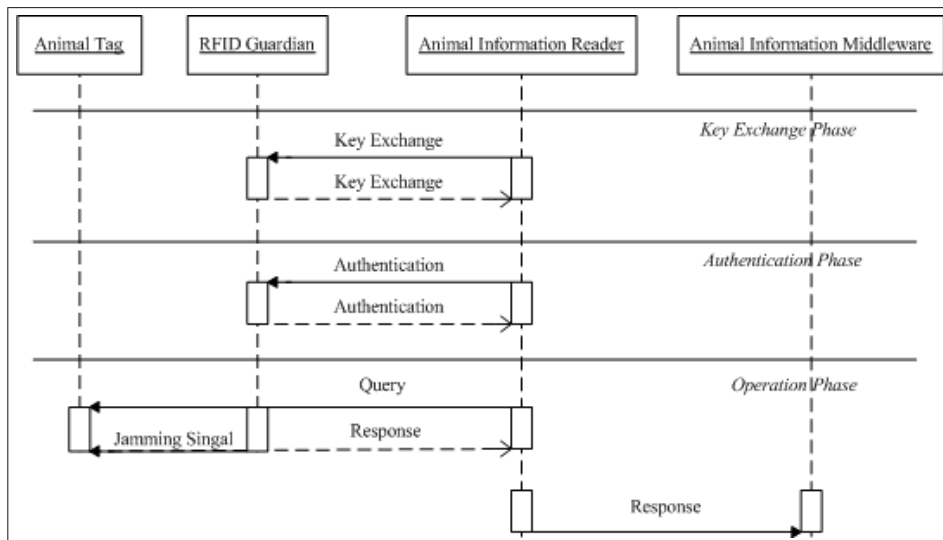


Figure 2.39: UML Sequence Model: Animal Identification

```
#####
# Animal rules
#####
context trusted;

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Do not permit others to read pet ID
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
    query = { command = *; };
};

# Allow pet service to read pet ID
rule P15693 ACCEPT
{
    context = trusted;
    role = LEGAL_READER;
    tags = @MY_TAGS;
    query = { command = *; };
};
```

Figure 2.40: RFID Guardian ACL Rule: Animal Identification

```
reader PET_SERVICE;

role LEGAL_READER
{
    PET_SERVICE,
};
```

Figure 2.41: RFID Guardian Reader File: Animal Identification

```
tag P15693 CAT
{
    tagid = 0xE0123456;
};

tag P15693 DOG
{
    tagid = 0xE0123457;
};

@tag P15693 MY_TAGS
{
    $CAT,
    $DOG,
};
```

Figure 2.42: RFID Guardian Tag File: Animal Identification

Suppose the cat and the dog are lost and found by the shelter, then the messages exchanged between reader and tags are as follows:

```
# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel set up between Guardian and Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Several read/ write_multiple_request /respond messages
```

```

# Agreement on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# On top of SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not shown)

# Query Cat/Dog Tags
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

```

Then the result will be the same as Table 2.10:

| From | To | Goal | Result |
|----------|----------|------------------|--------|
| Reader | Guardian | Inventory | ALLOW |
| Guardian | Reader | Start Connection | ALLOW |
| Reader | Guardian | Hand shake | ALLOW |
| Guardian | Reader | Hand shake | ALLOW |
| Reader | Guardian | Authentication | ALLOW |
| Guardian | Reader | Authentication | ALLOW |
| Reader | CAT Tag | Query | ALLOW |
| Reader | DOG Tag | Query | ALLOW |
| Guardian | Reader | Close Connection | ALLOW |

Table 2.10: RFID Guardian Query Result: Animal Identification

2.3.8 Supply Chain

The usage of RFID tags in military service and commerce is mostly related to supply chains. Although RFID technology improves efficiency in military and business applications, it also has the potential danger of leaking information to unknown or even hostile readers. One proposal is to install RFID Guardians in transport vehicles, such as trucks, to prevent espionage in retail outlets or military supply chains.

Military

Armies already deploy active RFID tags for container shipment, for example the Department of Defence (DoD) in the United States spends more than 100 million dollars on research into RFID technology and used a form of RFID to track container shipments during the Gulf War (Figure: 2.43 and 2.44). It is reported that the department will require its suppliers to apply RFID tags to cases, pallets and all packaging of commodities shipped to all DoD locations by 2007. Military officials report that although that RFID technology also saves dollars for taxpayers, its most important benefit is to increase military readiness. Ensuring a multimillion-dollar aircraft isn't sitting idle on a aircraft carrier waiting for a part can produce enormous savings in terms of readiness. [17, 18, 19].



Figure 2.43: Military Tag

Figure 2.44: Military Container

The UML is in Figure 2.45 and 2.46 and ACL is in Figure 2.47.

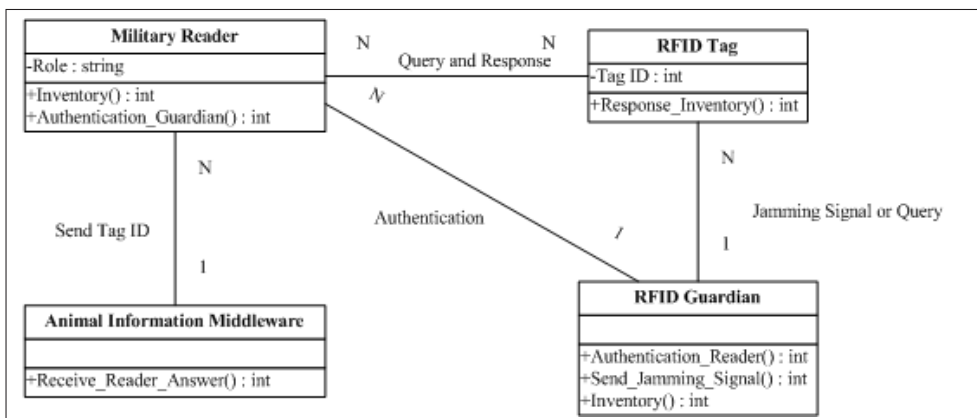


Figure 2.45: UML Object Model: Military Container Shipment

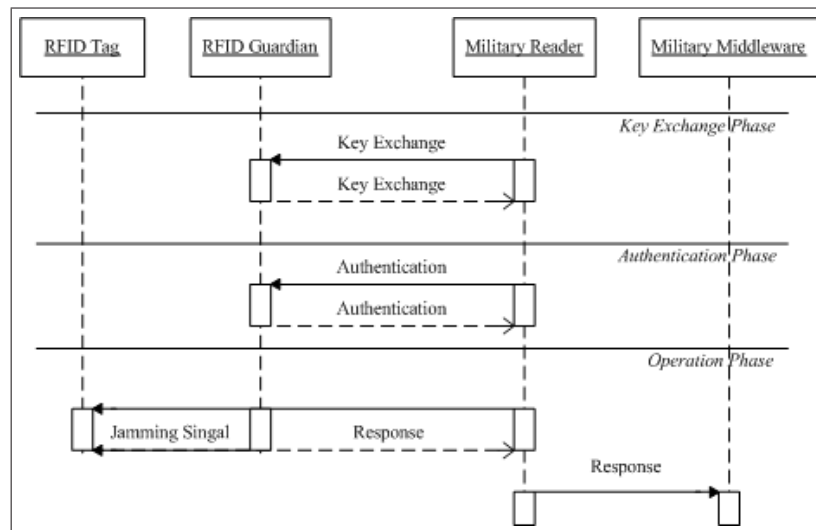


Figure 2.46: UML Sequence Model: Military Container Shipment

```
context supplier;
context home;

#####
# Military rules
#####

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Do not permit others to read military items
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MILITARY_TAGS;
    query = { command = *; };
};

# Permit military shipment reader to read items
rule P15693 ACCEPT
{
    context = {supplier, home, }
    role = LEGAL_READER;
    tags = @MILITARY_TAGS;
    query = { command = *; };
};
```

Figure 2.47: RFID Guardian ACL Rule: Military Container Shipment

```

reader MILITARY_SHIPMENT;

role LEGAL_READER
{
    MILITARY_SHIPMENT,
};

```

Figure 2.48: RFID Guardian Reader File: Military Container Shipment

```

tag P15693 BOMB
{
    tagid = 0xE0123456;
};

@tag P15693 MILITARY_TAGS
{
    $BOMB,
};

```

Figure 2.49: RFID Guardian Tag File: Military Container Shipment

When the trucks arrive at the destination, the interaction between the RFID readers and these tags is as follow:

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel set up between Guardian and Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.

```

```

GP_command(SSL_connect(RFID BIO));# Guardian → Reader

# SSL handshake using BIO over RFID
# Several read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# On top of SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not shown)

# Query Military Tags
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian disconnect
GP_commmmand(SSL_close());          # Guardian → Reader

```

The result of executing these queries will be as follows.

| From | To | Goal | Result |
|----------|----------|------------------|--------|
| Reader | Guardian | Inventory | ALLOW |
| Guardian | Reader | Start Connection | ALLOW |
| Reader | Guardian | Hand shake | ALLOW |
| Guardian | Reader | Hand shake | ALLOW |
| Reader | Guardian | Authentication | ALLOW |
| Guardian | Reader | Authentication | ALLOW |
| Reader | Bomb Tag | Query | ALLOW |
| Guardian | Reader | Close Connection | ALLOW |

Table 2.11: RFID Guardian Query Result: Military Container Shipment

Commerce

Wal-Mart, one of the biggest supermarkets in the world, already asked its top 100 suppliers to implement RFID in its supply chain. Goods shipped to its stores with RFID tags are recorded once at their arrival. Employees can simply wave scanner at the boxes to know what is inside without having

to open anything. Even before the arrival it is possible to know where everything is, which helps to reduce loss during shipment. The tags are read again before they are brought to the sales floor, but no reader is installed at the sales point. They are read for the last time at a box crusher after all the items in the case have been put on the store shelves.

The Object Model and Sequence Model of this user case is shown in 2.50 and 2.51.

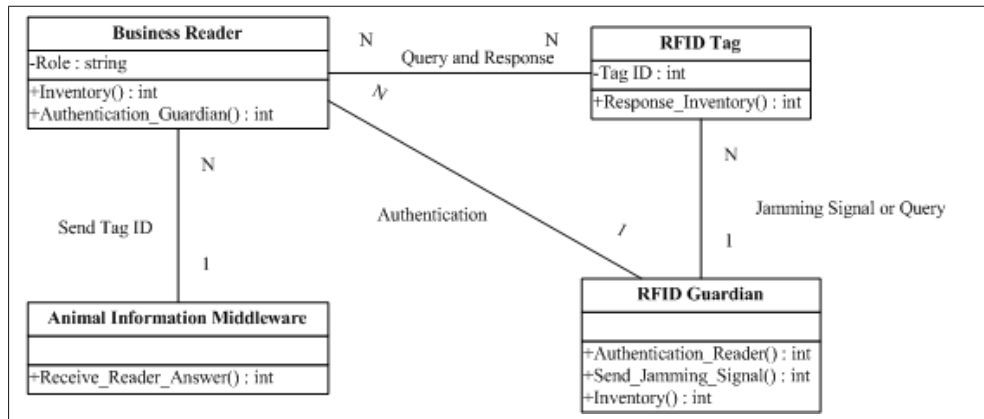


Figure 2.50: UML Object Model: Business Usage

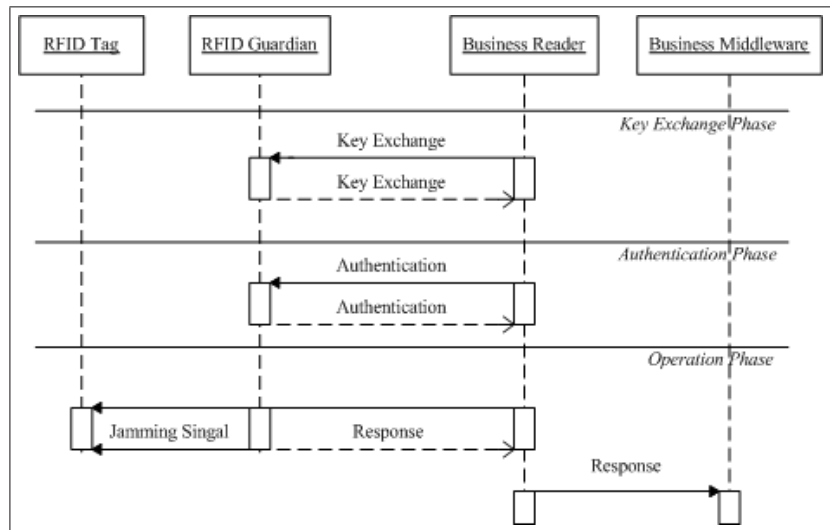


Figure 2.51: UML Object Model: Business Usage

The Access Control List for this user case is shown below:

