Leveraging Object Linking and Embedding for Process Control Unified Architecture Standards with Smart Card Technology for Secure Applications and Services

by

Yuankui Wang

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik

Leveraging Object Linking and Embedding for Process Control Unified Architecture Standards with Smart Card Technology for Secure Applications and Services

Master's Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by
Yuankui Wang
Dessauer Str. 4
33106 Paderborn

Thesis Supervisor:
Dr. Simon Oberthür
and
Dr. Stefan Sauer

Paderborn, July 2014

Declaration

(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

City, Date

Signature

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Challenges . . . 2
- 1.3 Solution Idea . . . 2
- 1.4 Thesis Structure . . . 3

2 Fundamentals **5**
- 2.1 OPC Unified Architecture Overview . . . 5
  - 2.1.1 OPC UA Specifications . . . 5
  - 2.1.2 OPC UA Client-Server Structure . . . 6
  - 2.1.3 OPC UA Terminologies . . . 7
  - 2.1.4 Standard OPC UA Server . . . 8
  - 2.1.5 Standard OPC UA Client . . . 8
  - 2.1.6 Secure Channel and Session . . . 9
  - 2.1.7 Security Handshake . . . 10
  - 2.1.8 OPC UA Communication stack . . . 10
  - 2.1.9 Other Competitor . . . 11
- 2.2 Smart Card . . . 12
  - 2.2.1 Overview . . . 12
  - 2.2.2 Universal Integrated Circuit Card . . . 13
  - 2.2.3 Smart Card Software Components . . . 14
  - 2.2.4 Smart Card File Management . . . 14
  - 2.2.5 Data Exchange with Smart Card . . . 16
  - 2.2.6 Secure Messaging . . . 17
  - 2.2.7 Life Cycle . . . 17
  - 2.2.8 Conclusion . . . 18
- 2.3 Java Card . . . 18
  - 2.3.1 Language Specification . . . 19
  - 2.3.2 Transaction Integrity . . . 19
  - 2.3.3 Persistent Object and Transient Object . . . 20
  - 2.3.4 Java Card Applet . . . 20
  - 2.3.5 Java Card Cryptography . . . 21
  - 2.3.6 Conclusion . . . 23
- 2.4 GlobalPlatform and Remote Application/File Management . . . 23
  - 2.4.1 OPEN - GlobalPlatform Environment . . . 24
  - 2.4.2 Conclusion . . . 24

2.5 OpenMobileAPI 24
2.6 Android 25
2.6.1 Overview 25
2.6.2 Android Software Stack 26
2.6.3 Android, Dalvik and Java 28
2.6.4 Android Application Overview 29
2.6.5 Intent 29
2.7 Conclusion 30

3 State of Art **33**
3.1 Smart Home Research 33
3.1.1 Overview 33
3.1.2 Smart Home Optimized for Energy Services 34
3.1.3 Smart Health Home 35
3.1.4 Agent-based Smart Home 35
3.1.5 Conclusion 37
3.2 OPC UA Application and Security Policy 37
3.2.1 OPC UA Applications 37
3.2.2 OPC UA Secure Policy 38
3.2.3 OPC UA Secure Consideration 39
3.2.4 Conclusion 40
3.3 Smart Card Security 40
3.3.1 Smart Card in Access Control Management 40
3.3.2 Smart Card Key Management Mechanism 41
3.3.3 Threatens and Countermeasures 43
3.3.4 Smart Card Secure Applications 47
3.3.5 Conclusion 47
3.4 Secure Android Design 48
3.4.1 Android Security Mechanism 48
3.4.2 Security Design Guide-Lines 49
3.4.3 Conclusion 51
3.5 Embedded System Secure Design 52
3.5.1 Overview 52
3.5.2 Embedded System Security Requirements 52
3.5.3 Software Security during Software Design Life Cycle 53
3.5.4 Secure Architectures 53
3.6 Conclusion 54

4 Implementation Scenario **57**
4.1 Overview 57
4.2 Component Structure 58
4.3 Communication Flow 60
4.3.1 CommunicationStack Applet 61
4.3.2 OPC UA Application Functionalities 61

4.3.3 Cell Phone Component Software Structure 62
4.3.4 Housing Device Component Software Structure 64
4.3.5 Implementation Tool Support 65
4.4 Conclusion 66

5 System Design **67**
5.1 UICC Applet 67
5.1.1 Overview 67
5.1.2 Work Flow 68
5.1.3 Commands: Interface between Applet and CAD 71
5.1.4 Classes 74
5.2 Android Application 78
5.2.1 Utility Classes 79
5.2.2 Layout 80
5.2.3 Activity Classes 80
5.3 Smart Home Web Server 83
5.4 Conclusion 84

6 Implementation Test **85**
6.1 Javacard Applet and Remote Administrator Server Tests 85
6.1.1 *UTE* Test Cases 85
6.2 Android Application Tests 89
6.2.1 PIN Verification 90
6.2.2 Communication with Web Server 90
6.3 Conclusion 92

7 System Security Analysis **95**
7.1 Attack Tree Overview 95
7.2 Smart Card Security Compromise 95
7.2.1 Sub Attack Tree and its Content 96
7.2.2 Analysis and Countermeasures 97
7.3 OPC UA Application Level Security Compromise 98
7.3.1 Sub Attack Tree and its Content 98
7.3.2 Analysis and Countermeasures 99
7.4 Conclusion 100

8 Summary and Future Work **101**

Bibliography **103**

List of Figures

2.1 OPC UA Client-Server Structure [OPC12] 6
2.2 OPC UA Server Structure [OPC12] 7
2.3 OPC UA Client Structure [OPC12] 8
2.4 OPC UA Client-Server Communication [OPC09a] 9
2.5 OPC UA Client-Server Security Handshake [OPC09a] 10
2.6 OPC UA Client-Server Communication Stack [OPC09a] 11
2.7 Smart Card Software Components [Sun98] 13
2.8 The Internal Structure of a File in Smart Card File Management System [Wol08] 14
2.9 File Tree on Smart Card [Wol08] 15
2.10 Communication between Smart Card and CAD [Wol08] 16
2.11 Java Card VM Components[Sun98] 19
2.12 Javacard Applet Execution States [Wol08] 21
2.13 APDU Command Processing by JCRE [Wol08] 21
2.14 GlobalPlatform Smart Card OS Architecture [Glo11] 23
2.15 OpenMobileAPI Architecture Overview [sim14] 25
2.16 Android Stack Overview [Mar11] 27
2.17 Compiling Process Comparison [Mar11] 29
2.18 Comparison between Java and Dalvik byte code [Dav10] 30

3.1 DER Scheduler in Smart Home optimized for energy[Mic10] . . . 34
3.2 Overview of Smart Health Home structure [Vin02] (RCC: Remote Control Center) 35
3.3 MavHome agent architecture [Dia03] 36
3.4 Keys Presented in Smart Card Description Language Developed by *Morpho* 42
3.5 the PSK Key Used to Configure Administration Session 42
3.6 Differential Power Attack on a DES implementation [Pau04] . . . 45
3.7 Android Malware Growth [Bar13] 48
3.8 Detailed Android Permission Model [Han12] 49
3.9 Permission Required by Application *k9mail* [Han12] 50
3.10 Software Security Consideration in the Software Design Life Cycle [Pau04] 53
3.11 Architectural Design for a Secure Information Processing [Pau04] 54

4.1 Smart Home Scenario 57
4.2 Smart Home System Component Structure 58

4.3 Smart Home Component Communication 59
4.4 Cell Phone Component Architecture 62
4.5 Communication Flow between AP and APSD [Glo12] 63
4.6 Housing Device Component Architecture 64

5.1 Request and Response Message Exchange Demonstration 67
5.2 Handshake and Message Exchange between SD and Remote Server 70
5.3 Class Diagram 76
5.4 Android Application Class Diagram 79
5.5 Customized Text Color for the Command Result 80
5.6 Logo Image Designed for Web Applications 83
5.7 Smart Home Web Overview screen-shot 84

6.1 Screen-shot of Push Short Message including parameters which are necessary for the communication channel construction 86
6.2 Screen-shot of TLS client-hello-message sent by smart card applet 87
6.3 Screen-shot of TLS server-hello-message sent by remote server . . 88
6.4 Screen-shot of successful generation of master key between remote server and *CommunicationStack* applet 88
6.5 Screen-shot of Http message configuration in test case 88
6.6 Screen-shot of Http message generated by simulated remote administrator server 89
6.7 Screen-shot of cmd Buffer in *CommunicationStack* applet 89
6.8 Screen-shot of Http message received by remote server 90
6.9 Screen-shot of translation of Http content of figure 6.8 90
6.10 Screen-shot of Wrong PIN input 91
6.11 Screen-shot of blocked PIN 91
6.12 Using Android application remotely managing home devices . . . 91
6.13 Historical Record Overview 92

7.1 Attack Tree Overview 95
7.2 Smartcard Security Compromise 96
7.3 OPC UA Application Security Compromise 99

List of Tables

2.1 OPC UA Specifications 6
2.2 ISO/IEC 7816 [Wol08] 13
2.3 Command APDU Structure 17
2.4 Response APDU Structure 17
2.5 Package: *javacardx.crypto* 22
2.6 Package: *javacard.security* 22
2.7 Minimum Android Recommendations [Dav10] 28

3.1 Key Data Stored in Smart Card[Wol10] 42

5.1 SELECT Command APDU 71
5.2 SELECT Response APDU 72
5.3 Verify PIN Command 72
5.4 Verify PIN Response APDU 72
5.5 Reset PIN command 72
5.6 Reset PIN Response APDU 72
5.7 Communication Session Creation Command APDUs 73
5.8 Trigger Session Return Codes 73
5.9 Close Session Command APDU 73
5.10 Close Session Return Code 74
5.11 Process Data Command APDUs 74
5.12 Process Data Return Code 74
5.13 Command Data Type-Length-Value Structure 75

1 Introduction

Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the recently released industry standards set from OPC Foundation, which compared with its predecessors is equipped with a list of charming new features, with whose help OPC UA is capable of developing a common communication interface for devices manufactured by different vendors. Meanwhile, the technology of smart card is widely used in information security fields of finance, communication, personal as well as government identification and payment. Therefore it is meaningful and promising to develop OPC UA standard compliant applications on embedded smart cards secure devices, for the purpose of secure remote application control, enterprise resource planning and etc. The applied OPC UA client and server application code builds a common connectivity and communication platform for other device specific applications. Meanwhile the employed smart card not only acts as security token storing credential information but also is in charge of peer identification as well as secure messaging by applying Remote Application/File Management over Http protocols standardized by GlobalPlatform. The in this thesis implemented demonstration scenario and corresponding analysis result prove the feasibility and reliability of my proposal.

1.1 Motivation

According to the report *Mobile Economy 2013* from *Global System for Mobile Communications Association*, at the end of year 2013 there are over 3.2 billion mobile subscribers in total, which means one half the population of the earth now enjoy the social and economic convenience brought by the mobile technology. Moreover by the year 2017, 700 million new subscribers are expected to be added. The number of mobile subscriber will reach 4 billion in 2018 [A.T13]. Mobil technology opens nowadays a promising market.

Mobile products play an irreplaceable and immense role at the heart of our daily life. With the help of mobile technology, the user's world in many domains such as, education, financial transactions, health and etc. are inter-connected. Mobile users are enjoying the advantages of mobility. Services, such as 24/7 monitored home security and customized control as well as management over housing devices, exist not only in science fiction film but also could be realized using present-day technologies.

At the same time, mobility is also a critical assert in industry and business world, which can not only increase efficiency and productivity but also drive new

revenue generation and competitive advantage. The most convincing example here is the well known Machine to Machine communication technology, that is also referred as M2M technology. In a M2M communication system, machines are capable of exchanging gathered data with each other using wireless or wired network connection, together accomplishing complex tasks and making intelligent scheduling decisions. M2M technology is widely employed in different industry spheres such as factory automation, remote access control and device monitoring. It boosts the efficiency of manufacturing processes, offers centralized service support and data management, minimizes system response time.

1.2 Challenges

But in order to enjoy the aforementioned advantages, two tough issues must be resolved. First one is, how to achieve a common communication and connectivity interface for devices manufactured by various vendors. And secondly since nowadays' systems are frequently exposed in a ubiquitous network and must confront a pervasive computing environment [Pau04], system designer must also concern about how to protect the system security in different communication environments with various data complexity and customer's requirement.

1.3 Solution Idea

In this thesis, I am going to present the solution solving the above mentioned challenges and design a Smart Home system for the purpose of proposal demonstration. In this Smart Home system, home owner with the help of a smart phone is capable of experiencing services such as, 24/7 home security, inner home environment management, assigning access permissions and so on. To be more specifically, the system consists of smart phones and various housing devices, such as digital door locks, coffee maker and environment sensors. Moreover each aforementioned device is equipped with an Universal Integrated Circuit Cards (UICC smart card), which acts not only as the secure token, that keeps user's credential information, but also is in charge of peer authentication, creating secure connection and managing communication with other devices by applying remote application and file management protocol from GlobalPlatform association.

In particular, I will introduce the newly released industry automation standards Object Linking and Embedding for Processes Control Unified Architecture (OPC UA standards) to build a common communication interface for devices that are mentioned above and design an OPC UA specifications compliant Java Card applet named *CommunicationStack* on the UICC smart card. The *CommunicationStack* provides services such as secure communication channel creation, entity authentication and secure messaging.

1.4 Thesis Structure

In the second chapter I will present the fundamental technologies which will be frequently mentioned in this paper. The third chapter is going to focus on the state of art technologies in the industry world, which could offer me inspirations and show me the best practices. In the fourth chapter, namely *Implementation Scenario*, I will present the concept of my demonstration system and explain software structures as well as communication flows which are applied by my system. The following *System Design* Chapter will concentrate on the design of system components. To be more precisely, I will discuss how I designed the Java Card applet *CommunicationStack* and the Android application *Smart Home App*. In the next *Implementation Test* chapter, I will present some testing results of my demonstration system. As seventh chapter, *System Security Analysis* will describe potential threads to my system as well as corresponding countermeasures, in order to prove the reliability and security of my proposal.

2 Fundamentals

In this chapter, I am going to give a brief introduction about technologies and terminologies that are applied in this thesis. In more detail, following six topics are covered,

- OPC UA Specifications and Overview
- Smart Card Technologies
- Java Card Language and Applet
- GlobalPlatform and Remote Application/File Management protocol
- OpenMobileAPI
- Android Platform and Application

2.1 OPC Unified Architecture Overview

Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the newly released industry standards from OPC Foundation, acts nowadays as the most promising candidate in industry M2M automation world, whose major duty is to build a secure communication interface for machines that participate in the automation process.

2.1.1 OPC UA Specifications

The OPC Unified Architecture specifications are shown in table 2.1 and can be divided into three main categories:

- core specification part, which consists of specification part1 to part7.
- access type specification part, including data access, alarm and conditions, programs and historical access specifications.
- utility specification part, covering discovery and aggregates specification.

