

SENG2250

SYSTEM AND NETWORK SECURITY

ASSIGNMENT 3 REPORT

---

# **RFID Protocol Design and Cryptographic Programming**

---

*Author*

Thomas MILLER

C3279309

Due 10/11/2018

## Contents

<b>1</b>	<b>Task 1: Secure Cloud-Based RFID Supply Chain System Design</b>	<b>2</b>
1.1	Analysing Potential Security Threats and Issues of the System . . . . .	2
1.2	Describing Architecture of Technologies That Can be Used to Provide Client Autho- risation . . . . .	3
1.3	Public-Key Based Mutual Authentication Protocol for RFID Tag Authentication . . . .	4
1.3.1	System Set-up . . . . .	4
1.3.2	Security Assumptions . . . . .	4
1.3.3	Protocol . . . . .	4
1.3.4	PKI . . . . .	4
1.4	Symmetric-Key Based Mutual Authentication Protocol . . . . .	5
1.4.1	System Set-up . . . . .	5
1.4.2	Security Assumptions . . . . .	5
1.4.3	Protocol . . . . .	5
<b>2</b>	<b>Task 2: Programming</b>	<b>6</b>

## List of Figures

1	Screen Shot of Program Execution, x3 . . . . .	7
---	--	---

# **1 Task 1: Secure Cloud-Based RFID Supply Chain System Design**

## **1.1 Analysing Potential Security Threats and Issues of the System**

- DDOS (distributed denial-of-service) Attack: Can arise from communication between the RFID tags and readers or from the readers to the server. It involves flooding the readers or server with multiple authentication requests from many different sources causing the system to overload and potentially crash depending on the system's capacity. This is made even more easier thanks to the simplicity of the authentication system as the tags can't hold more than 1kb of data meaning that to organise many authentication requests could be done easily and efficiently.
- SQL Injection Attacks: Can happen when an adversary can tamper with the contents/code of an RFID tag making it malicious to the reader and ultimately the server at which it is connected to. The code could be changed in the tag which allows the adversary to manipulate existing data in the system such as gaining access to private keys and to launch further attacks such as impersonation attacks. This is due to the wide availability and public access to the RFID tags being a physical, portable object that an adversary could get a hold of from a user then freely be able to change it at their leisure.
- Replay Attacks (Eavesdropping + Spoofing): Eavesdropping is when an adversary possesses a RFID tag reader with the capability of scanning any tag in a close vicinity without the user knowing it is being done. This process also known as skimming can be used to copy the contents of the tag for possible identity thefts. This is because the tag can be easily scanned being a lightweight portable device. This can be combined with spoofing where the adversary can overwrite tag data with their own spoof data to trick the server or disrupt any current processes. The whole process combined becomes the replay attack where the adversary can eavesdrop on a transaction between the tag and reader/server and secretly retrieve public and private keys or hash functions used to then at a later time replays them to spoof and pose as a valid tag and gain access to goods provided by the server using the information from the earlier eavesdropped session.

## **1.2 Describing Architecture of Technologies That Can be Used to Provide Client Authorisation**

Both the use of passwords and physical locking of tag memory can be used to provide extended authorisation to the system and extra protection from security attacks. By adding a password onto the RFID tag, it prevents tags from being read unless the owner has given explicit permission via the password to access any data. This can be used alongside the key pairs to provide two factor authentication for the client. This however can be attacked by the offline dictionary attack so other physical forms or locking tag memory are required such as tokens or biometrics which then enable multi-factor authentication. These physical forms are things the user has such as their own RFID reader to 'lock' their tag by sending a request to stop transmitting data, so it becomes unreadable until they scan their tag again unlocking it to be read again normally. Using biometrics which could be a simple fingerprint scanner on the tag or tag reader which give the client a unique specific unchangeable authentication system where only they have the power to access the system as no one else has an identical fingerprint. It involves the user first enrolling their fingerprint with the server who finds the matching identity, verifies the user and sends the information to unlock the tag back to the user so they can authenticate the tag and enable it to be able to send and receive data again. As well as biometrics, tokens can be used along-side them meaning the attacker needs to obtain both forms of authentication. Tokens which are in active mode not passive, have a token ID which changes periodically every 60 seconds with a new secure key for example to be used by the tag. Therefore, a possible combined system could be: The user has an app which communicates with the RFID tag. The user scans their fingerprint and enters an 8-digit alphanumeric password to unlock the app then retrieves the token from inside the app. They could then use this token for further authentication with the server or have it sync to their tag to become the new private key achieving a very secure multi-factor authentication system.