```
#####
```

```
# Business rules
#####

context trusted;

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Do not permit others to read ID
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
    query = { command = *; };
};

# Allow legal readers to read
rule P15693 ACCEPT
{
    context = trusted;
    role = LEGAL_READER;
    tags = @MY_TAGS;
    query = { command = *; };
};
```

```

reader Supplier;

reader Distributor;

role LEGAL_READER
{
    Supplier,
    Distributor,
};

```

Figure 2.52: RFID Guardian Reader File: Business Usage

```

tag P15693 Furniture
{
    tagid = 0xE0123456;
};

@tag P15693 MY_TAGS
{
    $Furniture,
};

```

Figure 2.53: RFID Guardian Tag File: Business Usage

The interaction between readers and tags is as follows;

```

# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel set up between Guardian and Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

```

```
# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO));# Guardian → Reader

# SSL handshake using BIO over RFID
# Several read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Optional: Authentication
GP_command(SSL_write(certificate));# Reader → Guardian
GP_command(SSL_write(certificate));# Guardian → Reader

# On top of SSL connection, Reader and Guardian
# can exchange GP messages, e.g. exchanging keys (not shown)

# Query Furniture
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian disconnect
GP_commmand(SSL_close());          # Guardian → Reader
```

| Reader | Tag Name | Tag ID | Result |
|-------------|-------------|------------------|--------|
| Supplier | Guardian | Inventory | ALLOW |
| Distributor | Guardian | Inventory | ALLOW |
| Guardian | Supplier | Start Connection | ALLOW |
| Guardian | Distributor | Start Connection | ALLOW |
| Supplier | Guardian | Hand shake | ALLOW |
| Guardian | Supplier | Hand shake | ALLOW |
| Distributor | Guardian | Hand shake | ALLOW |
| Guardian | Distributor | Hand shake | ALLOW |
| Supplier | Guardian | Authentication | ALLOW |
| Guardian | Supplier | Authentication | ALLOW |
| Distributor | Guardian | Authentication | ALLOW |
| Guardian | Distributor | Authentication | ALLOW |
| Supplier | Furniture | Query | ALLOW |
| Distributor | Furniture | Query | ALLOW |
| Guardian | Supplier | Close Connection | ALLOW |
| Guardian | Distributor | Close Connection | ALLOW |

Table 2.12: RFID Guardian Query Result: Business Usage

2.3.9 Casino

It is not surprising to find RFID tags in casinos, because the entertainment industry drives technological innovations. By putting RFID tags into their gambling chips, the operators can spot counterfeiting and theft, and also monitor the behaviour of gamblers. Originally the gaming industries implemented RFID technology with 125 KHZ in 1995, but soon they found that the speed for counting chips was too low, so they changed to a higher frequency — 13.56 MHz, and this proved to be a success. Their devices can now read 1,000 chips per second and the memory capacity is over 10,000 bits, which is five times the capacity of the nearest low frequency. [13]

The UML object model is shown in Figure 2.54 and the sequence model is shown in Figure 2.55.

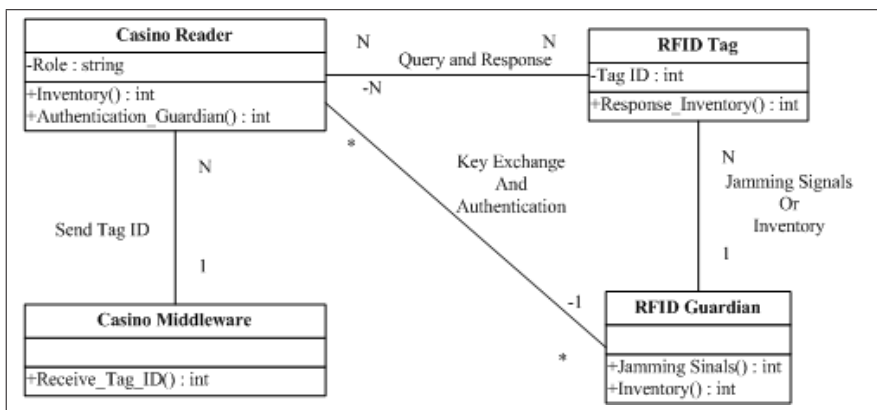


Figure 2.54: UML Object Model: Casino

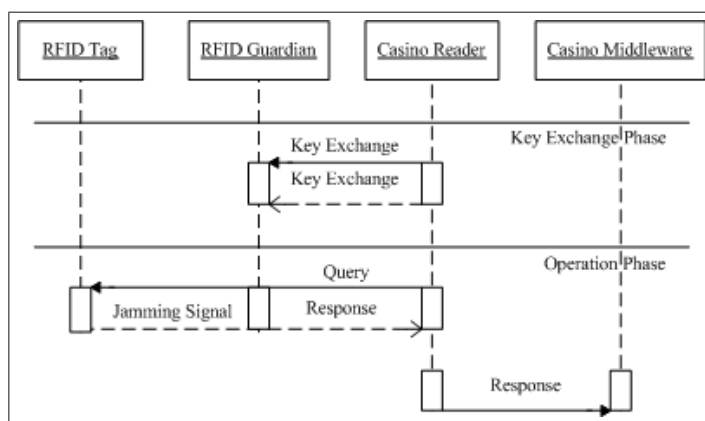


Figure 2.55: UML Sequence Model: Casino

```
#####  
# Casino  
#####  
  
# By default, leave RFID traffic alone  
rule P15693 ACCEPT  
{  
    context = *;  
    role = *;  
    tags = *;  
    query = { command = *; };  
};  
  
# Block all queries to tags  
rule P15693 DENY  
{  
    context = *;  
    role = *;  
    tags = @MY_TAGS;  
    query = { command = *; };  
};
```

Figure 2.56: RFID Guardian ACL Rule: Casino

Note that this ACL is shorter than previous ones and contains no reader names or tag names. This is because the user (a gambler) is carrying the RFID Guardian and the gambler does not own the poker chips, or the readers; these are both owned by casinos. Therefore, RFID Guardian only has to protect only the user's own tags. It is unclear whether the casino reader will authenticate itself to the Guardian but if it does this authentication procedure fails and Guardian will treat the Casino Reader as an unknown reader. So the ACL rule should be 2.56.

The message interaction between casino tags and readers are totally determined by casino, thus RFID Guardian should always allow this query, as shown below:

```
# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel set up between Guardian and Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO)); # Guardian → Reader

# SSL handshake using BIO over RFID
# Several read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Authentication
GP_command(SSL_write(certificate)); # Reader → Guardian
GP_command(SSL_write(certificate)); # Guardian → Reader

# Authentication fails , so there is no SSL

# Query Chip Tag
```

```

query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

```

So the querying result is shown in Table 2.13.

| From | To | Goal | Result |
|---------------|---------------|------------------|--------|
| Casino Reader | Guardian | Inventory | ALLOW |
| Guardian | Casino Reader | Start Connection | ALLOW |
| Casino Reader | Guardian | Hand shake | ALLOW |
| Guardian | Casino Reader | Hand shake | ALLOW |
| Casino Reader | Guardian | Authentication | ALLOW |
| Guardian | Casino Reader | Authentication | ALLOW |
| Casino Reader | Chip Tag | Query | ALLOW |
| Casino Reader | MY_TAGS | Query | DENY |
| Guardian | Casino Reader | Close Connection | ALLOW |

Table 2.13: RFID Guardian Query Result: Casino

2.3.10 Waste Management

RFID technology has been proposed for two separate processes within waste management — Automated Data Collection and Waste Separation.

In “Automated Data Collection” (See Figure 2.57 and 2.58), waste disposal trucks equipped with RFID readers pick up bins marked with RFID tags. The readers then record the exact time and place each waste bin is emptied. This permits a new degree of monitoring and control of the waste-disposal process. This information is then uploaded to the warehouse and later the RFID middleware — a computer server, which determines the bill for the customer.

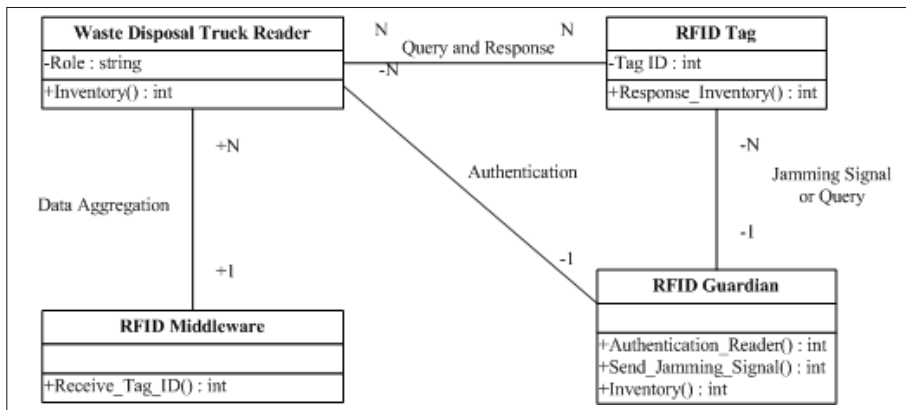


Figure 2.57: UML Object Model: Waste Management

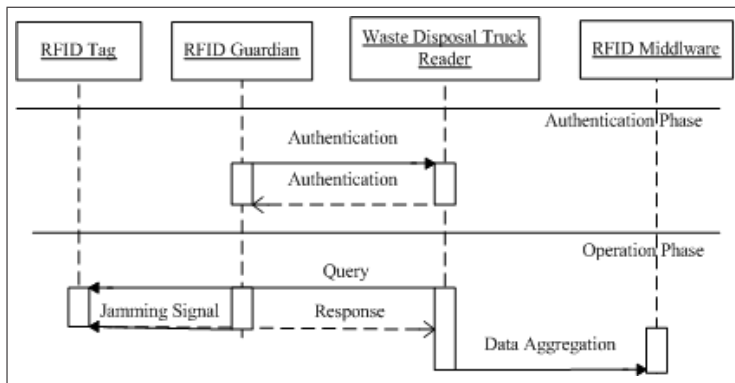


Figure 2.58: UML Object Model: Waste Management

Like in the previous section, it is the customer who is assumed to be carrying RFID Guardian, not the truck driver. This is because the dustbins are placed in public areas and it is not wise to put a Guardian close to a dustbin to protect it because it will quickly be stolen or damaged. Furthermore, dustbins are only used for public usage and it is not necessary to protect their “privacy”. The difference between this scenario and the Casino scenario is that these tags are placed in a public area for public usage, whereas the casino tags are inserted into chips and only used for by a company. However, if the user is standing close to a truck reader, the RFID Guardian will send out signals to protect the user’s own tags instead of protecting the dustbin tags!

Therefore, the ACL rule applied in this scenario is 2.59.