Table 2.1: OPC UA Specifications

OPC UA Part1	Overview and Concepts Specification
OPC UA Part2	Security Model Specification
OPC UA Part3	Address Space Model Specification
OPC UA Part4	Services Specification
OPC UA Part5	Information Model Specification
OPC UA Part6	Mappings Specification
OPC UA Part7	Profiles Specification
OPC UA Part8	Data Access Specification
OPC UA Part9	Alarms and Conditions Specification
OPC UA Part10	Programs Specification
OPC UA Part11	Historical Access Specification
OPC UA Part12	Discovery and Aggregates Specification

2.1.2 OPC UA Client-Server Structure

OPC UA standards apply the classic client-server architecture, where server is in charge of managing functionalities and data information provided by a machine, which runs OPC UA server implementation. Examples of sever functions are reporting and monitoring temperature data measured by a remotely allocated sensor or the make coffee function offered by a coffee maker. Meanwhile client possesses the ability to query information from server, submit subscription and send command to server.

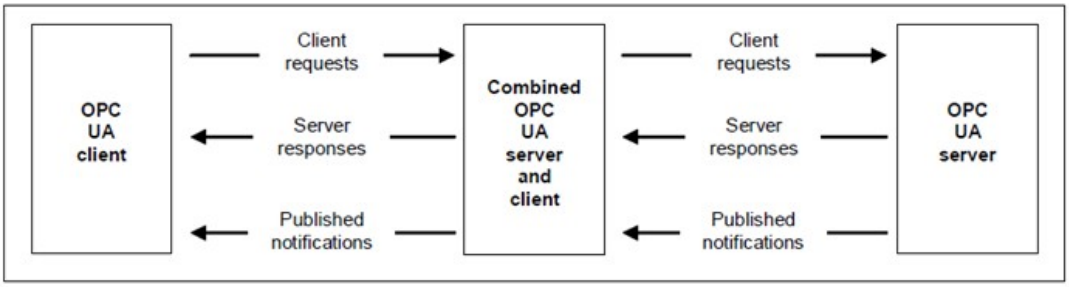

Figure 2.1: OPC UA Client-Server Structure [OPC12]

Figure 2.1 illustrates a typical OPC UA client-server architecture and also describes an internal combined server-client structure. The routine communication between client and server consists of requests from client, corresponding responses sent by server and notifications which are generated because of client's early subscriptions.

2.1.3 OPC UA Terminologies

In OPC Unified Architecture, the on server stored information, that can be browsed by clients, is defined as *address space* [OPC09b]. There exists a set of *services* [OPC09c], which are provided by server and are introduced in order to apply operations on *address space*. Information in *address space* is organized as a set of in particular hierarchy structured *Objects*. *Object* here could refer to data gathered by sensor, server configurable system parameters and etc.

OPC UA clients can query and accept information provided by OPC UA servers in two major ways, *binary structured data* and *XML documents*, depending on the complexity of exchanged data, network quality and so on. In addition three kinds of transport protocol are already defined to support the client-server communication. They are: *OPC UA TCP*, *Http/SOAP* and *Http*.

One of the charming features provided by OPC UA is *Event Notifications*. With the help of *Event Notification*, OPC UA servers are allowed immediately after satisfaction of conditions, which are normally predefined and customized by a client, to publish data to that client. In this way, clients can for instance discovery failures within client-server-communication quickly and recover that communication as soon as possible, which in return minimizes the lost to the smallest possible amount. Also clients are able to observe the subscribed data more precisely and find the pink elephant as fast as possible.

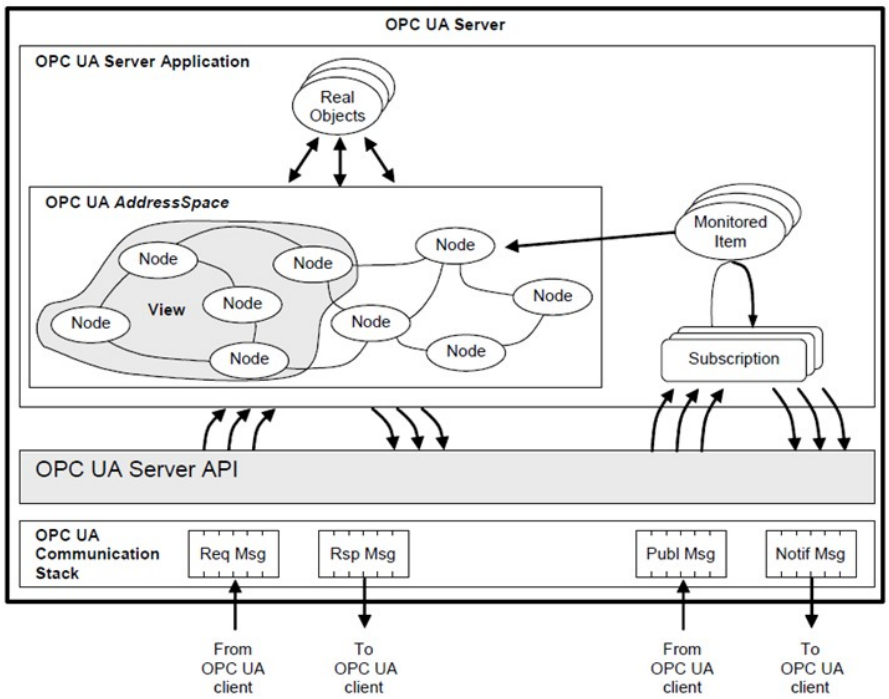

Figure 2.2: OPC UA Server Structure [OPC12]

2.1.4 Standard OPC UA Server

In figure 2.2, the structure of one standard OPC UA server is described. It includes an OPC UA server application, an internal API and a communication stack. The server application code realizes functionalities and services defined in OPC UA specifications, such as *Event Notification*, processing request from connected OPC UA client, data encryption and decryption. Moreover, *Real object* here refers to the device hardware and device specific software application that is maintained and managed by OPC UA server application code. *Node* in the *Address Space* presents abstractly the in previous paragraph mentioned *Real Ojbect*. *View*, which is pictured as a part of address space, presents *Nodes* that can be browsed by a particular client. The main task for communication stack is to establish a secure communication channel between OPC client and server. Upon on this secure channel, OPC client and server are able to construct an application session for messaging and command processing. Messages, which are frequently exchanged, include client request, server response, client subscription and server notification. The internal API connects the server application and the communication stack.

2.1.5 Standard OPC UA Client

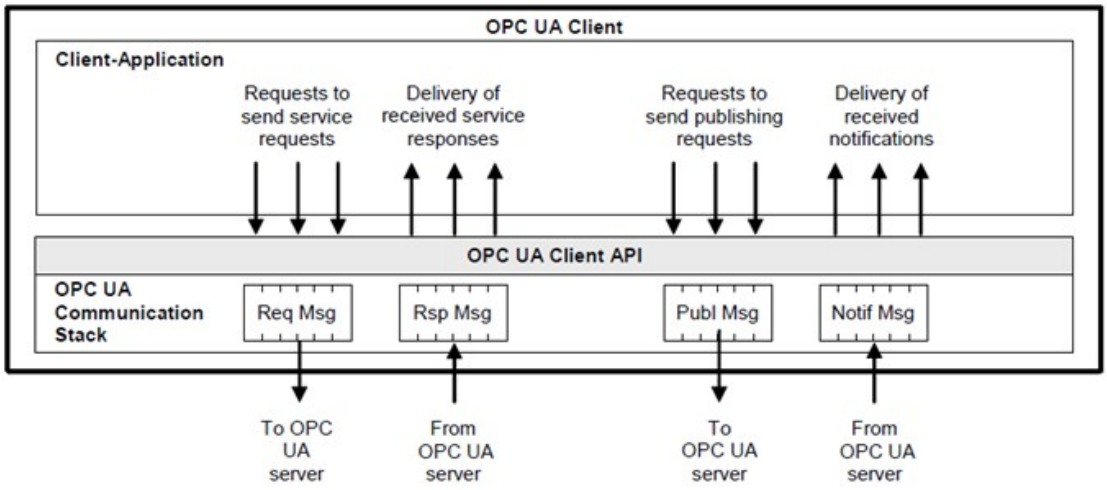

Figure 2.3: OPC UA Client Structure [OPC12]

Figure 2.3 pictures one simple OPC UA client containing client application, an internal API isolating the application code from communication stack, and a communication stack that converts API calls into messages and delivers them to OPC UA server.

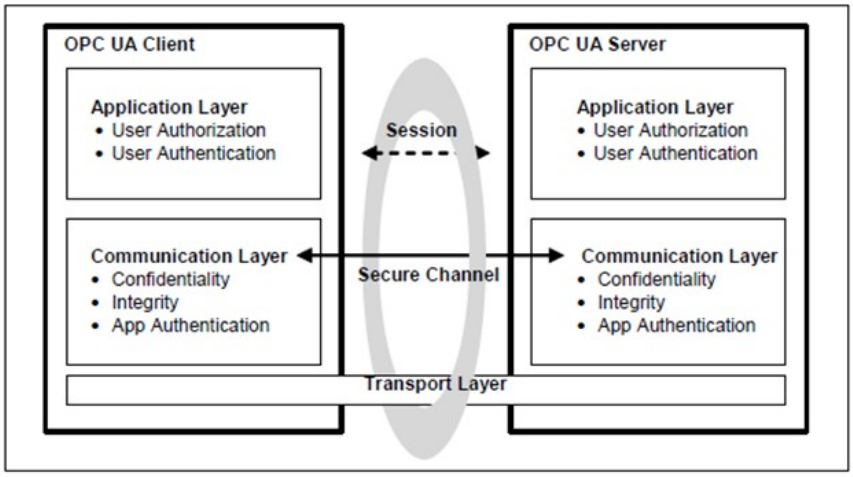

Figure 2.4: OPC UA Client-Server Communication [OPC09a]

2.1.6 Secure Channel and Session

Since data exchanged between client and server could be extremely precious and should be protected from other malicious third party, OPC UA defines a set of *security models*, with whose help the system developer is able to configure the application security level to meet the need of reality. In the *security model*, peer authentication as well as authorization, data integrity and confidentiality, system auditability (also known as traceability) and availability of services are protected. Also OPC UA standard provides a set of countermeasures against attacks such as message flooding, eavesdropping, server profiling and session hijacking [OPC09a].

Figure 2.4 pictures the typical communication architecture between OPC UA client and server. As shown in 2.4, the communication between OPC UA client and server is established above a secure channel, which is active during the whole application session and in this session, the state information, such as algorithms used for authentication, user credentials, is maintained. The secure channel is established only after a successful validation of both client and server identities and it provides necessary mechanisms to support content confidentiality, message integrity and application authentication. On the top of secure channel, an application level session between OPC UA client and server is created, whose responsibility is to transmit data and command information. It should be pointed out that, even a secure channel is out of work for some reasons, the session is still valid. Consequently, OPC UA client and server involved in the aforementioned session are able to re-establish the broken secure channel. A secured transport layer is guaranteed by encryption and signatures methods provided by the platform that supports and applies OPC UA structure.

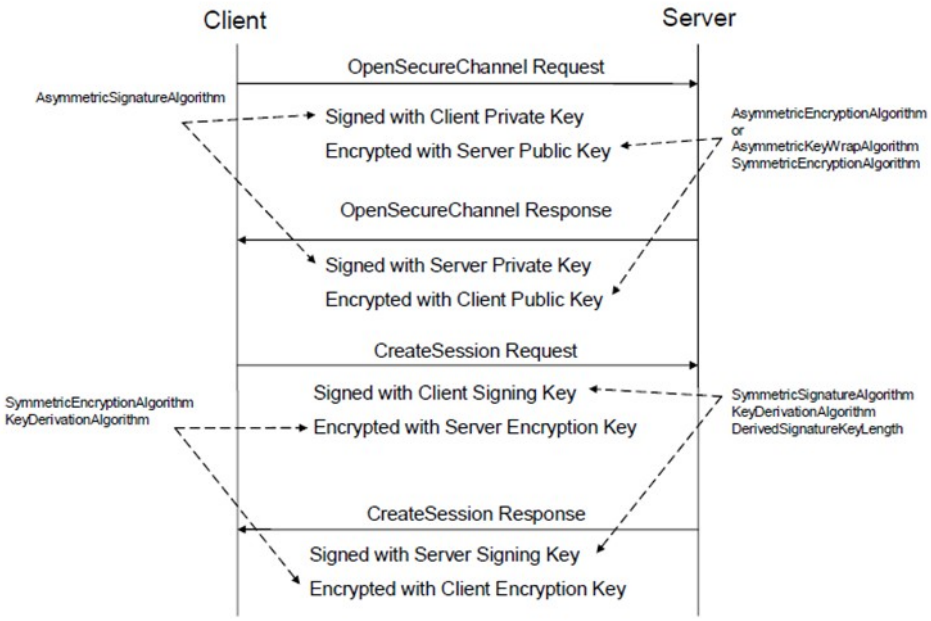

Figure 2.5: OPC UA Client-Server Security Handshake [OPC09a]

2.1.7 Security Handshake

Security handshake as illustrated in figure 2.5 explains with some details how secure channel and session are established. Normally, OPC UA client initiates the first *OpenSecureChannel* request and waits for the response from server. Messages exchanged during the secure channel construction phase are encrypted using asymmetric encryption and signature algorithms. Alternatively *AsymmetricKeyWrapAlgorithm* could also be applied to encrypt symmetric keys. In this way, OPC UA client and server are able to use symmetric encryption algorithm with the encrypted keys to secure the messages. After a successful construction of secure channel, OPC UA client sends the *CreateSession* request and waits for the server response. Messages transported during this procedure are encrypted with symmetric encryption algorithms and signed with signature keys.

2.1.8 OPC UA Communication stack

As described in figure 2.6, based on different responsibilities, OPC UA communication stack can be divided into three layer: serialization layer, secure channel layer and transport layer. Figure 2.6 also pictures a precise functionality overview of each layer. To be more precisely, from top to bottom,

- Serialization Layer and Secure Channel Layer. Serialization layer together with secure channel layer build the communication layer of an OPC UA application and their jobs are, dividing long message into pieces referred as

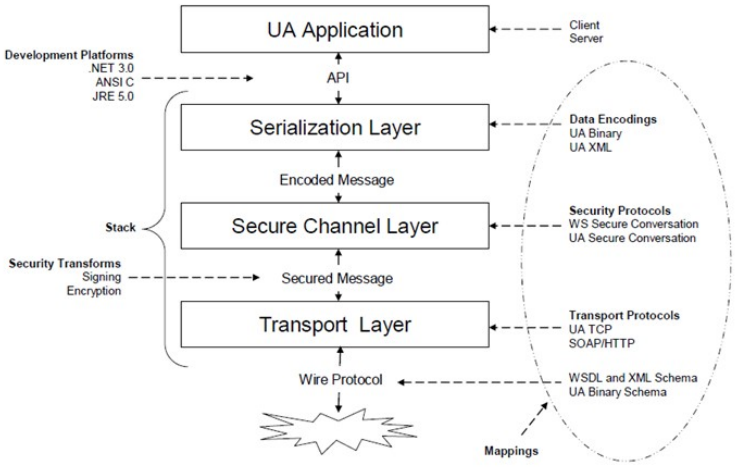

Figure 2.6: OPC UA Client-Server Communication Stack [OPC09a]

message chunks, encrypting each individual message chunk using cipher suit which is negotiated between him and the message receiver, and forwarding the encrypted message chunk to the transport layer in a sequence. Upon receiving message chunks from others, the message receiver will firstly verify whether this message piece meets the security standard negotiated between him and the message sender. If not, message receiver will close the secure channel. After a successful verification of all message chunks, the original OPC UA message will be reconstructed and forwarded to the UA Application Code through an API.