## 1.3 Public-Key Based Mutual Authentication Protocol for RFID Tag Authentication

### 1.3.1 System Set-up

- Perfect Forward Security = generating random numbers for each session
- S = Server
- T = Tag
- PuS = Server's Public Key
- PuT = Tag's Public Key
- PrS = Server's Private Key
- PrT = Tag's Private Key

### 1.3.2 Security Assumptions

- RFID tag reader does no computational services so all communication from RFID tag goes directly to server unchanged by the reader
- Both the tag and server already have access to each other's public keys
- The private keys act as identifiers for both server and tag

### 1.3.3 Protocol

1. S: Generate random number 'a' (Challenge request),  
 $S \rightarrow T: a$
2. T: Generate random number 'b',  
 $T \rightarrow S: E_{PuS}(a, b, PuT) = \text{Ticket T}$
3. S:  $D_{PrS}(\text{Ticket T})$ , search database for IDT = PuT to verify the tag,  
 $S \rightarrow T: E_{PuC}(a||b, PuS \text{ XOR } PuT) = \text{Ticket S}$
4. T:  $D_{PrT}(\text{Ticket S})$ , separate message into (1):  $a||b$ , (2):  $PuS \text{ XOR } PuT$ , Because the tag knows the length of 'a', it separates 'b' from 'a' and compares it with 'b' verifying the server (1/2) of a two-part authentication. It then XOR's (2) with the servers public key to get its own public key and compares it with its public key to verify the server again (2/2).

### 1.3.4 PKI

PKI (public key infrastructure) is a management system for public key encryption which uses CA (certificate authority) to register the keys and RA (registration authority) to manage the requests for authentication. It can be used to verify the public keys of the above protocol because it has the ability to bind the public keys of both server and tag with their respective IDs therefore confirming and verifying the ownership of said keys.

## 1.4 Symmetric-Key Based Mutual Authentication Protocol

### 1.4.1 System Set-up

- Symmetric keys = Both RFID tag and server share the same public keys
- S = Server
- T = Tag
- IDT = RFID tags identity
- $h_K(x)$  = hash function using the shared symmetric key 'K'

### 1.4.2 Security Assumptions

- RFID tag reader does no computational services so all communication from RFID tag goes directly to server unchanged by the reader
- Upon initial creation of the RFID tag its identity IDT is initially defined and set-up by the server

### 1.4.3 Protocol

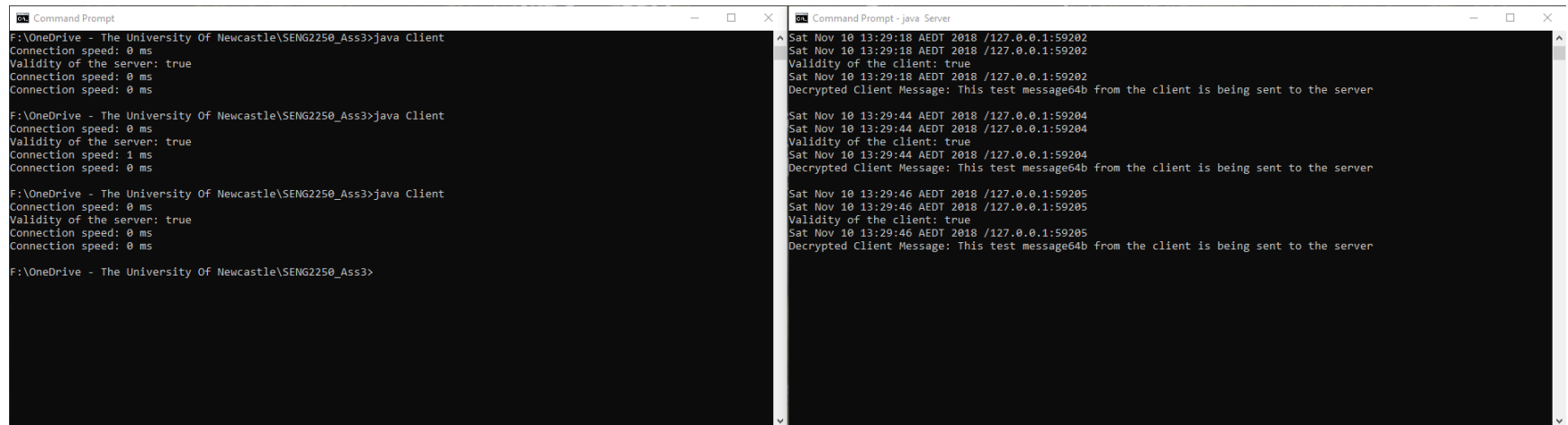
1. S: Generate random number 'a' (Challenge request),  
S  $\rightarrow$  T: a
2. T:  $h_0 = h_K(\text{IDT})$ , generate random number 'b',  $h_1 = h_K(h_0 \parallel a \parallel b)$  = private/secret key of tag  
T  $\rightarrow$  S: ( $h_1$ , b)
3. S: Searches database for IDT and upon successful location verifies the tag, then computes  $b+1$ ,  $h_2 = h_K(h_K(h_1) \parallel a \parallel b+1)$   
S  $\rightarrow$  T: ( $b+1$ ,  $h_2$ )
4. T: Computes  $h_K(h_1)$  then,  $h_K(h_K(h_1) \parallel a \parallel b+1)$  and compares it to  $h_2$  and if they are equal then the tag verifies the server and mutual authentication is achieved.  
Tag then replaces its private/secret key  $h_1$  with  $h_2$  which becomes  $h_2 = h_K(h_1 \parallel a \parallel b+1)$