```
#####
# Waste management rules
#####

# By default, leave RFID traffic alone
rule P15693 ACCEPT
{
    context = *;
    role = *;
    tags = *;
    query = { command = *; };
};

# Block all queries to own tags
rule P15693 DENY
{
    context = *;
    role = *;
    tags = @MY_TAGS;
    query = { command = *; };
};
```

Figure 2.59: RFID Guardian ACL Rule: Waste Management

The interaction between truck readers and dustbin tags is as follows;

```
# Reader broadcasts inventory request
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

# Guardian replies with spoof UID
GP_command(Guardian's spoof UID); # Guardian → Reader

# Reader proves discovery of Guardian
GP_command(read_multi_request); # Reader → Guardian
GP_command(read_multi_respond); # Guardian → Reader

# Communication channel set up between Guardian and Reader
# Reader waits for connection.
GP_command(SSL_accept(RFID BIO));
```

```

# Guardian connects to Reader.
GP_command(SSL_connect(RFID BIO));# Guardian → Reader

# SSL handshake using BIO over RFID
# Several read/ write_multiple_request /respond messages
# Agreement on cipher suite and generate session key etc.

# Authentication fails , thus there is no SSL connection
# Query Dustbin Tag
query P15693
{
    flags = FL_INVENTORY;
    command = INVENTORY;
};

```

| From | To | Goal | Result |
|--------------|--------------|------------------|--------|
| Truck Reader | Guardian | Inventory | ALLOW |
| Guardian | Truck Reader | Start Connection | ALLOW |
| Truck Reader | Guardian | Hand shake | ALLOW |
| Guardian | Truck Reader | Hand shake | ALLOW |
| Truck Reader | Guardian | Authentication | ALLOW |
| Guardian | Truck Reader | Authentication | ALLOW |
| Truck Reader | Dustbin Tag | Query | ALLOW |
| Truck Reader | MY_TAGS | Query | DENY |
| Guardian | Truck Reader | Close Connection | ALLOW |

Table 2.14: RFID Guardian Query Result: Waste Management

For “Waste Separation”, some researchers proposed use of RFID tagged items to support separation process by placing a reader on the disposal device which distinguishes between goods that should be separated and goods for incineration. [14] In their proposal, no specific device is specified, but only that the purity of the recycling input goods could be improved and the burden on the incineration plant could be relieved. It is clear that this is another potential application for RFID technology.

The difference between these two applications lies in how RFID tags are read by RFID reader. In the “Automated Data Collection”, information from the dustbin is used; while in the “Waste Separation”, information from garbage itself is used.

2.4 Summary

This chapter investigated the usage of RFID Guardian in real applications. For each scenario, the background was briefly introduced, and then a UML object model and a sequence model was given. After that, a Guardian access control list, together with a sample reader file, a tag file and a query result, was shown. The aim of all these examples is to show that RFID Guardian can protect privacy by only allowing the right RFID Reader to query the right RFID tags.

Note that also this chapter also paid more attention to personal privacy than commercial security by looking at scenarios found in an individual's everyday life.

- Contact-less Smart Card
- Shopping
- Library
- E-Passport
- Car key-less entry system
- Home appliance
- Animal Identification

Of course RFID Guardian can be used in supply chains as well (see the military and commercial scenarios). The difference is that companies (or armies) must install several RFID Guardians not just one.

The radio range of RFID Guardian is approximately 1 meter, so if a person hangs an RFID Guardian on his belt (middle of his human body) (Figure: 2.60), all that person's items can be protected, including cards, wallets, keys, clothes, etc. However, more Guardians are needed for trucks (Figure: 2.61).

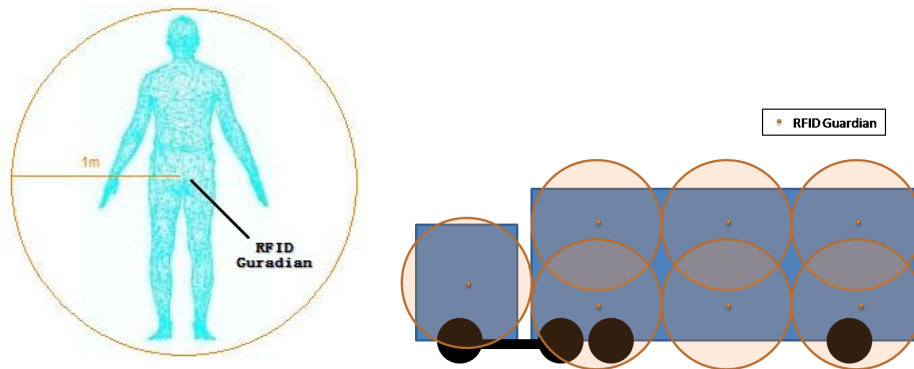


Figure 2.60: Range of a personal Guardian

Figure 2.61: Range of Guardians for a truck

Sometimes, RFID tags that are not owned by the user may come into the vicinity of RFID Guardian. For example, when the user is sitting in a casino while the operator's reader is querying RFID chips, or when the user is walking past a waste disposal truck while the truck is querying the RFID dustbins. In this case, RFID Guardian should leave the radio traffic alone; these readers are doing their own job.

In summary, when provided with the correct Access Control List, an RFID Guardian protects the user's privacy and will not interfere with other interactions.

Chapter 3

Authentication

Previous chapters described how RFID Guardians can be used to prevent unauthorized queries, and RFID Readers must prove to an RFID Guardian that they have the right to query an RFID tag. In other words, RFID readers have to authenticate themselves to the RFID Guardian before querying. Conversely the RFID Guardian sometimes needs to authenticate itself to an RFID reader. For example in the supermarket scenario, when the reader is ready to transfer the ownership of what the user's bought to the user's Guardian it has to be sure that it is talking to the correct Guardian. This authentication phase is shown in Figure: 3.1. This chapter describes the (mutual) authentication between RFID Reader and RFID Guardian.

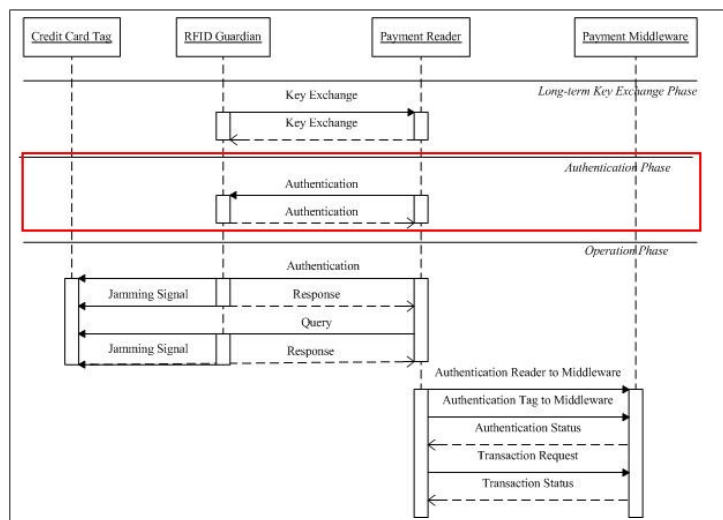


Figure 3.1: UML Sequence Model Sample: Credit Card

As already described, the communication between an RFID Guardian

and Guardian-aware readers is based on SSL. The designers choose this because SSL is the most widely used tool to provide a secure connection over an insecure medium. But if a better security tool appears then this design decision could be revised. This section therefore looks at security protocols generally, rather than focusing on SSL exclusively.

This chapter is organized as follows: In section 3.1, existing security techniques are reviewed, notably asymmetric key protocols and symmetric key protocols. Then in section 3.2, specifics about advantages and disadvantages of applications to RFID Guardian are discussed. Section 3.3 revisits each user case. Then in section 3.4, conceptual pictures of RFID Guardian user interfaces for authentication choices are drawn. Finally section 3.5 is the conclusion.

3.1 Introduction

RFID Guardian and RFID readers talk to each other over broadcast radio waves which can be heard by any nearby receivers. Therefore, it is important they use cryptographic services that prevent others from reading their messages. Another benefit of cryptography is that it can provide authentication between each participants. Generally speaking, cryptographic techniques are divided into two categories: Asymmetric Key Protocols and Symmetric Key Protocols. These techniques are reviewed below.

3.1.1 Asymmetric Key Protocols

Asymmetric key protocol (also known as Public Key Cryptography) is an arrangement that makes use of a pair of keys - public key and private key. Users keep the private key secret and publish the public key, i.e. by declaring it on a website. So when Alice wants to send a message to Bob, then Alice should first find Bob's public key, then Alice should encrypt that message with that key. When Bob receives this encrypted message, he can decrypt it with his private key. Even if that message may be overheard by somebody else (say Carol), that person cannot read the message, because Carol does not have Bob's private key. In other words, the two parties that are communicating do not need to share secrets. Because of this important feature asymmetric key protocol has been widely deployed in many applications: authentication of web services, secure email, electronic signatures, etc.

A central problem with asymmetric key protocol is proving a public key is authentic, and has not been tampered with or replaced by a malicious third party. Suppose that Carol declares her public key to be Bob's, and Alice believes this declaration for whatever reason. Then when Alice encrypts her message and sends that to Bob, Bob cannot decrypt that message, because

Bob does not have the right private key. But if Carol can get this message, she can decrypt it! In practice, the usual solution is to use a Public Key Infrastructure (PKI), in which we use certificates to verify the relationship between the public key and the entity who claims ownership of it.

A certificate is a piece of message that reliably associates a particular identity to a particular public key. This reliability is guaranteed by the fact that the certificate is signed and issued by a Trusted Third Party (usually known as Certification Authority (CA)) that all the parties of the system assume trust with respect to issuing authentic certificates.

Certificate Authority

Certificate Authority (CA) is the organization that generates certificates in the PKI and has the responsibility to certify ownership of key pairs. Two basic organization models of CA are:

- **Centralized CA:** Most centralized CAs are commercial and make profit by signing the customer's public keys. Usually when an individual person or an organization needs a certificate from the CA, public key must be sent to the CA, together with documents proving the authenticity of the key and of the entity requesting the certificate. This conventional evidence bootstraps trust in the digital scenario. For an individual, documentation could be an Identification Card; for an organization, it could be its registration with the chamber of Commerce. When the CA is convinced that the public key really belongs to the identity, then it will sign the identities certificate. Some centralized CAs have such a good and global reputation that certificates signed by them can be recognized almost anywhere. The market leading CA is VeriSign (<http://www.verisign.com>).

One benefit of a centralized CA is that if the CA has a good reputation it can be trusted by everyone. There are however two disadvantages: 1) it means extra charges for the user, and 2) the user may have to wait a few weeks.

One important feature of certificates is that they can be chained, i.e. each certificate can refer to sequence of previous certificates. The purpose of a certificate chain is to establish a chain of trust from a 'peer' certificate to a trusted CA certificate. The CA vouches for the identity in the peer certificate by signing it. If the CA is one that you trust (indicated by the presence of a copy of the CA certificate in your root certificate directory), this implies you can trust the signed peer certificate as well. [21]

- Localized CA: if the public key is not used in a generic *public place*, but instead only within a single organization (i.e. a small company) then there is no need to use to a centralized CA. For example, some buildings use an RFID card as proof of entry. In this case there is no need to have VeriSign sign the certificates; the administration office can instead set up a local CA and sign these cards directly.

Two advantages with local CA are: 1) there is usually no extra fee. 2) these certificates can be generated immediately. However, the disadvantage is this local CA cannot be trusted by the public.

Different models have different advantages and disadvantages. A more detailed discussion and comparison of CA usage for RFID systems is given in section 3.2.3.

Certificate Revocation

Public key certificates today are usually based on X.509, and in addition to the public key, they tend to carry more information to make them easier to use, including:

- Date of Issue
- Expiration date
- Version of this certificate format
- Indications of types of keys formats
- Owner's name in various formats
- Other properties of the owner (for example, address)
- Owner's rights and privileges

Most of this information is clear: it has either to do with the public key (key format, etc), with the owner (owner's name, address, etc), or with the certificate itself (issue date, certificate format, etc). The purpose of the expiration date is less straightforward. A certificate cannot stay valid forever, because:

- CA might improperly issued a certificate.
- The private key could become compromised.
- The user could lose sole possession of the private key (e.g. the key could get stolen or lost).

So certificates need to be revoked from time to time. Techniques for revocation can be roughly divided into two strategies: on-line and off-line. With on-line revocation, the end user will call a Certificate Authority (CA) on-line (Internet, for example) to check the certificates's real status; With off-line revocation, CAs will periodically publish a good list or bad list of the certificates signed by that CA.

Although on-line revocation provides the best security for end users, in practice off-line revocation is for more common, since it does not require a network facility [22]. This is a trade-off between security and convenience. These two techniques are discussed in section 3.2.5 with reference to RFID Guardians.

Summary

The authentication phase for asymmetric key protocols is certificate-based. When one of the communicating parties receives a certificate from the other party, it needs to check the validity of this certificate (who signs the certificate? is it expired? etc.). This procedure is called Certificate Validation. If the validation process succeeds, both parties are satisfied and the conversation can begin. If this fails, however, there are more options than simply refusing to communicate. These are discussed in more detail in section 3.2.4.

3.1.2 Symmetric Key Protocols

In symmetric key protocols the same key is used for both encryption and decryption. This implies that the symmetric key is shared between both of the two parties that are communicating. This key is typically generated either by the participants who are involving in the communication, or by a third party who is trusted by both of the parties.

The authentication phase for symmetric key protocols is more complicated than asymmetric ones, because the precious key cannot be delivered over an insecure network. There are various so-called "Strong Symmetric Key Protocols" which can generate secure session keys and do mutual authentication for both parties. "Needham-Schroeder protocol" is one which is briefly described here.

Assume that there is a server (called S), which is trusted by Alice and Bob. S also shares symmetric keys with Alice and Bob, which are noted as K_{AS} and K_{BS} . Then the basic steps for N-S protocol are:[24]

1. $A \longrightarrow S : A, B, N_A$

Alice sends a message to the server identifying herself and Bob, telling the server she wants to communicate with Bob. N_A is a nonce number, which is used only once.

2. $S \longrightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
The server generates K_{AB} and sends back to Alice a copy encrypted under K_{BS} for Alice to forward to Bob and also a copy for Alice. Since Alice may be requesting keys for several different people, the nonce assures Alice that the message is fresh and that the server is replying to that particular message and the inclusion of Bob's name tells Alice who she is to share this key with.
3. $A \longrightarrow B : \{K_{AB}, A\}_{K_{BS}}$
Alice forwards the key to Bob who can decrypt it with the key he shares with the server, thus authenticating the data.
4. $B \longrightarrow A : \{N_B\}_{K_{AB}}$
Bob sends Alice a nonce N_B encrypted under K_{AB} to show that he has the key.
5. $A \longrightarrow B : \{N_B - 1\}_{K_{AB}}$
Alice performs a simple operation on the nonce, re-encrypts it and sends it back verifying that she is still alive and that she holds the key.

Detailed description of N-S protocol can be found in [25]. However, finding an online trusted server is very difficult and not applicable in most cases. So another interesting strong symmetric key protocol is Encryption Key Exchange, or EKE, which is a password based protocol. If we assume that Alice and Bob already shares a password P , then [23]

1. $A \longrightarrow B : A, \{E_A\}_P$
A generates a random public key E_A and encrypts it in a symmetric cryptosystem with P . And sends that message to B.
2. $B \longrightarrow A : \{\{R\}_{E_A}\}_P$
Sharing the password, B is able to decrypt to obtain E_A , generates a random secret key R , and encrypts in in both E_A and P .
3. $A \longrightarrow B : \{Chanllenge_A\}_R$
A generates a unique challenge $Chanllenge_A$ and sends it to B.
4. $B \longrightarrow A : \{Chanllenge_A, Chanllenge_B\}_R$
B decrypts that message to obtain $Chanllenge_A$, generates a unique challenge $Chanllenge_B$, and sends both challenges back to A.
5. $A \longrightarrow B : \{Chanllenge_B\}_R$
A decrypts to obtain $Chanllenge_A$ and $Chanllenge_B$ and compares the former against her earlier challenge. If it matches, she encrypts $Chanllenge_B$ with R and sends that back to B.

Summary

In summary, strong symmetric key protocols either involve third party (Needham-Schroeder Protocol), or are password-based (EKE). Even if the two communicating parties have some long-term symmetric keys, they should avoid using these keys directly, but try to use short-term or even one-time session keys. Otherwise eavesdroppers can intercept their messages and perform a Dictionary Attack.