- Transport Layer. The transport layer provides mechanism for message transmitting.

2.1.9 Other Competitor

In this paragraph, I will shortly discuss the reason why I prefer OPC UA standards over other IoT protocols for my proposal.

Message Queuing Telemetry Transport (MQTT) [Den] is another machine to machine (M2M) Internet of Things communication and connectivity protocol. Compared with OPC UA standards, MQTT also supports UDP protocol in the transport layer. In OPC UA, only unidirectional, client to server, communication is provided. But according to MQTT standard, server to client communication is also possible without server implements client code. Moreover the communication overhead of MQTT in comparison with OPC UA is relative small.

Even though MQTT protocol supports communication environment with low bandwidth and high latency, OPC UA provides complexer object modeling rules

and supports more features including historical data record, alarm, notification as well as sophisticated security policies. Therefor OPC UA is more suitable for the application system that handles sensitive data with complex structure and needs immediate response.

Another member from Internet of Things is Constrained Application Protocol (CoAP) [Wik] which is designed for the extreme simple electronic device with less memory and computing power. The original CoAP only runs over UDP. Compared with OPC UA, simplicity from CoAP is its advantage. But apparently it should be considered that in most industrial products other transport protocols like TCP are frequently preferred. Moreover complexer functionalities and services other than pure message exchange between client and server are desired. This is the reason why I propose to apply OPC UA specification for complex industrial system.

2.2 Smart Card

For cyber devices, secure information storage and the security of communication play alway a signification role. In my proposal, I suggest to employ smart card in order to add additional protection to my system and let this magic bullet handle the device communication management as well as storage of credential data. In the following paragraphs, I am going to present the characteristics of smart card and discuss its security features.

2.2.1 Overview

Smart Card, whose characteristic feature is an integrated circuit that is embedded in a chip card, which is capable of performing data processing, information storage and message transmitting [Wol08]. The most charming feature of smart card is that, sensitive user's credential data such as certificates, encryption keys, digital signatures along with other precious information can only be accessed though a serial interface, which stands under the strict control of the card operation system. This characteristic provides a strong protection against unauthorized data access as well as illegal data modification and ensures the confidentiality of on card stored information. Therefor smart cards are widely applied in applications that require strong secure protection.

With sophisticated communication protocol using Application Protocol Data Units (APDU for short), smart card and Card Accepting Device (CAD) are able to perform secure message exchange. Smart card is also able to process cryptographic algorithms on its hardware. Nowadays, it supports symmetric key algorithms like DES, triple DES; standard public key cryptography for instance RSA, hash functions such as commonly applied SHA-1 [Wol08]. More powerful microprocessor on chip card is, the computation speed performance is better.

Table 2.2: ISO/IEC 7816 [Wol08]

ISO7816 document	Description
ISO 7816-1	Physical characteristics
ISO 7816-2	Dimensions and location of the contacts
ISO 7816-3	Electronic signals and transmission protocols
ISO 7816-4	Industry commands for interchange
ISO 7816-5	Number system and registration procedure
ISO 7816-6	Interindustry data elements

In ISO/IEC 7816 standards family as shown in table 2.2, the smart card's fundamental properties and functionalities are defined.

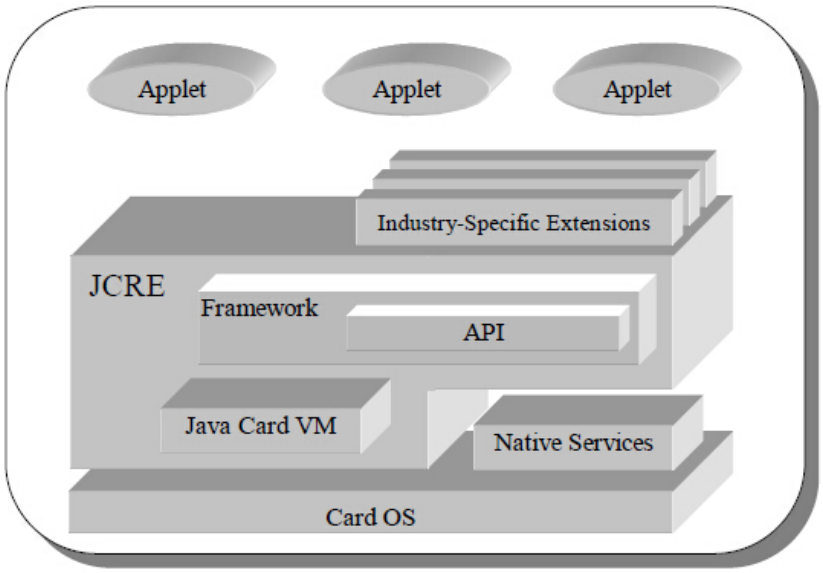

Figure 2.7: Smart Card Software Components [Sun98]

2.2.2 Universal Integrated Circuit Card

The Universal Integrated Circuit Card is the smart card used in mobile terminals in GSM and UMTS networks. It enables authenticated subscriber to join the network with their mobile terminals and at the same time protects essential user data. UICC acts also most time as the secure token, that stores and protects subscriber's confidential information. Moreover, as a 32 bit processor, UICC is also capable of processing necessary encryption and decryption algorithms [Jea09].

2.2.3 Smart Card Software Components

As illustrated in figure 2.7, one typical smart card includes [Sun98],

- Card Operation System.
- Native services such as I/O operation and memory management.
- Java Card Runtime Environment (JCRE), that consists of Java card Virtual Machine and the Framework, who is in charge of dispatching APDU as well as applet management.
- Installed applets.
- Other optional industry specific extensions.

2.2.4 Smart Card File Management

As a data storage media, in contrast with other traditional storage device, the most distinguishing feature provided by smart card file system is that, there exists none man-machine interface [Wol08], which means all data on smart card is addressed with help of hexadecimal codes and each single file processing command is strictly based on this schema.

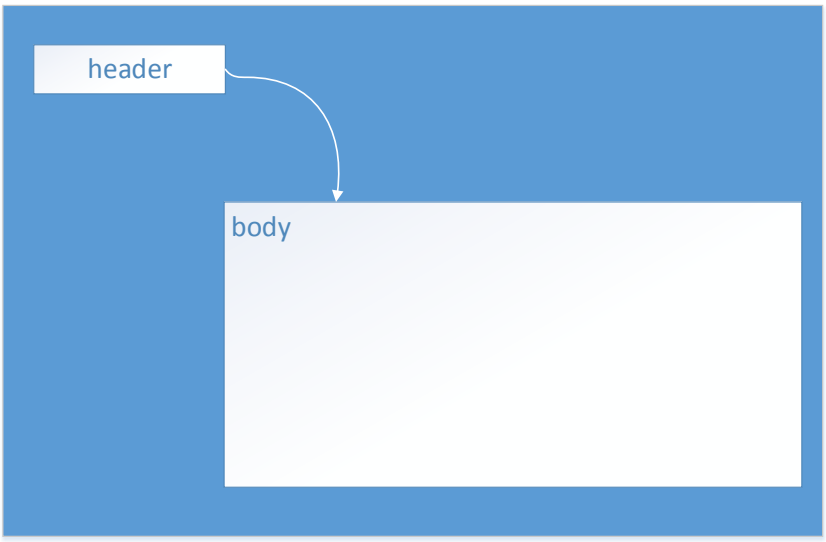

Figure 2.8: The Internal Structure of a File in Smart Card File Management System [Wol08]

File Structure

As pictured in figure 2.8, the file stored on smart card consists of two parts. The file header, which encapsulates administrative information such as, file structure and access conditions. The body, that stores real user data and is linked with the file header using a pointer. This file management mechanism has its own advantage. To be more specifically, since file header and body are separately located, therefore even when write/read error occurs in the file body, the file header, that is under normal circumstances never altered and saves essential access conditions, will not be affected, which in return provides better physical storage security.

File Types

According to ISO/IEC 7816-4 specification, smart card offers two major file types, dedicated file (DF) and elementary file (EF). DF is also described as directory file, which contains lower-level DFs and EFs. And in EF, real user data is stored. Moreover there is a special DF, called master file (MF), which represents root directory of smart card file system and only can be selected by smart card OS. Figure 2.9 illustrates one possible architecture of smart card file system.

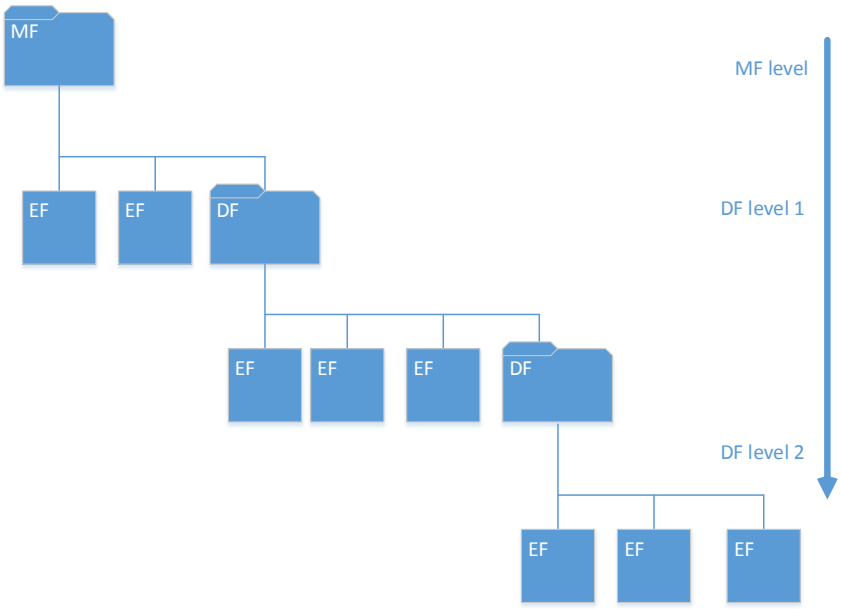

Figure 2.9: File Tree on Smart Card [Wol08]

PKCS#15

As already discussed, smart card is not only used as a storage media but can also act as a cryptographic token which offers enhanced security and privacy functionalities for other applications. The PKCS#15 (Public key cryptography standards)

specification [RSA99], which was proposed by RSA Inc. and is nowadays worldwide accepted, provides standards, that define how to store credential information such as cryptographic keys, certificates on smart card, and how to retrieve specified token with the help of a PKCS#15 interpreter.

2.2.5 Data Exchange with Smart Card

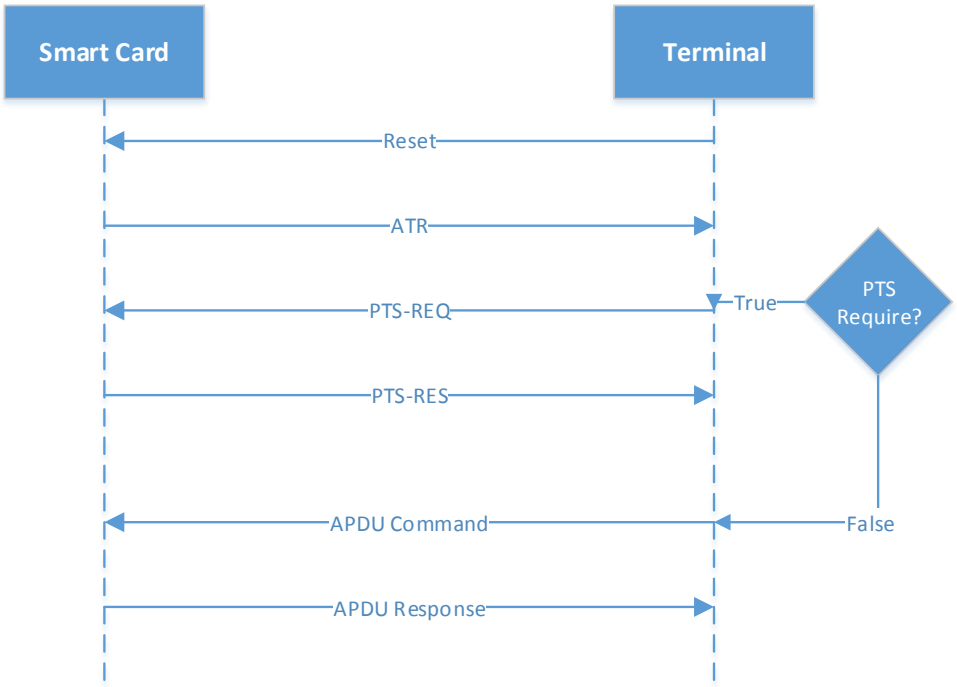

Figure 2.10: Communication between Smart Card and CAD [Wol08]

The communication protocol between smart card and terminal is described as Master-Slave relationship [Wol08], that means the terminal device knowns as Master processes unidirectional control over its slave, namely the smart card. Once a Master-Slave relationship is established, each communication will be initialized by Master and Slave only reacts based on Master's command.

When a smart card is inserted into a terminal, Master will send its Slave a RESET command which causes the smart card to perform a Power-on-Reset behavior. After this Power-on-Reset, smart card informs its Master an Answer-to-Rest (ATR) message, which describes the card state and communication parameters. In the next step, if necessary terminal will generate a Protocol Type Select (PTS) command, which is used to choose a communication protocol and associating parameters suggested by smart card. After a successful negotiation, smart card and terminal are able to exchange data using Application Protocol Data Units.

Application Protocol Data Unit, or APDU for short, is used to perform data exchange between a smart card and a Card Accepting Device. Its structure abides

ISO 7816-4 specification [Zhi00]. There are two categories of APDU, namely command APDU and response APDU. Command APDU structure and response APDU structure are described in Table 2.3 and Table 2.4 respectively [Wol08].

Table 2.3: Command APDU Structure

CLA	class byte identifying application	mandatory
INS	instruction byte representing the actual command	mandatory
P1	parameter 1 providing more command information	mandatory
P2	parameter 2 providing more command information	mandatory
Lc field	length of the data received by card	mandatory
data field	data sent to card	optional
Le field	length of the expected response data	optional

Table 2.4: Response APDU Structure

data field	length decided by Le of preceding command APDU	mandatory
SW1	status word 1 also called return code 1	mandatory
SW2	status word 2 also called return code 2	mandatory

2.2.6 Secure Messaging

Since all communication between smart card and terminal is based on digital electrical pulse performed on card I/O line, adversary can easily record all communication information and recover it. Therefore secure messaging mechanism is proposed and used to protect against aforementioned message eavesdropping, to ensure authenticity and confidentiality of exchanged information.