## 2 Task 2: Programming

- Language: Java, (JDK required: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>)
- To Compile:
  1. Open up CMD (Command Prompt)
  2. Change directory to the folder with the source code in it
  3. Type into CMD: javac Client.java ENTER javac Server.java
  4. Open up a second CMD window
  5. In the first window type: java Server ENTER, (The server is in an infinite loop so to stop simply type CTRL+C)
  6. In the second window type: java Client ENTER, and the program should run
  7. Repeat step 6 again to run the program more than once
- Cryptographic libraries used (already implemented no need to install):
  - Crypto:

```
import javax.crypto.*;  
  
import javax.crypto.spec.SecretKeySpec;  
  
import javax.crypto.spec.IvParameterSpec;
```
  - Security:

```
import java.security.*;
```



The image shows two side-by-side Windows Command Prompt windows. The left window, titled 'Command Prompt', shows the execution of a Java client program. The right window, titled 'Command Prompt - java Server', shows the execution of a Java server program. Both windows display timestamps, IP addresses, and connection details.

```
Command Prompt
F:\OneDrive - The University Of Newcastle\SENG2250_Ass3>java Client
Connection speed: 0 ms
Validity of the server: true
Connection speed: 0 ms
Connection speed: 0 ms

F:\OneDrive - The University Of Newcastle\SENG2250_Ass3>java Client
Connection speed: 0 ms
Validity of the server: true
Connection speed: 1 ms
Connection speed: 0 ms

F:\OneDrive - The University Of Newcastle\SENG2250_Ass3>java Client
Connection speed: 0 ms
Validity of the server: true
Connection speed: 0 ms
Connection speed: 0 ms

F:\OneDrive - The University Of Newcastle\SENG2250_Ass3>

Command Prompt - java Server
Sat Nov 10 13:29:18 AEDT 2018 /127.0.0.1:59202
Sat Nov 10 13:29:18 AEDT 2018 /127.0.0.1:59202
Validity of the client: true
Sat Nov 10 13:29:18 AEDT 2018 /127.0.0.1:59202
Decrypted Client Message: This test message64b from the client is being sent to the server

Sat Nov 10 13:29:44 AEDT 2018 /127.0.0.1:59204
Sat Nov 10 13:29:44 AEDT 2018 /127.0.0.1:59204
Validity of the client: true
Sat Nov 10 13:29:44 AEDT 2018 /127.0.0.1:59204
Decrypted Client Message: This test message64b from the client is being sent to the server

Sat Nov 10 13:29:46 AEDT 2018 /127.0.0.1:59205
Sat Nov 10 13:29:46 AEDT 2018 /127.0.0.1:59205
Validity of the client: true
Sat Nov 10 13:29:46 AEDT 2018 /127.0.0.1:59205
Decrypted Client Message: This test message64b from the client is being sent to the server
```

Figure 1: Screen Shot of Program Execution, x3