3.2 Technique Discussion

3.2.1 Asymmetric vs Symmetric Key Protocols for RFID Guardians

This section does not discuss the general differences between these two techniques, instead focuses on the authentication issues.

Asymmetric Key Protocols

The advantage of using certificate based authentication protocols is that the public keys and certificates can be sent over insecure medium, even if adversaries may be listening. The reason is that certificates are affixed with Digital Signature by the CA. Therefore, it is generally impossible for an attacker to create bogus certificates.

Another advantage of using asymmetric key is that the two participants do not have to share any secret before they communicate. This feature is especially useful when the Guardian user is concerning about privacy problems. Consider this scenario. A supermarket places an RFID reader at its check out desk (“Supermarket” scenario in previous chapter). Since the shop cannot predict who its customers will be, asymmetric key protocols are a natural choice. This allows the shop to publish its public key (possibly on its website), and invites its customers to install that key.

The disadvantage of using asymmetric key protocols, however, is that you have to know each other’s public keys in advance. Of course, with the current implementation based on SSL, the two communication participants can exchange their public keys at the beginning of communication (handshake phase). But this introduces another problem — what if one party does not trust this public key or the certificate that signs that key? Otherwise, the two parties have to use either plain text or one-time symmetric key.

Symmetric Key Protocols

The advantage of strong symmetric key authentication is that all the nonces are fresh and it is difficult for eavesdroppers to perform dictionary attacks.

The disadvantage is that there are always extra requirements: if the protocol is N-S, there has to be an active on-line server for authentication and key generation; if the protocol is EKE, there has to be a shared secret beforehand.

Another disadvantage is that there is a potential threat against the Guardian holder that the reader holder could deliberately “lend” this shared symmetric key to hostiles. Even though Guardian’s “context” feature could help its user to a certain extent, this does not solve all problems. For example, when the Guardian user enters a supermarket, the Guardian could already be switched to a context that is ready to exchange messages with the reader; Then, a hostile with the symmetric key from the supermarket could begin a conversation to the Guardian. Therefore, for security reasons, it is recommended that RFID Guardian give the user a warning, i.e. a beep, so that the user knows that the Guardian is changing information with a reader. It is better that the user can see and recognize the RFID reader visually, so that she knows for sure which company is querying her tags. After that, RFID Guardian may ask the user for permission for communication, i.e. click a “OK” button or enter a password on the pin pad. Of course, if the reader holder and Guardian holder knows and trusts each other, they can change symmetric keys beforehand.

Some people may argue that use of symmetric key protocols is also a threat because the reader’s owner will record Guardian user’s habits. For example, there is already some reports saying that some inns are eavesdropping their customers’ choice of wines using RFID technology. But this also holds true for asymmetric key protocols — by sending the certificate to the Guardian, asymmetric-key-readers could also authenticate themselves to the Guardian. And that is the reason why the designers of the RFID Guardian introduced the “context” feature.

3.2.2 Key Generation

This section assumes that all keys are pre-generated and stored inside RFID Guardian and RFID readers, so that they are available when a conversation begins. For asymmetric key protocols, the keys this concerns are the private keys; and for symmetric key protocols, this either concerns the password or long-term key in EKE, or the symmetric key for the authentication server in N-S.

Asymmetric Key Protocols

As described previously, asymmetric key protocols use a pair of keys: the *private key* and the *public key*. These key pairs have the unique mathematical property that the public key can be quickly generated from the private key, but it is not practical for attackers to deduce the private key from the public key. This means that the public key can safely be published and distributed without concern that it could fall into the hands of attackers.

The private key, however, must remain secret to avoid impersonation. Generally speaking, each party generates their own key pairs. The aim is obviously to keep the private key secret. For the reader, in most cases, the key pair should be generated by the company holding that reader; for the Guardian, the key pair can be generated by the holder herself. So when the Guardian appears near the reader, they could start to exchange their public keys and try to authenticate with each other. Of course, if the reader is held by the Guardian user, it should be the Guardian user who will generate the private key.

In certain cases however, the private key can also be generated by others, i.e. the factory that produces RFID readers. This is convenient if the reader-holding company a) does not have hardware or software to generate key pairs, or b) forgets to generate its own pairs. Of course, the basic assumption is that the RFID reader customer is willing to trust RFID reader manufacturer to a certain extent, which is akin to the fact that sometimes manufacturers of tumbler lockers preset the codes for us.

Symmetric Key Protocols

In symmetric key protocols, however, the symmetric key acts as the base secret and it is used both for cryptography purpose and authentication purpose. So it is quite obvious that this key should be generated by the communication parties and this is indeed what happens in most cases. Take the “Supermarket” scenario as an example. When the Guardian holder first visits a supermarket, the supermarket can ask this customer to share a symmetric key which they can then use for future authentication.

If the Guardian holder prefers N-S protocol to password-based protocol, then the Guardian holder and the reader holder have to find a mutually trusted on-line server, which is very difficult.

In rare cases, however, the secret key can also be generated by some trusted third party. Take the “home appliance” scenario as an example. The owner of RFID Guardian also has some RFID readers at home. Since the owner controls both RFID Guardian and these readers (acting as the trusted-third-party), the owner could generate symmetric keys for radio

communication between them. Of course, this key can also be generated by somebody else as well, i.e. family members.

3.2.3 Certificate Authority in RFID Systems

When the reader-holding organization is sufficiently large that it is organized into a hierarchy with many readers held by different layers, it seems better for it to choose a centralized CA for certificate signing and to certificate chains. In this way, the centralized CA does not need to sign each certificate, but only the root certificate of that organization. The root certificate can also be used to sign other individual certificate for that company. In the “Smart Card” scenario (shown in Figure 3.2). The smart card company or bank, i.e. ABN AMRO, has its root certificate signed by VeriSign, and signs each RFID Reader certificate with that root certificate. Therefore, when an RFID Guardian receives such a certificate, the Guardian knows that: if I trust VeriSign, I should trust ABN AMRO’s root certificate; if I trust the root certificate, I should trust this reader’s certificate. In this way, that individual RFID reader finally gets itself trusted by RFID Guardian.

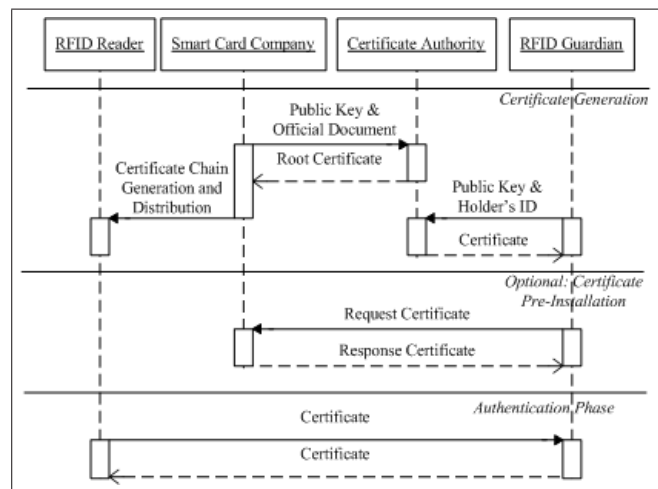


Figure 3.2: Centralized CA Sample: Smart Card

On the other hand, when the organization can be sure that all its customers will trust its own root certificate (even if that root certificate is signed by itself), then it is best to choose a local CA. For example, the library of Vrije Universiteit serves mainly VU staffs and students, and it only has several RFID readers. Then the simplest way for this library is to set up a local CA and signs certificate in each reader with that root CA. When a staff or a student first registers at this university, the administration office

could politely asks him/her to accept the local root certificate of the university's library. This local CA can also create certificate chains, as long as the Guardian holders are ready to accept the root certificate, as can be seen in Figure 3.3.

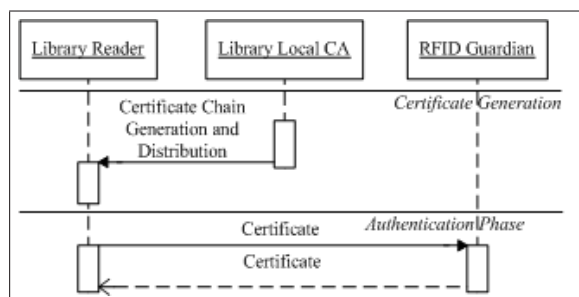


Figure 3.3: Local CA Sample: Library

3.2.4 Certificate Validation in RFID Guardian

Modern web browsers often install root certificates of well-known CAs for the convenience of users. RFID Guardian could do the same, since RFID readers may also have their certificates signed by those CAs. But the problem remains if a company has a certificate signed by a CA that the Guardian does not recognize. In this case, what actions should RFID Guardian take? For example when a user enters a new supermarket or goes abroad, the Guardian may receive an unauthorized certificate from an unknown RFID Reader. Some options are:

- If the verification of certificate fails, the Guardian automatically treats the reader as anonymous, and the Guardian's further action will be defined according to its access control list.
 - Advantage: Best security: usually the ACL will deny anonymous readers.
 - Advantage: The Guardian will not bother its holder.
 - Disadvantage: sometimes the holder needs to know the query is going to happen, and the Guardian should actually accept that certificate, i.e. the shop reader is going to contact the Guardian holder's contactless smart card to finish the transaction.
- Upon verification failure, the Guardian informs the user to ask for further decisions.

- Advantage: User is aware of what is happening and can choose the correct reader, for example, the reader in visual range.
- Disadvantage: In some situations, there is just no need to bother the user. For instance, when the user is passing by the front of a store while the store is using its own reader for shipping items and not intend to query the user's tags.
- Mixed Solution: context + Inform User. The RFID Guardian has a pre-defined parameter called “context”, it can understand the current environment to a certain extent. Therefore, RFID Guardian does not have to inform the user all the time. For example, if the user feels or hears that some strangers are trying to track her on the street, she can set the Guardian context to “DANGEROUS”, so that the Guardian will automatically deny all queries. In other situations however, the Guardian may choose to inform the user. But if the Guardian switches contexts based on the signals from the Reader, then how can we authenticate the reader for the context switch? This is a “chicken-egg” dilemma. One solution may be to allow only trusted to switch the context. For example, a reader can be placed at the door of a user's home so that when the front door is reached it switches the context. Since this reader is installed and controlled by the user, it can be trusted. To guarantee a higher security level, the Guardian should inform the user whenever the context switches.

3.2.5 Certificate Revocation in RFID Guardian

If we focus on RFID technologies only and assume that all CAs can provide on-line service, then the question is; do RFID end users have external network access?

On the RFID reader side, notice that most RFID readers are connected to an RFID Middleware. This is because RFID tags usually return just their tag IDs to the reader; a single number with no other meaning. The RFID systems need extra information to associates that particular number in their middleware (typically a database). This moves it easier for the reader to access external network via the middleware. For example, the reader-holding company can establish an connection from its RFID middleware to Internet, which can be used to access the CA. Then the reader can contact the CA in real-time to verify certificates from the Guardian. Of course, the company needs a firewall or other security techniques to protect its database from being attacked. But the point here is that on the reader side, there is generally not a problem for on-line revocation.

For the Guardian, however, there is a different story. Remember that most RFID readers are placed in public areas, so the Guardian holder must

operate RFID Guardian there, too. So obviously a computer will seldom be available to access the Internet through cables. Therefore, the Guardian designers instead implement the bluetooth protocol on RFID Guardian, so it can communicate to external devices via bluetooth, i.e. mobile phones and PDAs. The RFID Guardian can thus use these devices to access the Internet to check certificate status. So if we assume that

1. The Guardian holder is carrying a bluetooth device.
2. That device could act as a bridge for the Guardian to access Certificate Authority.
3. CAs are providing on-line revocation service.

Then it will be appropriate for the Guardian to deploy on-line revocation.

Nowadays, more and more mobile service providers are offering internet facility, i.e. email checking. Also there is a trend for mobile phones to have bluetooth modules installed, although the current bluetooth implementation is restricted to data transfer and data storage, i.e. transferring photos. Perhaps in the future there could be a business agreement so that the mobile phone manufacturers could allow Guardians to communicate with mobile phones.

In the meantime the most convenient and practical approach is for the Guardians to choose off-line revocation. This implies that Guardian holder must ensure that the Certificate Revocation List (CRL) is inserted into RFID Guardian periodically, either by manually or by RFID Guardian downloading the information automatically if there happens to be a PC nearby.

3.3 Authentication in our Usage Scenarios

3.3.1 Contactless Smart Card

Most contactless smart cards are produced by big banks, and such banks usually have many branches therefore many card readers distributed in different areas. This means asymmetric key protocols are their first choice because the Guardian holder can easily download the public key from its webpages. Of course the Guardian and the card reader can still use their public key to exchange a short-term symmetric key, and use that symmetric key for further communication.

If they indeed agree on asymmetric key protocols, then the authentication process will be based on certificates. As explained earlier, in order to simply things, it is best for the bank to choose a commercial CA. After all,

such a successful and well-known bank should not care about some charges for certificate signing!

Of course, if the reader-holding organization instead chooses to use a symmetric key, or the certificate verification fails (i.e. the certificate expires), then the Guardian and the reader have to use short-term secret keys.

3.3.2 Supermarket

As discussed before, the cryptographic protocols for shop readers could be either asymmetric or symmetric. In case of asymmetric key protocols, a two or three layered certificate chain signed by a well-known CA would be convenient for the Guardian holder. But if the shop instead decides to set up a local CA then if its customers are Guardian holders, they will have to perform extra step of installing that certificate themselves. If the customers can be sure that the supermarket will not leak symmetric key information to others, then they could also choose to use EKE symmetric key protocols.

3.3.3 Library

People usually prefer to go to the nearby libraries, so the customers of a library is almost fixed. Nobody will fly across the Pacific Ocean if he could find exactly the same book in a library down the street — this is a matter of convenience. Therefore, the libraries could possibly start with a local CA, and asks its customer to accept its root certificate when the customer registers at the library.

3.3.4 E-passport

Passports are held by the total population, and the country will have many airports and police stations. So perhaps symmetric key protocols are not fit for this scenario, and instead asymmetric key protocols and certificate should be used for authentication. It is obvious that a centralized CA could help in this scenario. But some citizens or governments may not like their identification document information to be accessed by a business, so they may prefer for the national security department (or whatever government department) to set up a local CA.