In the secure messaging mechanism, both message sender and receiver must negotiate the to be applied message cryptographic algorithms and possess corresponding pre-shared keys. In the realm of telecommunication, in accordance with ISO/IEC 7816-4 [Wol08], TLV (Type-Length-Value) format data is applied to perform secure messaging as the data carrier.

2.2.7 Life Cycle

The typical life cycle of a smart card is described as following [Nim]:

- *Fabrication Phase.* As the start of smart card life cycle, this process takes place under the strict control of smart card manufacturers. One unique *fabrication key*, *KF* for short, which is derived from a *manufacturer key*, is applied in order to protect smart card from unauthorized modification.

- *Pre-personalization Phase.* After fabrication phase, smart card chip is gonging to be integrated on a plastic frame by card suppliers. Moreover *fabrication key* will be replaced by a *personalization key*, which is also known as *KP*, for the purpose of secure smart card delivery to card issuer. In order to protect smart card from malicious adversaries, *KP* is locked by a *personalization lock*.

- *Personalization Phase.* In this phase, card issuer will write complete data files on the card including card PIN, card holder identity and so on. When the card issuer finishes this phase, he will add an *utilization lock* and the life cycle of smart card will come to next *utilization phase*.

- *Utilization Phase.* Card holder will be able to use his smart card in this phase.

- *Invalidation Phase.* Smart card operation system will block the smart card in this phase. To be more specifically, operations such as data writing and update will be disabled. There are two reasons that why a smart card is locked by the card system. Reason one is that both *smart card PIN* and *smart card unblock PIN* reach their limited try counts, which indicates someone is trying to force brute guess PIN. The other reason is that an application adds a *invalidation lock* on the *master file* of the disabled card.

2.2.8 Conclusion

From the above described smart card security characteristics and life cycle, it can be concluded that, smart card manufacturers and issuers have managed to develop a secure card system protected by various robust security mechanisms during the card design, production and delivery phases.

Nowadays, smart card with its outstanding tamper resistant feature is widely employed as the secure token in various industrial fields. Moreover, by adapting standardized industrial system frameworks, which will be introduced in section 2.4, smart card is also capable of supporting remote peer authentication and remote application as well as file management.

2.3 Java Card

Java Card is widely applied to write secure java-based applications on smart cards. Java Card technology not only inherits the distinguishing features from Java, such as productivity, security and portability [Sun98], but also makes Java technology available on smart card, where programmer must be faced with more harsh conditions including limited memory resource and computing capacity.

Moreover, in contrast with Java VM, Java Card VM consists of two parts, the on-card part, which is in charge of bytecode execution, class as well as object

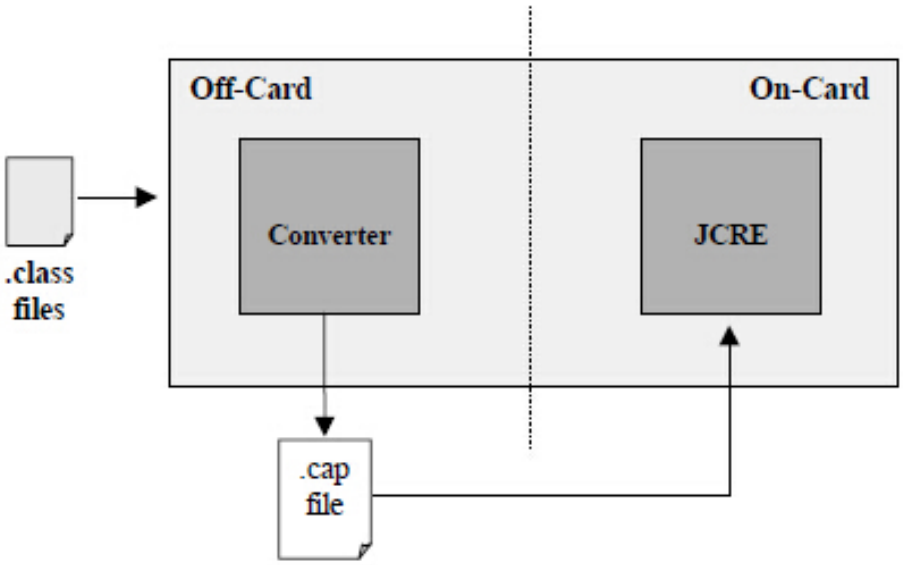

Figure 2.11: Java Card VM Components[Sun98]

management and secure data exchange. Secondly the off-card part, which is the real Java Converter.

As illustrated in figure 2.11, a complied Java Card applet (.cap file) is generated by off-card Converter based on the input Java Card class file and executed by the on-card Java Card Runtime Environment (JCRE).

2.3.1 Language Specification

Apart from the above mentioned Java Card VM, compared with the original Java language, Java Card has many unique features. For instance, since current smart card OS does not support multitasking, therefore threads are not backed up in Java Card. Moreover garbage collection is automatically performed by Java Card VM, as a result, function *finalize()* is not provided. Since the memory space and processing ability provided by smart card are limited, programmer can only use three main primitive types, namely *byte*, *short* and *boolean*. Furthermore only one-dimensional array is offered. Nonetheless Java Card language supports all features of inheritance and provides all Java language security mechanisms, such as private access modifiers and byte-code verification mechanism [Sun98].

2.3.2 Transaction Integrity

One of the most import features of Java Card technology is that Java Card Runtime Environment ensures the integrity of transaction, which means even an unexpected loss of power occurs on smart card, the integrities of ongoing transactions are protected with the help of following schema [Wol08]:

```
// Transaction Starts
JCSystem.beginTranscation();

doSomething

//Transaction Ends
JCSystem.commitTransaction();
```

Only when JCRE finishes running method *JCSystem.commitTranscation()*, the corresponding transaction will be finished and submitted. Otherwise JCRE will throw transaction exception and reset data involved in this broken transaction.

2.3.3 Persistent Object and Transient Object

In the realm of Java Card, objects are preserved in smart card's nonvolatile memory, which means this persistent object exists beyond the execution time of corresponding applet, as long as there exists a reference pointing to it. But also it is allowed to develop transient object using Java Card. To be more precisely, for instance class *array* object stays in the card nonvolatile memory space but in contrast one instance of *array* is stored in volatile memory [Wol08].

2.3.4 Java Card Applet

Java Card applet refers to the Java Card language programmed code, which extends the class *Applet* from package *javacard.framework*. Meanwhile under normal circumstance Java Card applet implements following four methods:

- *install()*, this method must be implemented and is used to create an applet instance.
- *process()*, the implementation of this method is also mandatory and uses APDU as the input parameter. In this method, the applet developer designs how to process APDU sent to this applet and which type of response APDU should be generated.
- *select()*, when JCRE detects a SELECT APDU command, which is applied to select one installed applet, JCRE will call this method of the to be selected applet.
- *deselect()*, this method is called by JCRE to inform corresponding applet that it is no longer selected by JCRE.

After finishing programming one applet, it comes to the next phase, namely applet installation. This process takes place usually in the card issuer's factory or office under his strict control. When one applet is installed on smart card, it only directly communicates with JCRE and other installed applet classes. It should

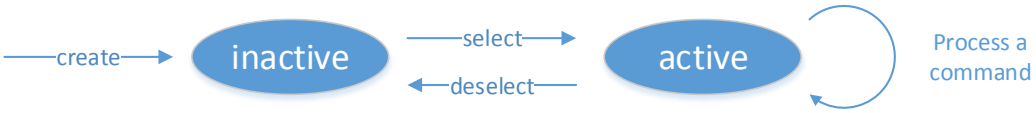

Figure 2.12: Javacard Applet Execution States [Wol08]

be pointed out, this installed Java Card applet is also belongs to *persistent object* and stored in smart card's nonvolatile memory space. Every Java Card applet is assigned one unique Application ID, which is also known as AID and used by JCRE to *register* and *select* corresponding applet at run time.

Figure 2.12 pictures execution states transaction of Java Card applet. Furthermore figure 2.13 illustrates how JCRE selects and deselects one applet based on the input APDU command.

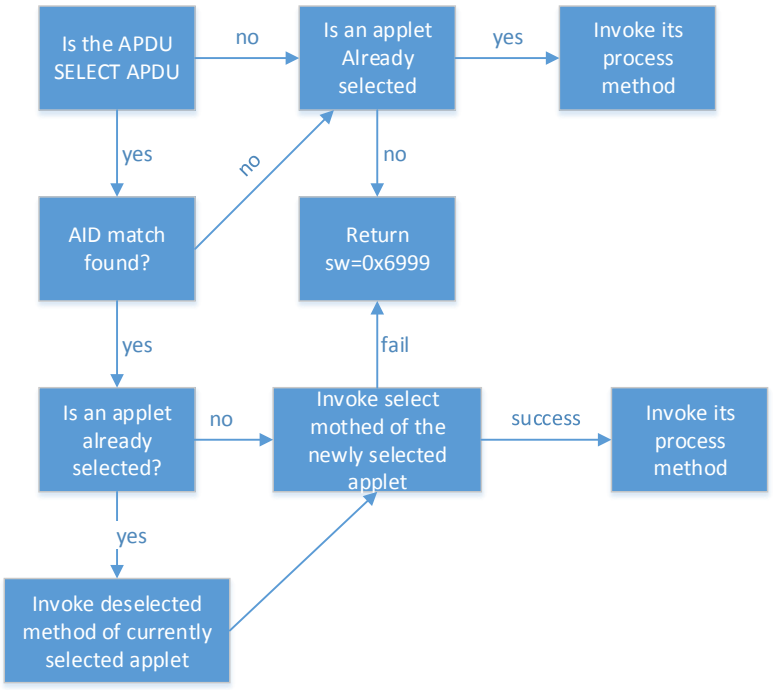

Figure 2.13: APDU Command Processing by JCRE [Wol08]

2.3.5 Java Card Cryptography

Java Card provides a list of APIs to protect smart card applet security and to reinforce the system security of other high level applications by acting as essential

security token. In order to ensure secure messaging between smart card and terminal, following three aspects must be considered:

- *Entity Authentication:* Usually mutual authentication is applied to exam the identities of both communication partners.
- *Message Confidentiality:* The transfered information is encrypted using encryption algorithms negotiated between two communication entities to ensure data privacy and security.
- *Message Integrity:* In order to protect exchanged message from unauthorized modification and to provide authenticity assurance, message authentication code (MAC) is calculated and integrated in that message.

Packages *javacard.security* and *javacardx.crypto* support aforementioned security mechanisms with following classes and interfaces [Zhi00]:

Table 2.5: Package: *javacardx.crypto*

Class or Interface	Function Description
Cipher	Abstract class offers cryptographic cipher used for encryption and decryption
KeyEncryption	Class provides implementation of keys

Table 2.6: Package: *javacard.security*

Class or Interface	Function Description
Key	Interface for all keys
SecrectKey	Interface for symmetric algorithms' keys
DESKey	Interface for keys used for DES or two-key triple DES or three-key triple DES
PrivateKey	Interface for private keys
PublicKey	Interface for public keys
RSAPrivateKey	Interface for keys used by RSA algorithm to sign data
RSAPublicKey	Interface for keys used to verify signatures generated with RSA
DSAKey	Interface for keys used by DSA
DSAPrivateKey	Interface for keys to sign data with DSA algorithm
DSAPublicKey	Interface for keys to verify signatures generated with DSA
KeyBuilder	Factory class implemented to construct key objects
MessageDigest	Abstract class for hashing algorithm
Signature	Abstract class for signature algorithm
RandomData	Abstract class for generation of random data
CrptoException	Exception class

2.3.6 Conclusion

In a short conclusion, Java Card is a for embedded secure elements such as smart card specifically designed programming language, which emphasizes the concept of portability and security. A great number of Java Card applications are compliant with GlobalPlatform specifications, which are going to be introduced in following paragraphs.

2.4 GlobalPlatform and Remote Application/File Management

GlobalPlatform is an international non-profit organization that provides standardized specifications for multiple smart card applications. The GlobalPlatform's protocols are now widely accepted and applied as industry standards for managing Java Card applet based applications in many industrial domains, for instance in telecommunication industry and payment company [Glo11].

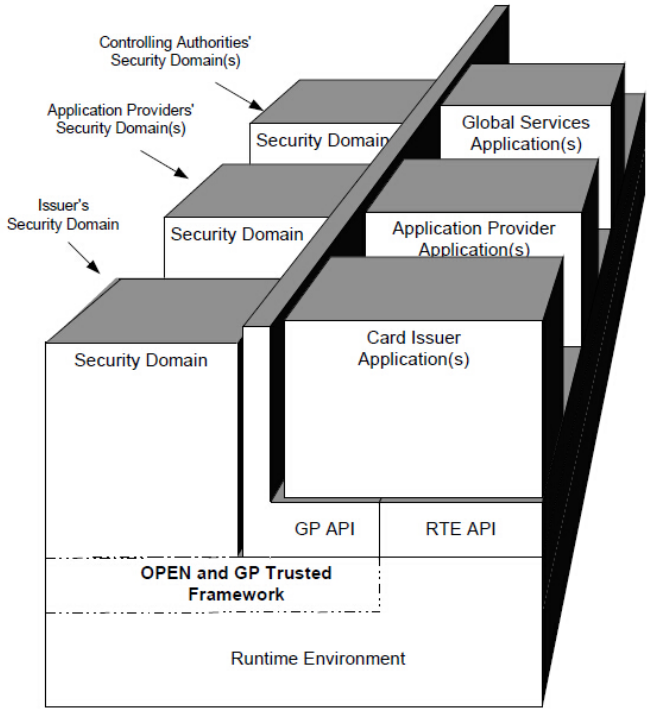

Figure 2.14: GlobalPlatform Smart Card OS Architecture [Glo11]

As shown in figure 2.14, the GlobalPlatform proposed smart card operation system contains four essential components. The runtime environment, that provides hardware-neutral API for card application and manages card memory spaces. The on card installed applications, which offer customers various functionalities and

services. The security domain (SD), that is usually associated with particular application and known as on-card representative of off-card authority. SD is in charge of message encryption as well as decryption, creation and validation of digital signature and handling keys used in cryptographic processes. The last component is OPEN framework [Glo11].

2.4.1 OPEN - GlobalPlatform Environment

OPEN provides various sets of APIs offering functionalities such as entity authentication, remote data exchange, secure channel configuration and remote application management. This framework also performs APDU dispatching as well as application selection and logical channel management [Glo11]. Logical channel is designed to enable the data exchange between multi applications and one terminal. Each opened logical channel will handle messages regard of one application. Moreover, a special logic channel named basic channel is always opened.

In order to ensure system security, OPEN supports secure mechanisms such as, user authentication and secure channel protocol.

Secure Channel Protocol In particular, GlobalPlatform has designed the secure mechanism: secure channel protocol, to guarantee a secure remote communication between smart card and authenticated remote entity. It ensures confidentiality of exchanged information and offers data integrity check. Secure Channel Protocol also introduces a cryptographic exchange process to enable smart card and off-card entities to perform mutual peer authentication.

2.4.2 Conclusion

The emphasized secure channel protocol, to be more specifically the remote application and file management standards, will be employed in my proposal, in order to enable smart card and remote administrator server to perform secure messaging and peer identification.

2.5 OpenMobileAPI

OpenMobileAPI, provided by SIMalliance, constructs an interface between terminal and chip card, which can be used by the on terminal installed mobile application to access recourse stored on smart card as well as to call functions provided by smart card applets.

Figure 2.15 presents an architecture overview of OpenMobileAPI, which consist of three functional layers:

- *Transport Layer:* This layer is in charge of constructing secure connection between application and secure element.

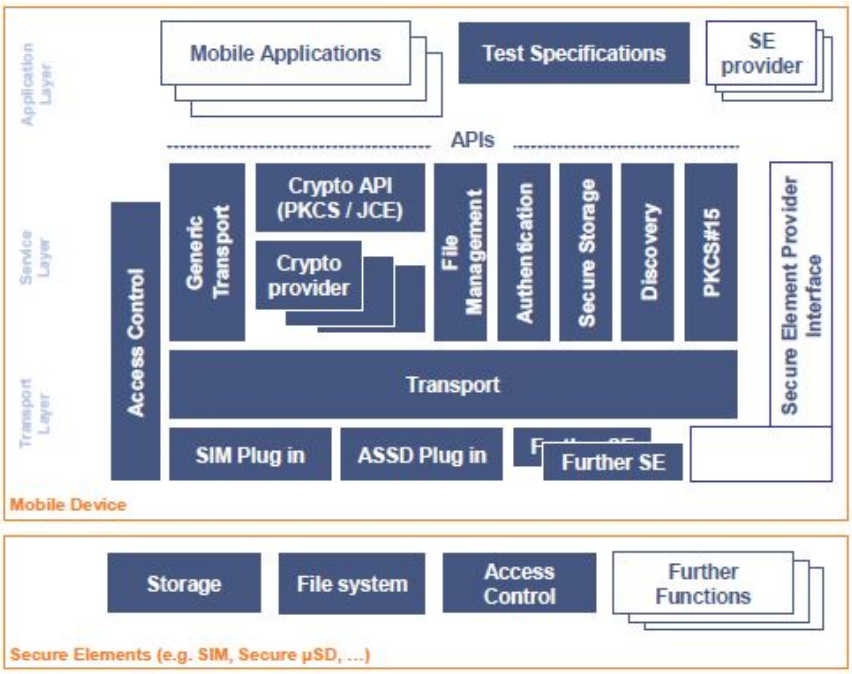

Figure 2.15: OpenMobileAPI Architecture Overview [sim14]

- *Service Layer:* Abstract interfaces, that provide various functions such as secure storage and cryptographic services, are offered by this layer.
- *Application Layer:* Mobile applications which benefit from OpenMobileAPI stay in this layer.

In my demonstration system, I will apply the OpenMobileAPI to perform the secure data exchange between my Java Card applet named *CommunicationStack* and the simulated OPC UA client which is coded as a Android application, as presented in section 5.2.3.

2.6 Android

2.6.1 Overview

Android is an open source platform, that is based on Linux, modified by Google, designed for mobile devices. As a comprehensive platform, Android manages to create a separation between hardware and software that runs on it. At the same time, being an open source project means its entire stack is open. Android developers are able to deploy their Android systems on any specific hardwares as well as to learn the system to the fundamental level [Mar11]. Moreover, Android system provides a list of attracting software and hardware features to its customers, such as:

- *Security:* Linux, as the cornerstone of Android, has proved to be a secure system through many harsh tests over the years, which in return guarantees the Android system security [Mar11].
- *Multi-Media:* A wide range of media formats are supported by Android. For instance: MPEG4, ACC, PNG, GIF and so on [Fra11]. As a result, Android customers are able to enjoy a comfortable user experience and entertainment functionalities.
- *Designed for Online:* One outstanding core feature of Android system is the ability to stay Online [And11]. Under various conditions such as Wi-Fi network, GSM/CDMA and so on, Android device always provides its end user qualified network connection.
- *Variety of applications :*As one of the most popular platform, Android, with the help of Android Market and a great number of talent Android application developers, offers its customers the possibility to extend functionalities of their Android devices [And11]. Android user is capable of downloading and installing various innovative applications from Android Market and enhancing the user experience.

In the following four paragraphs I am going to briefly introduce the Android software stack, Dalvik and an overview of Android application.

2.6.2 Android Software Stack

Android stack is composed of four different layers as shown in figure 2.16.

Linux Kernel Layer:

This fundamental layer separates other three layers from device hardware and provides core functions such as power and hardware driver management.

Libraries Layer:

As the second layer of Android stack, libraries layer supports Android runtime environment by applying various C/C++ core libs. Also in this layer, the for Android specifically designed virtual machine, namely Dalvik [Mar11] is deployed. There exist two reasons for not using standard Java VM [Mar11]. First of all, since Java VM is a generally developed virtual machine, therefore the constrains from mobile systems and devices are not concerned, such as *processing gap* and *battery gap* [Pau04]. Processing and battery gap refer to the limited process capabilities and battery lifetime of mobile device. Secondly, when the Android project was being carried out, Java VM didn't belong to the open source projects. For these reasons Dan Bornstein and his group developed this license-free and mobile platform specific virtual machine, Dalvik.

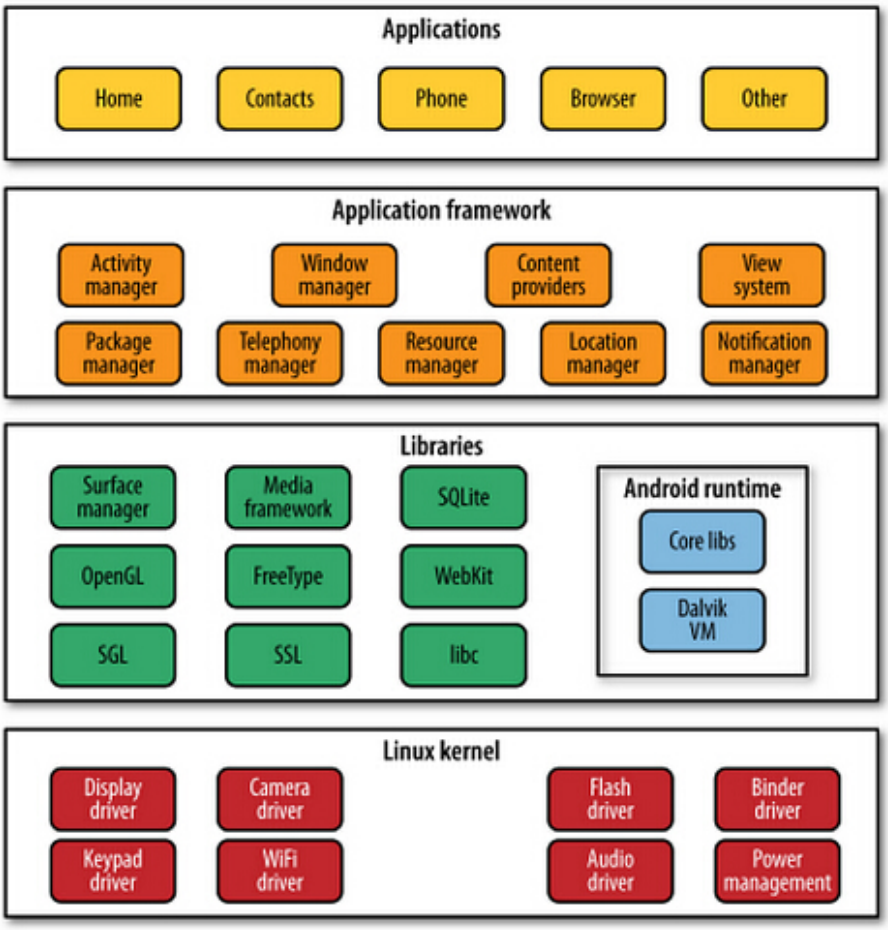

Figure 2.16: Android Stack Overview [Mar11]

Moreover, other libraries which provide services to application framework layer are also included in this libraries layer, they are for instance:

- OpenGL, library that supports 2D and 3D graphics rendering.
- SSL, the widely applied secure socket layer library.
- SQLite, which provides lightweight SQL database services.
- WebKit, the fast web-rending engine.

Application Framework Layer

Application framework layer provides a large amount of application framework components, such as activity manager which is in charge of managing application life cycles and content providers that controls data exchange between applications.

Application Layer

As the top layer of entire Android software stack, application layer with the help of both native and third party reside components from application framework layer, provides various services to the end users.

2.6.3 Android, Dalvik and Java

Table 2.7: Minimum Android Recommendations [Dav10]

Feature	Minimum Requirement
Chipset	ARM-based
Memory	128 MB RAM; 256 Flash External
Storage	Mini or Micro SD
Primary Display	QVGA TFT LCD or larger, 16-bit color or better
Navigation keys	5-way navigation with 5 application keys, power, camera and volume controls
Camera	2MP CMOS
USB	Standard mini-B USB interface
Bluetooth	1.2 or 2.0

As introduced above, Dalvik VM compared with the general Java virtual machine, takes the constrains that are specific about mobile device into account, such as hardware shortcomings including less memory space, low processing power, no swap space as well as short battery lifetime. Minimum recommendations for Android device [Dav10] are list in table 2.7.

When Dalvik virtual machine is applied, the compiling process is different from what is taken by Java VM. Figure 2.17 clearly pictures the difference. In Android runtime environment, after the first compilation of Java source code, the newly generated Java byte code will be compiled by Dalvik Dex complier, as a result, Dalvik byte code is created, which is going to be executed by Dalvik VM.

As pictured in figure 2.18, compared with *.class* Java byte code, *.dex* Dalvik byte code adopts shared and type specific constant pools with the main purpose of conserving memory [Dav10]. To be more specifically, the constant pool in Java byte code, which is painted blue in the left side of above-mentioned figure, is heterogeneous, which means all kinds of constant pools are included, even if they might be unnecessary. As a consequence, duplication may occur in Java byte code.

Another obvious distinguish between Java VM and Delvik is that, the former one adopts stack-based architecture and the later one uses register-based architecture, that in comparison with stack-based architecture needs on average 32.3% less execution time [Dav10], which is obviously the better choice for the system that runs on mobile devices with limited battery life.

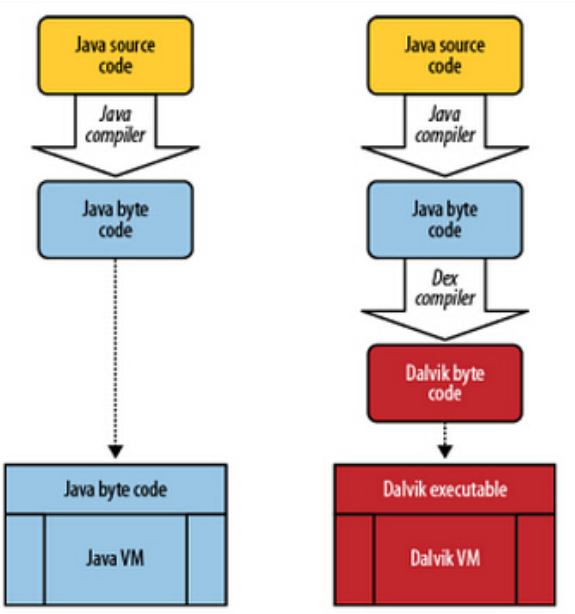

Figure 2.17: Compiling Process Comparison [Mar11]

2.6.4 Android Application Overview

One Android application consists of four categories of components, they are activities, services, content providers and broadcast receivers [Han12] described as following,

- *Activities.* The main responsibility of Android activity is to provide the end user an visual interface, with whose help the user can interact with the application.
- *Services.* Services are the important components that actually provide functionalities to the user through the GUI interface.
- *Content Providers.* Content Provider provides standardized and unified interfaces in order to perform shared data exchange among different Android applications.
- *Broadcast Receivers.* The last component is the broadcast receiver, who is in charge of asynchronously communicate with Android system broadcaster and other applications. For instance, when device battery life runs almost out, broadcast receiver will then be informed by Android OS.

2.6.5 Intent

For the purpose of inter- and intra- application communication, Android applies *intent*, which is in nature a self-contained object, that includes the target application reference and alternatively the to be shared data. Intent can be also

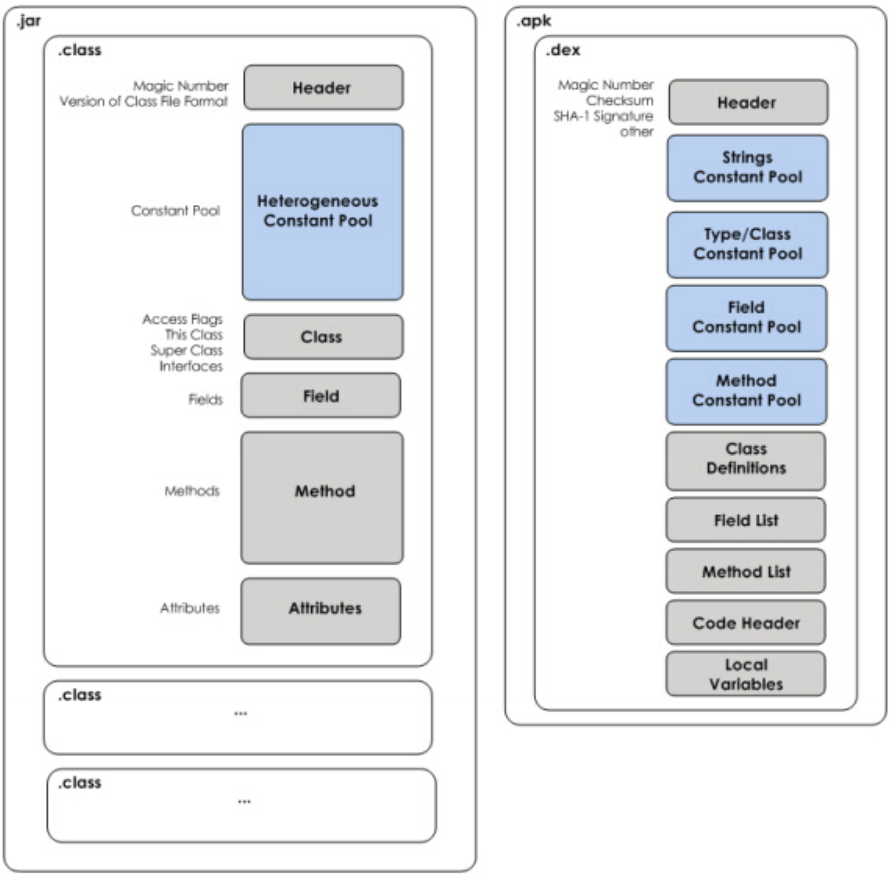

Figure 2.18: Comparison between Java and Dalvik byte code [Dav10]

understood as a well-formed message format that constructs the Android message passing system [Eri11].