3.3.5 Car Keyless Entry System

Although the RFID Reader in this scenario is owned by the user (actually the car, not the reader), it is impossible for the user to operate the reader directly, because the reader is usually firmly attached inside the car. So

a symmetric key protocol is probably more appropriate in this user case. The reason is that the reader is not publicly reachable, and there is no need for the automobile company to publish a public key for a particular reader, then asks all Guardians to communicate with that reader. So it may be more convenient for the automobile company to generate a secret key for each car, and pass that key to the Guardian holder when the Guardian holder becomes the owner of the car.

3.3.6 Home Appliance

By placing RFID readers at home, one can

- Use intelligent household appliance inside the house
- Have separate readers to do triggering. For example, by placing an RFID reader at the front door, one's Guardian could be triggered to switch to another context when the user leaves home.

Since the Guardian holder controls both the Guardian and the reader, there is no need to implement a PKI and use a certificate for authentication. But a symmetric key could be used with long-term keys for both encryption and authentication.

3.3.7 Animal Identification

Most animal identification systems have their shelters distributed across the country, or even across the continent. So it is easy for them to implement asymmetric key protocols and use certificate for authentication. Perhaps they can construct a certificate hierarchy by geographic locations. For example, the top layer represents the whole country, while the second layer represents provinces or states, then the third layer represents cities or towns. This means a centralized CA is best suited for this scenario.

3.3.8 Military and Commercial Supply Chains

The size of a commercial supply chain varies, so it cannot simply say whether asymmetric key or symmetric key would fit all cases. For a small chain (for example only three suppliers), a symmetric key could be used. They could exchange the secret key as soon as the managers sign the contract. On the other hand if the chain is large, they could perhaps better choose a asymmetric key protocol. For the certificate-based authentication process, it may be better for the customer (i.e. Wal Mart) to set up a local CA. The

reason is that all the suppliers are stake holders within this chain and their profits are restricted by the business contract.

The military supply chains always require a higher security level, so convenience could be sacrificed. For example, a new symmetric key could be generated for each stop, which could then be securely stored in all Guardians and readers through the internal military network. In this way, any other readers that do not know the correct symmetric key can be detected.

3.4 Conceptual User Interface

This section provides a conceptual user interface of RFID Guardian. The aim is to convince the readers that RFID Guardian could give them the freedom to choose authentication protocols and let them know what is going on as the communication between RFID Guardian and RFID reader starts, so that the Guardian users could find the best way to protect themselves.

Suppose that an RFID Guardian now enters the vicinity of a reader and is detected by the reader. Then they are ready to communicate.

| RFID Guardian User Interface | Radio Message | RFID Reader User Interface |
|--|------------------------------------|---|
| RFID Guardian Context: Common RFID Reader found | $\leftarrow \underline{Inventory}$ | RFID Reader: 123456 Waiting |
| RFID Guardian Context: Common Reply with Spoof UID | $\underline{SpoofUID} \rightarrow$ | RFID Reader: 123456 RFID Guardian (ID: 654321) found |
| RFID Guardian Context: Common RFID Reader Information ID: 123456 Name: Unknown | $\leftarrow \underline{ReaderID}$ | RFID Reader: 123456 Reply with ID |

Now the RFID Guardian and RFID reader know each other's existence. Whether and how the Guardian should inform its owner are left to future research. But here is one suggestion — maybe the Guardian could give its user a beep and display the following dialog to ask whether the user would like to start conversation with that reader. If the user does not reply within say, 10 seconds, then the Guardian will automatically deny that reader.

RFID Guardian Context: Common

Start conversation with reader

ID: 123456 Name: Unknown

The advantage of this approach is that the Guardian would always inform its user, whereas the disadvantage is that sometimes the Guardian user would prefer the Guardian to react automatically to the reader according to its context setting. But let's assume now the Guardian user would like the Guardian to talk to this reader, then the first thing is to select an authentication protocol. Possibly using this screen;

RFID Guardian Context: Common

Please select authentication protocol

Two options are offered: asymmetric key protocols and symmetric key protocols. For both options, the RFID Reader side will be aware of the choice as soon as the SSL handshake starts, because in the first message (ClientHello) sent by RFID Guardian, the ciphersuites are announced.

For asymmetric key protocols, the Guardian and reader will authenticate each other with certificates. This is fairly easy with the current SSL implementation. For symmetric key protocols, it is more difficult, because the original SSL/TLS library used public key certificates for authentication by default and it had little support for symmetric key protocols[27]. As explained in section 3.1, the two basic symmetric key authentication protocols are NS and EKE, but the SSL/TLS protocol ([27]) supports neither. It was not until the introduction of RFC 2712 [28], did SSL/TLS start to integrate Kerberos protocol cipher suites. Kerberos is even better than N-S protocol, because it has robust protection against replay attack, which N-S lacks. Finally after a delay of six years, RFC 4279 (also known as TLS-PSK) was released to support even more symmetric key protocols. TLS-PSK is short for "Pre-Shared Key Ciphersuites for Transport Layer Security"[29]. Here "pre-shared" keys are symmetric keys shared in advance between the communicating parties. The trick of TLS-PSK is to add some extra symmetric-ciphersuites (i.e. TLS_DHE_PSK_WITH_RC4_128_SHA) to TLS, then the

client indicates its willingness to use pre-shared key authentication by including one or more PSK ciphersuites in the ClientHello message. A third symmetric key protocol is called TLS-SRP, which means to use “Secure Remote Password” protocol over TLS for authentication. But TLS-SRP still requires a certificate for authentication, so it could be regarded as not a pure symmetric key protocol, and therefore will not be described it here.

In order to simplify things, the following assumptions are made;

1. The Reader and Guardian have the same set of encryption cyphercuites.
2. The Reader and Guardian are going to establish a typical mutual authentication conversation.

In the following pictures, the word “Request” is used to stand for the request of one party to the other party asking for authentication material.

If the user selects asymmetric key protocols, then the interaction is as follows;

| RFID Guardian User Interface | Radio Message | RFID Reader User Interface |
|---|----------------------|---|
| RFID Guardian Asymmetric Setting up communication channel with reader (ID: 123456) <input type="button" value="Cancel"/> | <u>Connect</u> → | RFID Reader: 123456 Guardian (ID: 654321) connecting <input type="button" value="Cancel"/> |
| RFID Guardian Asymmetric Connected! with reader (ID: 123456) <input type="button" value="Cancel"/> | ← <u>Accept</u> | RFID Reader: 123456 Accept Guardian (ID: 654321) <input type="button" value="Cancel"/> |
| RFID Guardian Asymmetric Start SSH handshake! with reader (ID: 123456) <input type="button" value="Cancel"/> | <u>ClientHello</u> → | RFID Reader: 123456 Guardian (ID: 654321) requests certificate <input type="button" value="Cancel"/> |

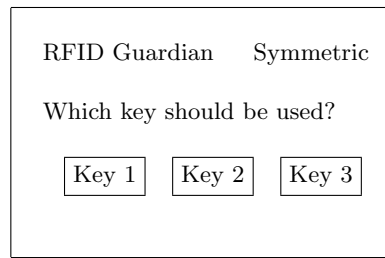
| | | |
|--|--|---|
| <p>RFID Guardian Asymmetric</p> <p>Certificate received from reader (ID: 123456) Trust Chain: Veri→Lidl→Uilenstede Please Select</p> <p>Accept Once Always Accept</p> <p>Detail Reject</p> | <p><i>ServerHello, Certificate, Request, ServerHelloDone</i></p> | <p>RFID Reader: 123456</p> <p>Send certificate to Guardian (ID: 654321)</p> <p>Cancel</p> |
|--|--|---|

At this point, the user needs to select whether or not to trust this certificate. If the user does not trust this certificate, “Reject” should be clicked; Otherwise, either “Accept Once” or “Always Accept” should be chosen. The user could also check the digits by clicking the “Detail” button. Suppose that the user decides to continue the conversation, then

| RFID Guardian User Interface | Radio Message | RFID Reader User Interface |
|--|---|---|
| <p>RFID Guardian Asymmetric</p> <p>Sending Guardian’s certificate to reader (ID: 123456)</p> <p>Cancel</p> | <p><i>Certificate, Pre-Master, Finish</i> →</p> | <p>RFID Reader: 123456</p> <p>Certificate received from Guardian (ID: 654321)</p> <p>Cancel</p> |
| <p>RFID Guardian Asymmetric</p> <p>Communication channel set up with Reader (ID: 123456)</p> <p>Stop</p> | <p>← <i>Finish</i></p> | <p>RFID Reader: 123456</p> <p>Communication channel set up with Guardian (ID: 654321)</p> <p>Stop</p> |

The communication channel between RFID Guardian and RFID Reader is established, and from now on the conversation will be secured by the session key computed from the Pre-Master secret.

If the user selects a symmetric key protocol, then maybe the user needs to select which symmetric key should be used for this Reader, like below,



This assumes that user possesses different symmetric keys for different parties. However, since the Reader already tells the Guardian its ID, then maybe RFID Guardian could help user remember which key should be used for a particular reader thus will not bother the user. This is left to future research.

A user can only chooses a Kerberos-based symmetric key, if a service ticket is held for the RFID reader. If not, then the Guardian must contact a Kerberos authenticator (also called “Key Distribution Center”) online.

| RFID Guardian User Interface | Radio Message | Kerberos Authenticator |
|--|------------------|------------------------|
| RFID Guardian Context: Common Contacting Kerberos authenticator | <i>Request</i> → | |
| RFID Guardian Context: Common Kerberos key received! | ← <i>Ticket</i> | |

Note that the messages in above pictures do not describe exactly how RFID Guardian, as a Kerberos Client, authenticates to the Kerberos Authenticator. For more information see [30].

Then the conversation between RFID Guardian and RFID Reader can continue as follows,

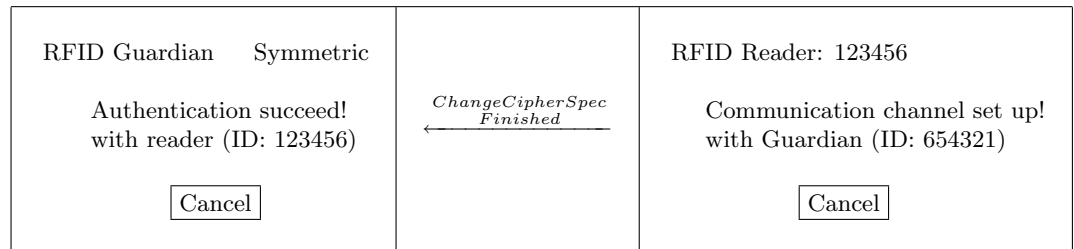
| RFID Guardian User Interface | Radio Message | RFID Reader User Interface |
|--|------------------|--|
| RFID Guardian Symmetric Setting up communication channel with reader (ID: 123456) <input type="button" value="Cancel"/> | <i>Connect</i> → | RFID Reader: 123456 Guardian (ID: 654321) connecting <input type="button" value="Cancel"/> |

| | | |
|--|---|---|
| RFID Guardian Symmetric Connected! with reader (ID: 123456) <input type="button" value="Cancel"/> | $\xleftarrow{\text{Accept}}$ | RFID Reader: 123456 Accept Guardian (ID: 654321) <input type="button" value="Cancel"/> |
| RFID Guardian Symmetric Start SSH handshake with reader (ID: 123456) <input type="button" value="Cancel"/> | $\xrightarrow{\text{ClientHello}}$ | RFID Reader: 123456 Start SSL Handshake! with Guardian (ID: 654321) <input type="button" value="Cancel"/> |
| RFID Guardian Symmetric Receive one-time session key from reader (ID: 123456) <input type="button" value="Cancel"/> | $\xleftarrow{\text{ServerHello, ServerKeyExchange, ServerHelloDone}}$ | RFID Reader: 123456 Sending one-time session key to Guardian (ID: 654321) <input type="button" value="Cancel"/> |
| RFID Guardian Symmetric Sending authentication key to reader (ID: 123456) <input type="button" value="Cancel"/> | $\xrightarrow{\text{ClientKeyExchange ChangeCipherSpec Finished}}$ | RFID Reader: 123456 Authentication key received from Guardian (ID: 654321) <input type="button" value="Cancel"/> |

Note that the Kerberos ticket is included in the “ClientKeyExchange” message if the user specifies use of Kerberos keys. Also the “authentication key” sent from RFID Guardian is not the precious symmetric key, but just a temporary key computed from the pre-shared symmetric key plus some other stuff, for example, the reader and Guardian’s ID number. After receiving this key, only the correct reader is able to extract the identity of the Guardian and verify the challenge from the Guardian.

Lastly, RFID Guardian and RFID reader exchange the finished message to complete the handshake. For more information about TLS-PSK or Kerberos-TLS see [29], [28] and [26].

| | | |
|---------------------------------|------------------|-------------------------------|
| RFID Guardian User Interface | Radio Message | RFID Reader User Interface |
|---------------------------------|------------------|-------------------------------|



3.5 Conclusion

This chapter analyzed the authentication process between RFID readers and RFID Guardians. Both asymmetric and symmetric key protocols can serve this purpose. Every protocol has its own advantages and disadvantages; but certain protocols are more suited for specific scenarios. Also, by providing a Graphics User Interface, RFID Guardian gives its user useful information and lets the user choose the desired authentication all by herself, even though the radio messages are not visible.

Chapter 4

Software Regression Test

The previous chapters showed that, in theory, RFID Guardian can use access control list to protect RFID tags. But in practice, this goal needs a software implementation, which has to be reliable enough to understand the ACL correctly. Therefore, this chapter contains a report of testing on the software that will be installed on RFID Guardian. Such testing is useful because it ensures that many common bugs have been discovered before the software is released and RFID Guardian becomes a real product in industry.

4.1 Introduction

Computer software is essentially written by human beings, and humans make mistakes. It is therefore inevitable that softwares contain errors (called bugs) or mistakes that prevent it from behaving as intended. The results of bugs may be extremely serious. A bug in the code controlling the Therac-25 radiation therapy machine was directly responsible for some patient deaths in the 1980s. Unfortunately (in both theory and practice), it is impossible to eliminate all bugs, especially when the software grows to millions lines of code. Commercial software typically has 20 to 30 bugs for every 1,000 lines of code, according to Carnegie Mellon University's CyLab Sustainable Computing Consortium.

Before software is released, testing is necessary to ensure quality. A common practice is to have testing performed by an independent group after the functionality is developed but before it is shipped to the customer. Another common practice is for tests to be developed during technical support escalation procedures; this is called regression testing. Regression testing seeks to uncover regression bugs. Regression bugs are when software that previously worked stops working correctly or starts to behaving differently.

The software installed on RFID Guardian is called MRG, and it is developed by the research groups at Vrije Universiteit at Amsterdam and Delft University of Technology. This chapter describes the regression test built for this software. Section 4.2 describes the software regression test framework, which is mainly used for testing software correctness and robustness. Section 4.3 described how software efficiency is tested. Finally section 4.4 is the conclusion.

Testing Environment is shown in Table 4.1.

| | |
|-------------------------|-------------------|
| CPU | Celeron 2.7GHz |
| Memory | 256Mb |
| Operating System | Ubuntu Linux 7.04 |
| Compiler | gcc 4.1.2 |

Table 4.1: Testing Environment

4.2 Correctness and Robustness

4.2.1 Framework

In its real working environment, the software maintains a list of known readers, a list of known tags, and, of course, the Access Control List. It then starts reacting to incoming queries. Since the software interacts with files, some tests are easy, because developers can provide sample files themselves. For example, the reserved names for reader files, tag files and acl files are: *example.readers*, *example.tags* and *example.acl* respectively. An optional file is *example.context*, in which users can declare contexts, but user can also do this inside the ACL file. If we put these files into a directory, then we can use the “-i” parameter to ask the software to take these files as input like below, so that we could check if the result are as expected.

```
./main -i ../test request.acl
```

In the above example, “request.acl” is the query file. The software supports other parameters, too. As listed in Table 4.2.

Using these parameters, the working environment can be simulated and regression tests can be performed. The framework of the software regression test consists of a master script and 35 tests. The advantage of using a script is autonomy, meaning that software developer can use that script to automatically run tests and then the developer can check the exit status of the script, so that the developer could focus on the result of the regression

| Parameter | Usage |
|-----------|-----------------------|
| -a | specify an acl file |
| -r | specify a reader file |
| -t | specify a tag file |
| -c | specify a context |

Table 4.2: Software Parameter and Usage

test without being disturbed by the mechanics of the testing process. The master script is written in Perl and it acts as the entry point of all regression tests. The individual tests reside in different sub-directories. The script enters these sub-directories one-by-one to perform individual test. The advantages of organizing regression test in this way is that it is easy to add or remove tests. Here are the detail;

1. The script first examines all entries in the current directory and checks if the element meets the following two requirements:
 - The entry is a directory.
 - The entry's name is not "CVS".

The first requirement is necessary in case the script detects itself ; the second requirement is necessary to prevent the script stepping into its source code control repository. If the entry passes these checks, then the script takes the entry as a test and proceeds to the next step.

2. Then the script prints the test name (actually sub-directory name such as "basic.context"), and checks if this directory contains the default result file name — "result.0". This step is useful because sometimes a new test gets added without a standard result to be compared to. It seems better to check the result file's existence before even executing this test.
3. Now the script checks the configuration file of the test: if there is no configuration file, the script simply passes the default result file name to the check function, which is also the next step; If there is a configuration file, the script reads that file line by line and calculates the corresponding result file name, then passes the configuration statement and the new result file name to the check function. How to edit configuration file and result file is explained in the section 4.2.3.
4. The check function executes the software program with user- specified parameters and decides whether the software reacted correctly by comparing the output stream to the standard result file.

The testing has two aims; The first aim is to test software correctness, meaning that if the user wants to block a reader in the ACL, the software should not allow the query. Since the keyword of software decision contains a “Verdict” , the script will filter the output stream for “Verdict”, and compare the filter result with the standard result file to see if the decisions match. The second aim is to test robustness. For example, if the user provide incorrect input, the software should not crash. Instead, the software should report to the user that the input parameter is wrong. In this case, the software should not provide “Verdict” decisions, but should print an error message and exit. Therefore, the script will open the result file, read the error message and try to capture that message in the software output stream.

In both cases, the check function will returns 0 for success and -1 otherwise.

5. The script will wait for the result from check function. If the checking succeeds, it will print a “passed” message and go on with the next test. However, if the checking fails, the script will print a “failed” message, and exits immediately with status 33.

The current design is that the script exits immediately if an error occurs. This is convenient for the software developer; they can react by checking the temporary file to find where the problem is before the temporary file is replaced by the output of subsequent tests.

4.2.2 Test Cases

The following table is a list of current tests.

| | | |
|-------------------------|-----------------------|-------------------------|
| anti collision | basic context | case sensitive |
| circular tag sets | circular reader roles | deny higher priority |
| different request | does star work | empty role |
| empty rule | empty tagsets | explicit inventory |
| explicit write multi | priority | nested role |
| nested rule | nested tagset | reader include reader |
| reader include role | reader on same line | role include reader |
| role include tagset | same role added twice | same reader added twice |
| same tagset added twice | superset role | superset tags |
| tag include tag | tag include tagset | tag on same line |
| tagset include reader | tagset include role | |

Table 4.3: Test Suites

Below is a brief description of each test.

anti collision RFID Guardian implements anti-collision algorithm, because occasionally it has to act as an RFID reader. This test contains a bash script implementing the same algorithm. The test uses this bash script to generate query files and expected result files. Then the master script runs MRG with the same query files for simulation and tests if the decisions from MRG and the bash script match.

basic context how the software behaves,

- if the user does not provide any context.
- if the user does provide a legal context.
- if the user provides an illegal context.

case sensitive : The current design of MRG is that it is case sensitive, so *reader A* and *reader a* are different readers. This test is added to check if this feature is maintained. Developers should remove this test if case sensitivity changes.

“Circular tag sets” and “circular reader roles” circular definitions of tag sets and roles in tag files and reader files are definitions like this.

| | |
|--------|---------------|
| role A | @tag P15693 A |
| { | { |
| A, | \$A, |
| }; | }; |

These definitions are illegal because they could cause confusion. But these are possible input mistakes by RFID Guardian users. So the standard error stream is filtered for MRG error messages, such as “Cannot lookup reader” and “Cannot locate tag set”. If the script encounters these messages, the software passes this test.

Deny has a higher priority the current implementation of MRG is that it will not depend on the anti-collision algorithm for protecting tags. Suppose that there are two tags: Tag A (0x010101010101) and Tag B (0x000000001), which are both contained in a large group called “MYTAGS”. If the ACL is as follows:

```
rule P15693 DENY
{
    tags = @MYTAGS;
    role = *;
```

```
};

rule P15693 ACCEPT
{
    role = LEGAL_READER;
    tags = TAG_A;
};
```

Then when a legal reader is querying tags like this;

```
query P15693
{
    command = INVENTORY;
    mask_len = 1;
    mask = 0x1;
};
```

Then both tags should match this query because their last tag id both end with 1. So both tag A and B will respond. According to rule No. 2, RFID Guardian should allow this query, because it is the legal reader who is doing query; however, RFID Guardian should deny this query as well, because according to rule No. 1, tag B is what RFID Guardian should protect. Some may argue that RFID Guardian does not have to take any action, because when responses from both tags arrive at the reader the reader will get confused and automatically start the anti-collision process. However the software developers decided that MRG should deny this query anyway. Their justification was that although the software decides actions based on the tag files, this tag file may be *not* up to date. In this example, tag B may be just out of range, or it might even have been left at home. Therefore, they cannot depend on the anti-collision algorithm to take effect. For the test, this principle is called “Deny has a higher priority”, because when implemented in this way, RFID Guardian is going to deny queries as much as it can.

different request In this test case, software is checked to see if it could recognize different kinds of RFID query request, including *inventory*, *write single block* and *write multiple block*.

does star work there are some rules written with a star (*), where star means ‘any’. For example, “*command = **”. This test contains two different ACL files and these files are tested in turn (option “-a” specifies which ACL file should be used for testing). After filtering the output stream, the exact number of denials or acceptance messages are checked.

empty rules, tag sets and roles To test of whether the software will crash if tag sets and roles are empty, and whether they will influence the ACL decision. An earlier version of the software would crash if there was an empty rule in the ACL file, so these tests ensure that this bug stays fixed.

explicit inventory a previous version of the software had the behavior that if the user writes the ACL to deny all kinds of queries,

```
rule P15693 DENY
{
    role = role1;
    command = *;
    tags = @TAGS1;
};
```

then the software behavior was correct. However, if the user specifies that only the *inventory* query should be denied,

```
rule P15693 DENY
{
    role = role1;
    command = INVENTORY;
    tags = @TAGS1;
};
```

then the inventory request was allowed! This test ensures that the bug does not reoccur.

explicit multi another bug was that if the ACL was to deny *inventory*, and then allow *write single block* request, the software would automatically deny a subsequent *write multiple blocks* request, which is of course incorrect. This test was added to ensure this bug stays fixed.

nested rule, role and tagsets When roles and rules are declared nested like below:

```
rule ACCEPT
{
    rule ACCEPT
    {};
};
```

the software will complain about grammar error. But the software does allow nested tag sets, For example,

```
@tag P15693 MYTAGS
{
    tag P15693 = { tagid = 0xE0123456; },
    @tag P15693 = { store = xxx },
};
```

This test is to check if the software accepts this kind of input.

```
@tag P15693 MYTAGS
{
    @tag P15693 {
        tag P15693 = { tagid = 0xE0123456; },
    },
};
```

Result shows it is not allowed.

priority there is an optimization algorithm for MRG to achieve high efficiency, which is explained further in section 4.3. This test is to check if this optimization algorithm still works.

X include Y X and Y could be any of Single Reader, Role, Single Tag and Tag sets. These could be common mistakes for RFID Guardian users.

reader or tag on the same line a previous version had a bug whereby if readers were added to roles in a same line (instead of in separate lines), like this,

```
reader A {};  
reader B {};  
role C  
{  
    A, B,  
};
```

then decision of the software was not determined. This test ensures this mistake stays corrected. Furthermore, an extra test sees what would happen if tags are added to tagsets in the same line.

same X declared twice Here, X could be single reader, roles, single tag or tag sets. These tests are useful because sometimes people use the same name to declare different identities, which costs confusion to RFID Guardian. Some tests are done to check if the software will find same information in its naming space. The key words used here include: “Cannot add role”, “Cannot create new tag list”, “Cannot add reader” and “Tag name already in use”.

simplest accept There is only one tag, one reader and the acl file simply allows the reader to query the tag. This is the simplest scenario, and the software should always pass this test.

simplest deny Almost the same as “Simplest accept” except that the acl file denies query request.

superset role or tagsets sometimes people make mistakes by writing “A is a superset of B is a superset of A”, like below:

| | |
|-------------------------|---------------------------------|
| role A { B, }; | @tag P15693 A { @B, }; |
| role B { A, }; | @tag P15693 B { @A, }; |

So this test checks whether MRG could be fooled by this kind of input.

To sum up, there are already 35 tests in total to test correctness and robustness of the software. These can be grouped into two categories, as shown in the following table. Some tests contain multiple small tests, so they may appear in both categories.

Correctness

| | | |
|----------------------|----------------|----------------------|
| anti collision | basic context | deny higher priority |
| different request | does star work | explicit inventory |
| explicit write multi | priority | simplest accept |
| simplest deny | | |

Robustness

| | | |
|-------------------------|-------------------------|-----------------------|
| basic context | case sensitive | circular reader roles |
| circular tag sets | nested role | nested rule |
| empty role | empty rules | empty tag sets |
| nested role | nested rule | nested tagset |
| reader include reader | reader include role | reader on same line |
| role include reader | role include tagset | same role added twice |
| same reader added twice | same tagset added twice | superset role |
| superset tags | tag include tag | tag include tagset |
| tag on same line | tagset include reader | tagset include role |

4.2.3 How to add a new test?

In order to add a new test, the user should first create a sub-directory in the same directory as the master script. Then the user should put all supplementary files into that sub-directory, including *example.acl*, *example.readers*, *example.tags* and *request.acl*. In addition, the user can choose to provide a configuration file and at least one result file.

The configuration file is named “conf.txt” and contains special parameters to run the software, such as “-a”, “-r”, “-c” or “-t”. Users don’t have to provide “-i” (which means the directory), because the test already resides in a single directory, so the script runs the application with “-i” by default. It is possible that user may want to test software behavior in multiple conditions within a single test . In this case, please put the configuration sentences in different lines in “conf.txt”. For example, the “basic_context” test tests how the software runs;

- if the user does not provide any context.
- if the user does provide a legal context.
- if the user provides a wrong context.

This test has the following configuration file