2.7 Conclusion

With the acknowledgment of above-introduced technologies and standards, I now have gather theoretical supports for my proposal. To be more precisely, my demonstration *Smart Home* system will adopt OPC UA architecture discussed in section 2.1.1 to build a common connectivity interface for various housing devices and cell phone. In this system, housing devices and associating OPC UA server applications will be simulated as web services, whose design and implementation will be shown in section 5.3. Also an Android application named *Smart Home App* will be presented to simulate application level functions provided by OPC UA client. This *Smart Home App* will run on the Android platform introduced in section 2.6. And with the help of OpenMobileAPI introduced in section 2.5, this application is capable of secure exchanging data with smart card. The smart card employed in my scenario is integrated with the *CommunicationStack* applet,

which is designed by me using Java Card technology introduced in section 2.3 and applying GlobalPlatform remote application and file management protocols introduced in section 2.4.

The concrete design as well as implementation of my demonstration system will be introduced in chapter 5 and 6.

In the upcoming *State of Art* chapter, I will analysis practical industrial application cases in the realm of home automation technologies , OPC UA applications and secure embedded system design including smart card security mechanisms, OPC UA secure policies, secure Android and embedded system design guide lines. Those aforementioned concepts and summaries of best practices provide me enlightening ideas about how to design, improve and perfect my demonstration scenario and system.

3 State of Art

In this chapter, I firstly discussed the current researches in the realm of home automation system, in order to achieve inspiration materials for my demonstration scenario. Also after studying the industrial smart home application cases listed in this thesis, I have concluded that present-day automation systems commonly require a well-defined communication interface among all the housing devices, system as well as information security is still strongly demanded, which in return proves that my proposal is meaningful and promising.

The second topic covered by this chapter is the study of security. In more details, I have studied security technologies and mechanisms employed in OPC UA standards as well as Smart Card system and Android applications. With the acknowledgment of aforementioned studies, I have learned the potential threads to my application system as well as how to provide corresponding countermeasures. Moreover, in the last paragraph, I introduced the general guild-line for the development of a secure embedded system, which enlightened me about how to design my demonstration system.

3.1 Smart Home Research

3.1.1 Overview

The smart home system, which is also described as automated home, integrated home system or intelligent building [Vin], has drawn more and more industry developers' and researchers' attention over the decades. Research groups from for instance Siemens, IBM, Cisco, Microsoft has already contributed in this domain [Li 04]. A great number of Smart Home applications, network protocols as well as gateways have come into world and been applied to benefit their customers [Dim02].

With the development of Smart Home technology, nowadays' Smart Home not only is in charge of monitoring and controlling lighting and heating devices inside the building, but also capable of connecting almost every electronic housing devices, predicting inhabitant behavior as well as making scheduler decisions. Functionalities provided by intelligent home are not just limited to turn device on and off, record and report senor data, but include self-adjusting the inner building environment and supporting various predefined patterns, such as energy saving pattern. Especially the concept of Smart Home for elderly [Sib08], which perfectly combines modern remote control and monitoring technologies with senior-friendly

and patient-concerning housing techniques, is welcomed by the market.

Three categories of Smart Home will be introduced in the following as best practice examples, they are Smart Home optimized for energy services , Smart Health Home and Agent-based Smart Home.

3.1.2 Smart Home Optimized for Energy Services

This type of Smart Home sets its main goal in the energy saving and monitoring domain, which helps householder to make wiser decisions under the energy crisis background.

The key component in this Smart Home is decision-support tool [Mic10], that applies a scheduling algorithm which offers house owner energy saving suggestions based on various variables such as distributed energy resources (DER), with the prospect of saving precious energy and resource.

Decision Support Tool

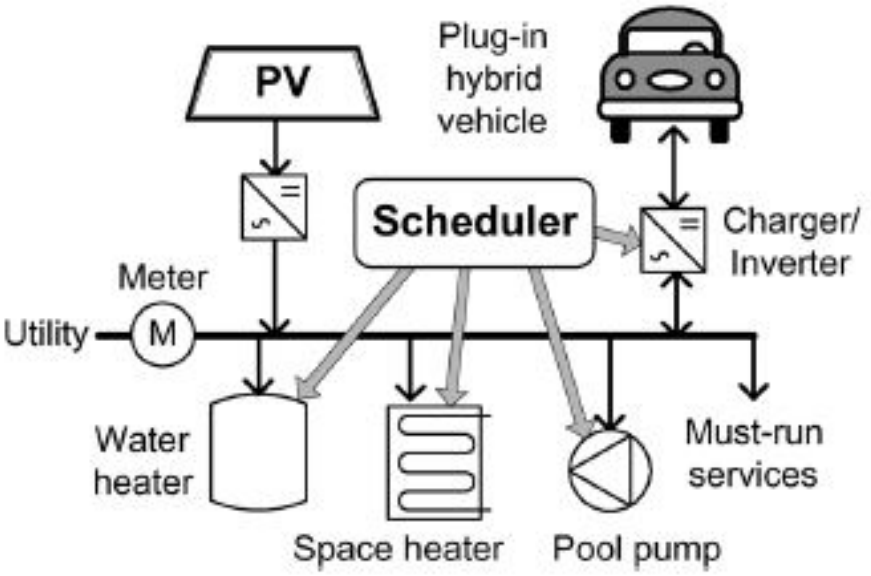

Figure 3.1: DER Scheduler in Smart Home optimized for energy[Mic10]

The decision support tool used in paper [Mic10] consists of two components, they are energy service model which describes the energy service request and distributed energy resource scheduling algorithm, as described in figure 3.1. To be more specifically, energy service model presents the demand of one particular energy resource. For instance the demand aimed at hot water means, the hourly consumption of heated water or the energy that is hourly needed by water heater. According to paper [Mic10], the heat content of water is also defined as "energy equivalent" and therefore the main goal of decision support tool is to increase "monetary benefit" from every "energy equivalent" unit.

The DER scheduler algorithm, that helps householder to reduce the unnecessary consumption of energy, is in nature one mathematical optimization problem defined as following [Mic10].

$$\sum_{t=1}^{T} \sum_{i=1}^{S} [\lambda_{ES,i}(t) \cdot U_{ES,i}(t,x)] - Cost$$

The purpose of DER scheduler presented as x is, to maximize that above introduced fitness function, where S represents all number of services offered by Smart Home, T stands for the whole simulation time, $\lambda_{ES,i}$ and $U_{ES,i}$ describe the desired monetary benefit for "energy equivalent" and energy demand of the ith service respectively. *Cost* means the total electricity consumption. Also in paper [Mic10] the choice of DER algorithm is well discussed.

3.1.3 Smart Health Home

Another suitable application domain for Smart Home is the Smart Health Home, which describes the intelligent housing that takes care of patients and elder residents.

The charming feature of this Smart Home system is the combination of home automation technologies with telemedical system, which provides customized services such as, teleconsulting, telediagnosis, real time imaging as well as long distance medical education [Vin02]. Smart Health Home will improve the living condition of householder and at the same time build a concerning system that takes care of residents' need. Nine best practice examples are provided and evaluated in paper [Sib08], they provide a guide-line for the design of Smart Health Home system and summarizes precious experience.

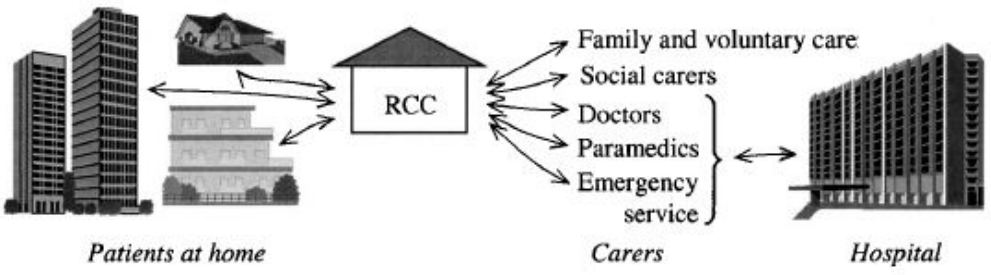

Figure 3.2: Overview of Smart Health Home structure [Vin02] (RCC: Remote Control Center)

3.1.4 Agent-based Smart Home

Agent-based Smart Home aims to build an intelligent home, which is based on machine learning, artificial intelligence technology and mobile computing. The core features of agent-based Smart Home are, prediction of householders' behaviors and making appropriate decisions. The achieved prediction can help residents

to experience a more comfortable and convenient living condition. MavHome (Managing An Intelligent Versatile Home) builds the best demonstrating example [Dia03] .

MavHome Architecture

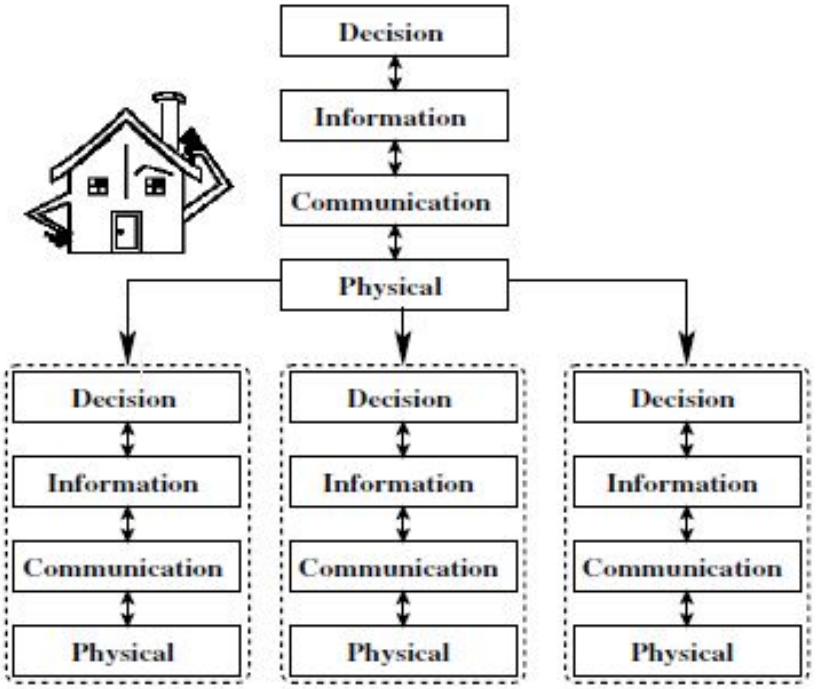

Figure 3.3: MavHome agent architecture [Dia03]

As shown in figure 3.3, agents which are employed by MavHome consist of four different layers. From bottom to top:

- *Physical layer*, where Smart Home system device hardwares are deployed. Moreover, underlaying agents can also act as physical layer for other high level agents.
- *Communication layer*, this layer provides communication service for agent by using functionalities offered by physical layer.
- *Information layer*, as higher layer of Smart Home system, the responsibilities of this layers are gathering and maintaining information which is going to be required by decision layer.
- *Decision layer*, in this layer agents make decision as well as learn resident's preference. Moreover, with the help provided by learning algorithms, agent

is also able to correct the unwanted system behavior and learn from bad decisions.

Prediction Algorithm

Prediction algorithm is the core component of MavHome project. several strategies are presented on the first IEEE international conference on pervasive computes and communication, they are SHIP algorithm that is based on sequence matching, compression-based prediction algorithm ALZ and Task-based Markov model [Dia03].

3.1.5 Conclusion

As described above, Smart Home is the representative application which is based on industrial automation, remote management and coordination system. Despite the decision making algorithms and application level components from above mentioned smart buildings are different from the other, but they all are integrated upon or connected with housing devices. The management and cooperation with those devices must take place under a secure system environment. None of householders are willing to be monitored by a strange third party using his own camera or to expose their daily life related information to the public. Therefore one common request for the design of such intelligent buildings is to ensure the system security.

Based on the fact, that home devices applied by Smart Home are manufactured by various vendors, Smart Home designers also must take it into account, that how to realize the system interoperability. Considered those two requirements, the proposal of this thesis is proved as practical and desired.

3.2 OPC UA Application and Security Policy

3.2.1 OPC UA Applications

OPC UA, as explained in former paragraphs, is understood as a platform independent, well designed, security concerning, IEC standard compliant, promising industry standards set, which provides a service oriented architecture and is being widely applied in industry fields such as critical control system and industrial automation. Application cases are given in following paragraphs, each of them has different focuses. To be more specifically, the charming demonstrated characteristics of OPU UA applications are: strong security protection, real time data exchange as well as coordination and scalability.

OPC UA and ICS

With the development of industrial automation technology, Industrial Control System (ICS) has became a hot topic, but most manufactures invested much more effort in designing automation as well as manufacturing process and neglected the communication security issues. As a consequence, cyber security of ICS is now drawing more and more attentions. OPC UA offers manufacturers who are applying ICS system not only sophisticated object oriented modeling rules, which can be extremely helpful for developers to design domain specific model and manufacturing process, but also provides them a reliable and robust security model [Ste], that has various secure arrangements in each layer of software architecture.

OPC UA and Smart Grid

Smart Grid is now considered as the future of electricity energy industry and therefor how to design an intelligent electricity distribution system, which coordinates the customer-supplier relation, becomes the most popular topic [Seb11]. Electricity industry is searching out for a standardized communication and connectivity technology to solve aforementioned issue. OPC UA standards set that includes *alarm and even* model, *data access* model as well as *historical data access* model, is without doubt one the of best candidates. With the help of all aforementioned models, the secure real-time peer communication and coordination among stockholders are guaranteed.

Nano OPC UA

Apart from those enterprise level systems, OPC UA is also suitable for lower level field devices, such as field sensor and other resource limited facilities. Recently a German company Lemgo even designed a nano embedded OPC UA server [Jah13], which provides the core OPC UA server service set. adopts TCP binary communication protocol and is implemented on a chip device.

3.2.2 OPC UA Secure Policy

At the beginning of OPC UA standards design phase, the OPC foundation has taken the construction of a secure and robust communication protocol as the center of their work and therefore developed an enhanced consistent security model which has clear and definite objectives in each layer of OPC UA software architecture. At the present day, OPC UA offers secure messaging mechanisms by applying WS security with Simple Object Access Protocol (SOAP) and alternatively SSL based Hypertext transfer Protocol Secure (Https) messaging [And13]. Besides secure messaging protocols, authentication mechanisms based on username-password or X.509 certificates are also introduced, in order to perform mutual authentication among OPC UA applications from different logic levels and hardware devices.

OPC foundation introduces the term *secure policies* [OPC09a] to describe the user authentication, user authorization, application authentication as well as message encryption mechanisms proposed by one OPC UA server profile. In this server profile, both secure related requirements and other server functionalities are described. One server is capable of maintaining several profiles in order to provide distinguishing services to different clients, who might have various security demands. Meanwhile one client can also accept a list of profiles, each of them is assigned by a different server.

3.2.3 OPC UA Secure Consideration

Along with core services set including data read and write operations, notification mechanism and etc, OPC UA server also provides the *discover service* [OPC09c], which provides OPC UA clients the mechanism to require server secure policy. Moreover a *secure channel service* set [OPC09c] is also designed, which is use to create and manage secure channel with the acknowledgment such as key deviation algorithm and message encryption algorithm, that are described by a secure policy.