```
-c DANGEROUS
-c non-exist-context
```

Please note that the first line is empty, because I don’t want to specify any context.

Of course, with different configuration parameters, the software will generate different results. The result file is named as “result.\$number”, where \$number runs in the sequence 0, 1, 2, 3, ... Users should provide at least one result file, so the “result.0” is checked even before the software is run. If there are multiple files, then the subsequent files should be “result.1”, “result.2”, “result.3”, etc. The relationship between the lines of configuration file and result files is shown in following table;

| Line No | conf.txt | result file |
|---------|----------------------|-------------|
| 1. | | result.0 |
| 2. | -c DANGEROUS | result.1 |
| 3. | -c non-exist-context | result.2 |

The content supplied in the result file depends on the purpose of the check. If the user wants to check correctness, then simply filter the output with “Verdict” keyword. For example,

```
./main -i ../basic_context request.acl | grep Verdict  
> result.0
```

If robustness is to be tested, then first run the software and catch the special error message manually. After that, put that message into the result file. Below is a list of error messages that are used in existing tests.

- Cannot lookup reader
- Cannot locate tag set
- Illegal reader attribute clause

4.3 Efficiency

In addition to the software regression test, a special effort was made to test the software efficiency. In theory, the jamming signals from RFID Guardian can arrive at the readers together with the RFID tags simultaneously, which means that software implementation has to be very efficient. And whereas the RFID tags just need to do one job (reply), RFID Guardian needs to do three things.

1. Read the Access Control List
2. Read the Tag file
3. Decide whether or not to send a jamming signal

The software achieves high efficiency by preprocessing, which means that before any queries arrive the software calculates rules for each tags that should be protected. So when the queries arrive, RFID Guardian can use the result of precomputation to decide which tags should protect, thus saving time.

The effect of preprocessing is similar to that of caching, namely that in both cases, a result is stored in memory to save time for future queries. Below (Table 4.4) is the result of a simulation test to illustrate the benefit. In this test a reader broadcasts the same queries 10 times, and the RFID Guardian replies with the same ACL rules and tag sets. The only difference is that first time the software enables preprocessing, but in the second time it does not. The total query time length was measured, with an increasing number of ACL rules to see how behavior changes when the work load becomes heavier. The time is measured in seconds.

| Number of Rules | Preprocess | Non-preprocess |
|-----------------|------------|----------------|
| 1 | 0.000076 | 0.000026 |
| 2 | 0.000123 | 0.000069 |
| 3 | 0.000206 | 0.000147 |
| 4 | 0.000478 | 0.000245 |
| 5 | 0.000431 | 0.000384 |
| 6 | 0.000605 | 0.000659 |
| 7 | 0.000943 | 0.000792 |
| 8 | 0.001057 | 0.000899 |
| 9 | 0.001574 | 0.000839 |
| 10 | 0.002044 | 0.002400 |
| 20 | 0.006804 | 0.006399 |
| 30 | 0.038175 | 0.021965 |
| 40 | 0.049829 | 0.041872 |
| 50 | 0.079150 | 0.083155 |
| 60 | 0.160024 | 0.128865 |
| 70 | 0.224686 | 0.225635 |
| 80 | 0.317000 | 0.352332 |
| 90 | 0.397201 | 0.465492 |
| 100 | 0.647474 | 0.924550 |

Table 4.4: Comparison Result: Preprocessing vs. Non-preprocessing

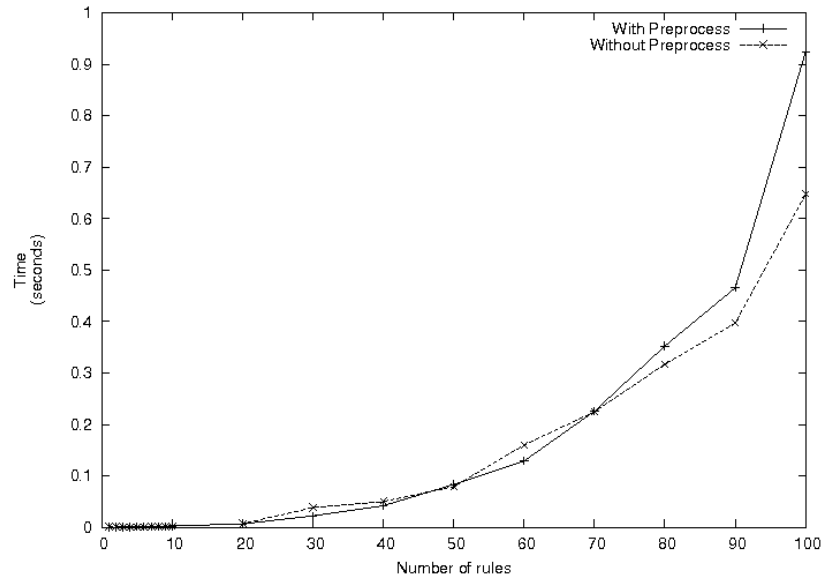


Figure 4.1: Comparison Result: Preprocessing vs. Non-Preprocessing

When these results are visualized (See Figure 4.1) it becomes apparent that the benefit of preprocessing becomes more important as the number of ACL rules grows. If there are 100 rules, preprocessing can save time to as much as 30%!

4.4 Summary

In this chapter, I described the effort and achievement of testing correctness, robustness and efficiency of the software. In fact the regression test framework has already in use by the software developer to improve software quality. Especially convenient is that the script returns different values upon success or failure, so by checking script's return value the software developer could easily find whether the newly introduced feature or modification caused a bug. It is also quite convenient to add or remove tests.

In addition to that, I also tested the software efficiency. I found that by implementing an optimization algorithm, the software could operate in a time-restricted working environment.

So as far as I can judge, MRG is a reliable software for the RFID Guardian. It accurately understands the ACL rules and is robust enough to handle common input mistakes. Also it can protect RFID tags in real-time with high level of efficiency.

Chapter 5

Conclusion

In this thesis, I presented how an RFID Guardian can be used in real application scenarios. I found that in almost all cases, it is best to write the ACL so that it begins with an accept rule, then a general deny rule, and then be more specific for specific readers. This ACL structure protects user's own tags but avoids unintended attacks to other RFID readers. In terms of reader-Guardian authentication, I found that both asymmetric and symmetric key protocols could be deployed with RFID Guardians. In particular the certificate plays an important role in asymmetric key authentication, while password-based EKE symmetric key protocol is also useful.

The software regression test has drawn positive conclusions that, in time-critical environment, RFID Guardian could still react accurately and efficiently. I tested various specific tests and found that the software is robust and reliable. It achieves such effect by the “precomputation” or “preprocessing” algorithm, with which the software computes tag sets that need to be protected before any query arrives, and stores the result in memory, thereby reducing response time.

Future Work

Future research work on RFID Guardian could focus on theory analysis, such as auditing, key management problems, etc. More work is need to make the software testing framework more user-friendly and robust. One problem for example is that the path of the software application is just hard coded, so if the Perl script is moved, it will fail. The script could also check the application path first before executing tests. More importantly, new tests should be added, for example to check if the software can handle key words as names for readers or tags (e.g. a reader named “reader”). Other checks could be whether hard-to-type or invisible characters (such as “\n”

or “\t”) could crack the software. Yet more checks could be done to uncover bugs of the type revealed in the “explicit inventory”.

Bibliography

- [1] Rieback, Melanie and Crispo, Bruno and Tanenbaum, Andrew: The Evolution of RFID Security
- [2] Avoine, Gildas: Cryptography in Radio Frequency Identification and Fair Exchange Protocols
- [3] Peris-Lopez, Pedro and Hernandez-Castro, Julio Cesar and Estevez-Tapiador, Juan and Ribagorda, Arturo: RFID Systems: A Survey on Security Threats and Proposed Solutions
- [4] Jimmy Atkinson: "Contactless Credit Cards Consumer Report 2006"
- [5] Thomas S. Heydt-Benjamin, Daniel V. Bailey, Kevin Fu1, Ari Juels, and Tom O'Hare: "Vulnerabilities in First-Generation RFID-enabled Credit Cards"
- [6] Ari Juels: "RFID Security and Privacy: A Research Survey"
- [7] Jaap-Henk Hoepman, Engelbert Hubbers, Bart Jacobs, Martijn Oostdijk, Ronny Wichers Schreur: "Crossing Borders: Security and Privacy Issues of the European e-Passport"
- [8] BSI. "Advanced security mechanisms for machine readable travel documents - extended access control (eac)". Technical Report TR-03110, BSI, Bonn, Germany, 2006.
- [9] Dennis Kugler. "Security mechanisms of the biometrically enhanced (eu) passport." Presentation at the Security in Pervasive Computing conference, Boppard, Germany, April 2005.
- [10] <http://www.avidid.com>
- [11] Kern, C I-Code makes library management simple. Philips Semiconductors 'on the move' 03-2001, Austria 2001
- [12] <http://www.pcadvisor.co.uk/news/index.cfm?newsid=6195>

- [13] http://www.morerfid.com/details.php?subdetail=Report&action=details&report_id=738
- [14] Claudia Binder, Svetlana Domnitcheva: When Waste Becomes Intelligent: Assessing the Environmental Impact of Microchip Tagging
- [15] Smart Card Alliance: RFID Tags and Contactless Smart Card Technology: Comparing and Contrasting Applications and Technologies
- [16] International Standards: ISO/IEC 15693-3
- [17] <http://www.defenselink.mil/news/newsarticle.aspx?id=18127>
- [18] <http://www.defenselink.mil/news/newsarticle.aspx?id=25313>
- [19] <http://www.defenselink.mil/news/newsarticle.aspx?id=27518>
- [20] John Viega, Matt Messier and Pravir Chandra: Network Security with OpenSSL
- [21] <http://www.iona.com/support/docs/orbix2000/2.0/tls/html/OpenSSL4.html>
- [22] Richard E. Smith: Authentication: From Passwords to Public Keys
- [23] Steven M. Bellovin, Michael Merritt: Encrypted Key Exchange: Password Based Protocols Against Dictionary Attacks
- [24] Wikipedia: Needham-Schroeder
- [25] Roger M. Needham and Michael D. Schroeder: "Using encryption for authentication in large networks of computers" Communications of the ACM 21(12), December 1978.
- [26] Michael Steiner, Peter Buhler, Thomas Eirich and Michael Waidner: Secure Password-Based Cipher Suite for TLS.
- [27] RFC 2246: The TLS Protocol
- [28] RFC 2712: Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)
- [29] RFC 4279: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)
- [30] RFC 4120: The Kerberos Network Authentication Service (V5)