Also OPC UA specifications provide guide lines for a secure system design [OPC09c]. They are:

- *appropriate timeout.* Timeout mechanism is widely employed in systems that adopt client-server architecture. With reasonable configured timeout property, both client and server can avoid unnecessary resource consumption caused by long time waiting of response from the other, which could be the consequence of an unexpected physical device failure or denial of service attack, which is intentionally initiated by an adversary.

- *Message exchange rate control.* Rate control is considered as one of the most practical approach to prevent denial of service attack. Under rate control mechanism, each client has limited chance over a period to build communication channel, reconstruct this channel, require information from server or send data to server. Alternatively the OPC UA server can also ban or block a client for a period of time, who was recently trying to flood messages.

- *random number generation.* The generation of random number is required by most encryption, authentication and authorization algorithms. Therefore a secure system must support and provide a robust random number generation function.

- *strict message processing*, which means the exchanged message between two communication partners must be compliant with predefined message format. Any ill-formed messages must be ignored, which in turn helps to avoid stack overflow attack and enhances system security

- *historical data management*, this strategy ensures the system traceability and records any behaviors taking place in system. The recorded data can be used for forensic research, when security related issue happens.

3.2.4 Conclusion

Above pictured OPC UA standards' features and characteristics prove the feasibility of applying OPC UA as communication protocol for Smart Home proposed in this thesis and the study of industrial application cases helps me to present this point better.

3.3 Smart Card Security

Integrated Circuit Card, ICC for short, was firstly designed and patented in Germany [Joh02]. Since then this credit sized, embedded with circuit chip card is being applied in a great number of industry domains working as for instance information storage media, on-line access token, identification method, small amount payment mechanism and etc. With the development of smart card technologies, nowadays apart from traditional *contact smart card*, *contactless smart card* [Nim] has come into the market. This *contactless smart card* is able to communicate with card accepting device without direct physical connection. To be more precisely, it applies radio frequency to contact with CADs. Now matter what categories of smart card, the tamper resistant nature always contributes to make them to be the most popular secure token media.

3.3.1 Smart Card in Access Control Management

The present-day concept of *security* no longer just meant the secure of our homeland borders or the individual personal safety. In this information age, the word *security* also refers to the confidentiality and integrity of users' personal data, the robustness of information system and so on. It is immediately related with our daily life. Especially when electronic devices, which can carry sensitive as well as precious information and could be vulnerable to malicious attackers, are playing irreplaceable roles in our society.

Smart card, as explained in section *fundamental technologies*, is believed as one of the best secure mechanism for access control and user identification.

Traditionally there are three ways to identity a person [Wol08]:

- *1 knowledge about a secret*, the secret is normally a password that negotiated earlier between two identification peers.

- *2 possession of an object*, this object could be anything that is predefined. One example could be the key that can open a particular door.

- *3 biometrics identification.* First two approaches have their own limitations, because third part could steal the object, that is used for identification, or copy user's password. Moreover it also could happen, that a password is too complex to remember or identification object is difficult to carry with. Compared with them, biometrics identification however depends on the unique human body characteristics, such as fingerprint and retina recognition.

Based on the above described facts, smart card is the best candidate for user identification and access control. Because when anyone wants to perform identification using smart card, he must possess the card (possession of an object satisfied), input PIN to unlock the card (knowledge about a secret). At the same time, complex passwords for the purpose of peer authentication and authorization can be stored on the smart card, which helps card holder to release the burden of remembering them. Also on a chip name card, which is usually carried by a card-holder in order to access a secured room or building, there usually exists a signature signed by the card-holder or his photo, that could be use to perform biometrics identification.

Nowadays with the help of standards such as *remote application management* protocol, which is introduced in *fundamental technologies* section 2.4, smart card is also capable of performing on-line identification based on either certificate or TLS protocol.

Personal Identification Number

Whenever an user intends to use one smart card, he must input a *personal Identification Number*, which is also referred as *Card Holder Verification*. PIN consists usually of four decimal number from 0 to 9 and tolerates at most 3 times wrong PIN input before a successful verification. PIN is stored in elementary files [CHA] on smart card, in order to be protected from unauthorized modification. After 3 times wrong PIN input, a smart card will be locked until the *unlock PIN* operation is successfully performed. Moreover if *unlock PIN* operation also fails three times in a roll, the PIN will never be unblocked [CHA].

3.3.2 Smart Card Key Management Mechanism

The above introduce PIN is not the only key that is stored on the smart card. In order to perform message encryption as well as decryption, peer authentication and etc. smart card must keep various keys in storage. As shown in figure 3.4, in a *Morpho* smart card product, a list of keys and key sets are designed such as Card Manager PIN and Administrative keys (*ADM1, ADM2, ADM5*).

Moreover, the key data stored on a smart card records not only the actual key value but also provides information such as key version number and the usage of key as shown in Table 3.1. Likely, in order to retrieve the value of a key from smart card, user must provide information such as key identifier and key version as shown in figure 3.5.

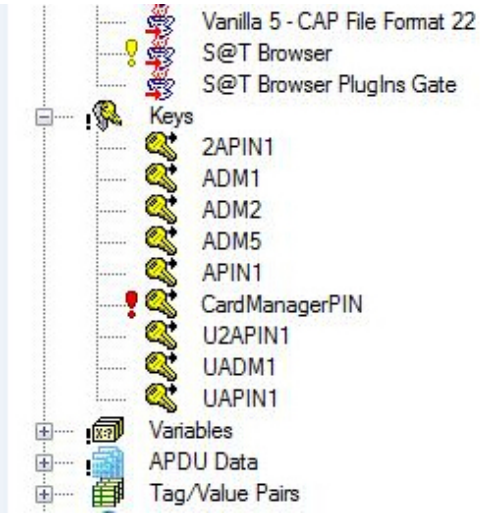

Figure 3.4: Keys Presented in Smart Card Description Language Developed by *Morpho*

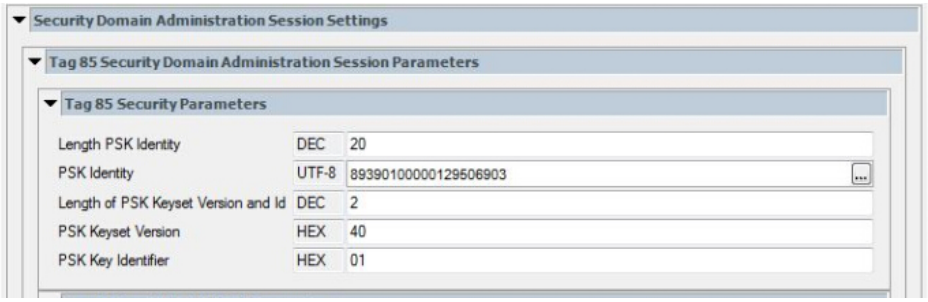

Figure 3.5: the PSK Key Used to Configure Administration Session

Table 3.1: Key Data Stored in Smart Card[Wol10]

Data object	Remarks
Key number	unique key reference number
Version number	version number used for key derivation
Intended usage	for which purpose this key is introduced
Blocked	used to block a key
Retry counter	wrong key input counter
Maximum retry count	key will be blocked if retry counter reaches this value
Key size	the size of key
Key	Key value

Since different keys play distinguishing roles in various system tasks and smart card must handle a huge number of keys, *key management* mechanism shows its importance. The core responsibilities of smart card key management system are listed as following [Wol10]:

- *key generation.* Key generation is the initial step of key life cycles and uses usually physical random numbers as initial data, in order to generate secure and unique keys.

- *key update.* This process keeps one key from being used for a long period of time. Because long time service of one key could lead to the compromise of system security.

- *key revocation.* When one key turns out to be compromised, key management system must destruct that key.

- *key storage.* Apparently a competent key management system must know how to safely store its secrets.

Key Generation

Derived Key Since smart card can be easily exposed and analyzed by anyone who holds or steals the card, in order to minimize the risk of key compromising, none master keys are present in the smart card. As a consequence, keys used in smart card are derived from unique card number, which is given during the card production phase, with the help of triple DES and AES cryptographic algorithms [Wol10].

Dynamic Key Dynamic key is normally applied to protect communication between peers and therefore it is also known as the session key or temporary key. The generation of dynamic keys begins with the generation of a random number that is proposed by one communication partner or unique data which is able to specify one particular session. There is a number of dynamic key generation approaches and I will take *ANSI X 9.17* standard as an example [Wol10].

$$Key_{i+1} = e(Key_{gen}, e(Key_{gen}, (T_i XOR Key_i)))$$

The function described above is one-way process and can not be reserved. To be more specifically, T_i and Key_{gen} are the time as well as session independent initialization input parameter and Key generation key respectively. The newly generated key key_i can not only be used for encryption but also to generate other new keys.

3.3.3 Threatens and Countermeasures

In this section, potential smart card vulnerabilities, threatens aimed at chip card as well as countermeasures will be discussed. But before I present the analysis of smart card threatens, the trust environment of smart card based system should be explained. Following five parties are involved [Bru99].

- *Cardholder.* In other word, the owner of smart card, who daily applies smart card for different purposes in various systems. For instance, using SIM card in cell phone to make phone call or using smart card digital wallet to perform smart amount payment like paying the parking fee.
- *Terminal.* Terminal is the card accepting device that communicates with smart card through smart card I/O.
- *Card Manufacturer.* As the name indicates, card manufacturer is the one who manufactures the card.
- *Card Issuer.* Card Issuer is the party that designs the card OS and initializes the smart card. For example, if we are taking about a cell phone smart card, then card issuer will be the cell phone service provider. When it comes to employees' ID cards which are applied as access control tool for a company. Then in this situation, the card issuer is the employer.
- *Software manufacturer.* This party is the one who designs applications that run on smart card.

Threatens and Vulnerabilities

Since threat modeling for smart card could be extreme complex and also the classification of smart card is various. In this paragraph four representative threaten categories are presented. They are,

- Logical attacks. Whenever we develop a software, there could exist a bug/bugs. And logical attacks just refer to the attacks that make use of bugs in software implementation [Hoo11]. Two examples are given as follow:
  - *Hidden Commands.* This kind of attack attempts to abuse commands from *initialization phase* of smart card life cycle, to modify or retrieve sensitive data from smart card in poorly designed smart card system. The abused command should have been inactive but not in above mentioned vulnerable system [Hoo11].
  - *Malicious Applets.* Malicisou applet is a Ill-designed smart card software, that attempts to break smart card system and steal information.
- Physical attacks. This type of attack aims at reverse engineering data and functions that are contained on smart card. Normally expensive and modern lab equipments are required [Hoo11]. Also physical attack is known as *invasive attack* [Phi06]. Three kinds of physical attacks are introduced as following :
  - *Micro-probe station.* Using Micro-probes, adversaries try to construct electrical connection with smart card chip, in order to observer communication between process and memory. With the observed information,

attacker is capable of capture sensitive data such as key information and etc [Phi06].

- *Focused Ion Beam.* Also it is possible to transmit intentionally generated signals to smart card processor using focused ion beam. As a consequence, it is possible for attacker to reveal data from smart card [Phi06].
- *Chemical Solvents, Etching and Staining Materials.* Using aforementioned chemical materials, attacker can observe etching speed difference from some ROM memories and with a step further analyze *0* and *1* in those memories [Hoo11].

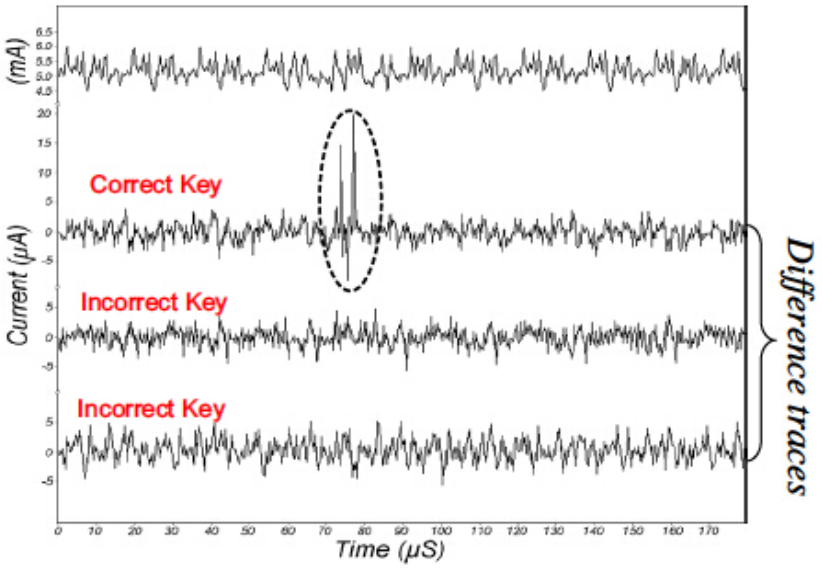

Figure 3.6: Differential Power Attack on a DES implementation [Pau04]

- Non-invasive attacks. This kind of attack makes use of smart card runtime environments such as difference in power consumption, processor voltage, clock frequency and etc. to analyze smart card behaviors. For example,
  - Power consumption attacks. The execution of smart card operations relays on power provided by terminal. When attacker is able to observe smart card power consumption, which is used by card performing cryptographic algorithms, he may retrieve related keys by applying *Differential Power attack* or *Single Power attack* [Phi06]. Figure 3.6 pictures a differential power attack analysis on a DES implementation.
  - Timing attack on RSA. The computing duration of RSA algorithm depends also on the input key. Therefore there is an opportunity for attacker using observed time gap in key computation process to get secret key information [Joh02]

- Other Attacks[Rak14].
  - Denial of service. This kind of attack aims at impacting smart card performance and consuming smart card system recourses to harm the availability of smart card service .
  - Eavesdropping. Through eavesdropping exchanged information is catapulted and analyzed.
  - Cover transaction. Fake session or session hijacking. Those approaches try to forge a communication session between smart card and the other connection peer, in order to gain information, which is related with peer authentication, authorization and messaging.

Countermeasures

Logical Attacks Countermeasure In oder to prevent this category of attacks, the number of bugs in smart card application should be minimized. Smart card software developer and card manufacturers can apply *structure design* by dividing application into small units which are easy to be tested and understood. Alternatively companies related in smart card industry should work together to propose standardized interfaces and protocols to regulate the smart card application development process [Hoo11].

Physical Attacks Countermeasure Smart card manufacturers have already realized the potential invasive attacks and are enhancing the physical security of smart card by offering:

- Reducing chip size. Now the size of smart card internal components can reach 150 nm, which makes it significantly hard for attacker to perform physical attack [Phi06].
- Metalization layers, which prevent smart card from atmospheric effects [Phi06].
- Multi-layered chips. Smart card manufacturers are now able to build chip card consisting of multi-layers. Vulnerable components, for instance ROM memory, are deployed in the lowest layer and protected from physical analysis [Phi06].
- Sensors. Smart card is protected by sensors covered with resin and those sensors will disable smart card whenever they detect physical attacks [Ahm11].

Non-invasive Attacks Countermeasure Countermeasures against above described non-invasive attack are:

- Against Timing Attacks by computing *superfluous calculations*, in this way adversaries are not able to get correct timing gap anymore [Phi06] .

- Against Power Consumption Attacks by applying digital noise. Alternatively chip manufacturers can reduce the electromagnetic emissions to make power consumption analysis harder[Phi06].

Other Attacks Countermeasure Other attacks are normal computer security issues and can be avoided by for instance applying message encryption mechanism, building communication session based on secure channel, introducing message transmitting rate control policies and defining appropriate timeout counts.

3.3.4 Smart Card Secure Applications

As given above, smart card is considered as a secure data storage media and a superior tool to protect information system security. People also describes smart card as the *magic bullet*, that can solve computer security issues, such as access control management, peer identification and so on. Understandably, more and more smart cards are applied in secure applications and products. For instance:

- In Payment System. Recently the concept of electronic purses and electronic payment systems becomes more and more popular. Not only because this newly proposed mean of payment can play a supporting role for traditional payment methods such as credit card, but also because with the integration of smart card, electronic purses are able to offer flexible, reliable and secure small amount payment services [Wol10].

- For Peer Identification. Based on the knowledge from section 3.3.1, Smart card holder is capable of performing two factor authentication [Joh02], which requires that at least two of three listed conditions are satisfied. Therefore smart card are suitable for the domain of personal identification. Since December 1999, Finland has begun to use electronic personal ID cards to replace traditional passports [Wol10].

3.3.5 Conclusion

In conclusion, the above discussed security features of smart cards, such as sophisticated key management system and counter measures against potential threads, prove the feasibility and reliability of my proposal, that is applying smart card for secure credential management and adding additional protection for the system which adapts smart card. Moreover with the help of industrial standardized protocols which are introduced in section 2.4, card holder can enjoy a secure remote management service and trust the electronic devices which are integrated with smart cards, let those facilities administer his properties.

3.4 Secure Android Design

With the development of smart phone market, more and more subscribers are using this new generation of cell phones with their smart cards. Accordingly an increasing number of attacks aiming at smart phone platforms is reported.

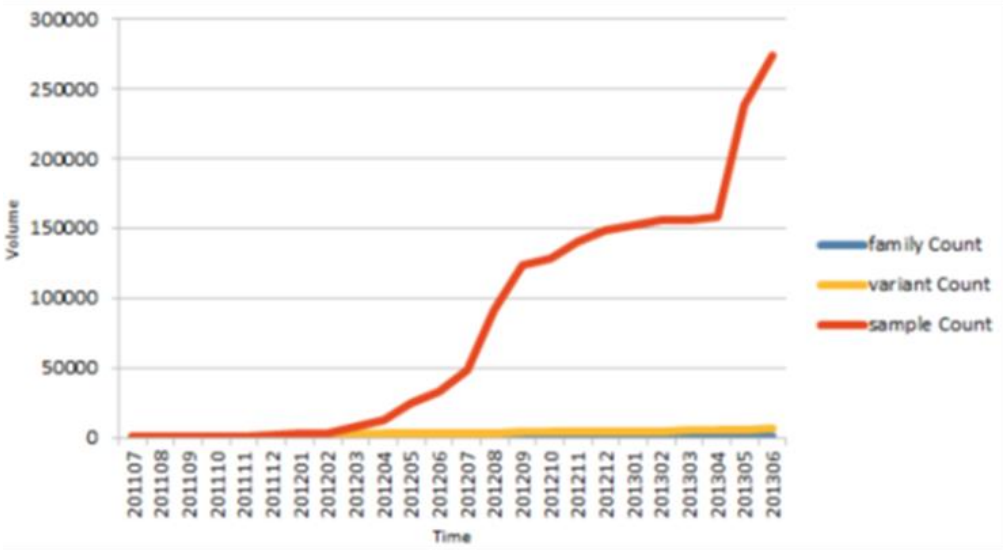

Figure 3.7: Android Malware Growth [Bar13]

Figure 3.7 shows a dramatic increasing amount of Malwares that target at Android platform from report [Bar13]. Especially when taken it into account that present-day smart phones are also in charge of sensitive users' information such as recent visited location as well as contacts list. And based on a study of Android applications, researchers have found that over 20% applications could have access to user personal data [Han12]. Consequently there exists a pressing requirement for Android application developers to design their Android software with strong security protection.

3.4.1 Android Security Mechanism

Even though a huge number of third party applications exist on Android market and they could be potential Malwares, but Android platform is not vulnerable and it provides secure as well as robust security mechanisms to protect its customers. Following listed three methods are the most outstanding and efficient approaches.

- *Application Isolation.* Each Android application has its own Linux process and each of this process possesses an isolated virtual machine. In this way, application data is separated and protected from the other [Avi09].
- *Access Permissions.* Permissions are given during the applications' installation time. As illustrated in figure 3.8, Android permission system restricts

the access right to user's private data, to other Android applications , to smart phone hardwares and to Android OS.

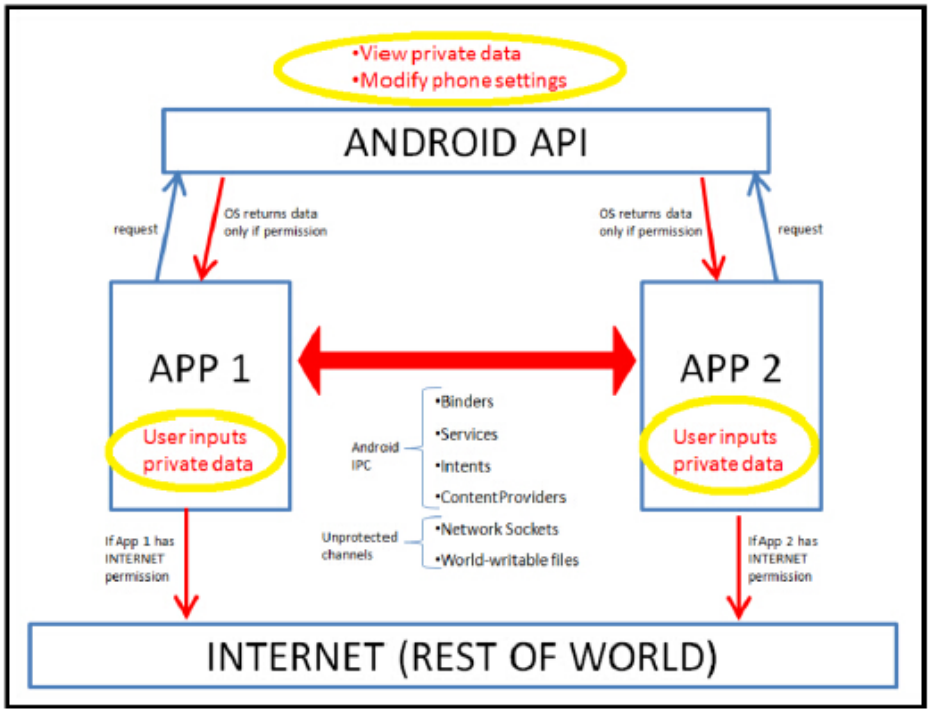

Figure 3.8: Detailed Android Permission Model [Han12]

- *Authorized Signatures.* Applications are signed with certificates. The corresponding private keys are protected only by the application developer. Every time when one application claims his identity, his signatures will be checked, in order to prevent the potential masquerade attacks [Avi09].

3.4.2 Security Design Guide-Lines

With the acknowledgment from previous paragraphs, it can be concluded that Android system is designed with consideration of security, especially in the domain of access control and permission management. But still ill-developed application is vulnerable. For instance, when an application carelessly sends an intent introduced in section 2.6.5, which has none explicit target. This implicit intent can be intercepted by a Malware. As a result, this malicious application may manage to access the sensitive data of victim application and perform attacks such as *activity hijacking* and *service hijacking* [Eri11].

Therefore it it necessary for application developers to apply protection techniques in oder to protect their application from attackers and revers engineering. In the following, three secure Android design guide lines are summarized.

Application Components Secure

In order to restrict the number of applications, which have access rights to the Android application components described in subsection 2.6.4. Android application designer should explicitly claim the access control policies in *AndroidManifest.xml* from their project. In extreme case that none applications are expected to invoke the secured component, following rule should be added in *AndroidManifest.xml* [Kei13].

Component Securing

```
<[component name] android:exported='false'>
</[component name]>
```

Default
RECEIVE_BOOT_COMPLETED
READ_CONTACTS
READ_SYNC_SETTINGS
READ_OWNER_DATA
GET_ACCOUNTS
WRITE_CONTACTS
ACCESS_NETWORK_STATE
INTERNET
VIBRATE
WAKE_LOCK
WRITE_EXTERNAL_STORAGE
READ_KEY_DETAILS
Custom
READ_ATTACHMENT
READ_MESSAGES
DELETE_MESSAGES
REMOTE_CONTROL

Figure 3.9: Permission Required by Application *k9mail* [Han12]

Custom Permissions for Access Control

As pictured in figure 3.9, Android OS provides a list of default permissions, which serve a generic access control management, including such as phone holder data reading and writing, Internet access right and etc. But when it comes to the case of sharing data between different applications, a more sophisticated and robust custom permission is desired. To be more specifically, following xml snippets, which declare customized permission rules, should be added to the protected components in project's *AndroidManifest.xml* [Kei13].

Custom Permission Snippet

```
<permission android:name='android.permission.CUSTOM_PERMISSION'
        android:protectionLevel ='dangerous'
```

```
android:description = 'Custom Permission Snippet'
android:label = '@string/custom_permission_label''>
```

Moreover four categories of protection level are supported [Kei13]:

- *normal.* This is the lowest level of permission and usually applied for non-dangerous permissions.
- *dangerous.* In this kind of permission, there exist the risks that authorized application would be able to access user credential information and sensitive data.
- *signature.* Signature permission means that applications which are signed with the same certificate can access each other.
- *signatureOrSystem.* In addition to *signature* level, this level of permission will be also automatically granted to the application that is part of system image.

Securing Content Provider

Last secure approach that I want to present is the content provider path secure mechanism. Since content provider handles mostly the exchanged data between applications, therefor it frequently becomes the first victim of a system attack. Application developer must ensure the security of content provider, especially the content provider path, which is in nature an uniform recourse identifier that is related to sensitive information such as datasets [Kei13]. Content provide path may look like *content://com.provider.android/account/balance*.

It is highly recommended that application developer explicitly defines the permission rules at path level, one demonstration example is given as following [Kei13],

Content Path Securing

```
<provider ...>
<grant-uri-permission android:path = 'path_name'>
</provider>
```

3.4.3 Conclusion

In conclusion, Android is a secure platform with sophisticated access control mechanisms and other security polices. But still Android application developers should apply security techniques to protect their project from malicious and ill-formed applications. Moreover I developed my Android application *Smart Home App* following the above presented Android secure design guide lines. The concrete design and implementation details will be presented in chapter *System Design*.

3.5 Embedded System Secure Design

3.5.1 Overview

In this last section of this chapter, a generic guide for the embedded system secure design is introduced. As already in previous sections described, low level components that build the embedded system compared with traditional computer software have additional security requirement. For instance, low end devices are normally dependent on battery as electricity supplement and have limited computing capacities. Those two constrains must be taken into account during the design phase of a secure embedded system.

3.5.2 Embedded System Security Requirements

The *Trinity of Trouble* [Pau04] three factors, namely *complexity*,*extensibility* and *connectivity* contribute together to make the embedded system more vulnerable and complicate the system security requirement. In the following a brief discussion about embedded system security requirements and corresponding solutions are presented [Pau04].

- *User Authentication.* Before the embedded system performs any functionalities, the identity of system user must be confirmed. Alternatively the user also has the need to verity the embedded system's identity. The expected mutual authentication can be realized by applying secure protocol such as *TLS 1.2* or using digital certificates and signatures.

- *Content Security.* The security of exchanged message also plays an import roll in a secure embedded system. Message integrity and confidentiality must be ensured. Popular solutions are to apply cryptographic algorithms such as symmetric ciphers, asymmetric ciphers and hash algorithms to protect system content from illegitimate modification and eavesdropping.

- *Secure Network Connection.* Most devices nowadays have the access to the Internet. As a consequence, secure network connection belongs also to the security domain in embedded system. In order to provide a secure network environment, system developer should apply secure communication protocols such as SSL or create secure VPN connection [Pau04].

- *Secure Storage.* System data must be protected from unauthorized modification and stealing by malicious entities. Therefore tamper resistance hardware should be applied.

- *Availability.* At last the service availability must be guaranteed by apply anti-DoS attack mechanisms, which are already introduced in this thesis.

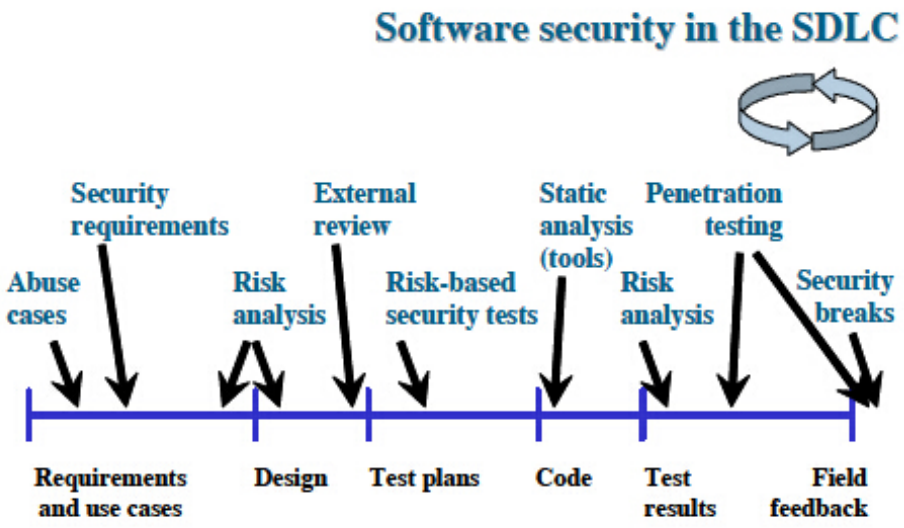

Figure 3.10: Software Security Consideration in the Software Design Life Cycle [Pau04]

3.5.3 Software Security during Software Design Life Cycle

With the acknowledgment of system security requirements as well as corresponding solutions, embedded system designer should concern potential security threads as well as system imperfections in each application design phase and perform corresponding security analysis as well as run secure-related tests, as shown in figure 3.10.

3.5.4 Secure Architectures

In oder to keep an embedded system from malicious parities' attacks, especially when present-day system is normally exposed in a ubiquitous networking and pervasive computing environment [Pau04]. System developer should design a robust and secure system architecture with the security consideration from architectural micro level to macro level. Figure 3.11 pictures the choice of architectural design space.[1]

When the hardware architecture characteristics of an embedded system are confirmed. Following two secure aspect as shown in last two rows of figure 3.11 should be considered

- *Security Processing Features.* Various fundamental secure protocols and algorithms could be selected by system designer. But some of them may

[1] ASIC:application-specific instruction set processor. EP: embedded general-purpose process. SPx: special purpose extensions. GPx:general purpose extensions. HWaccel: hardware accelerators. GOP: co-processor

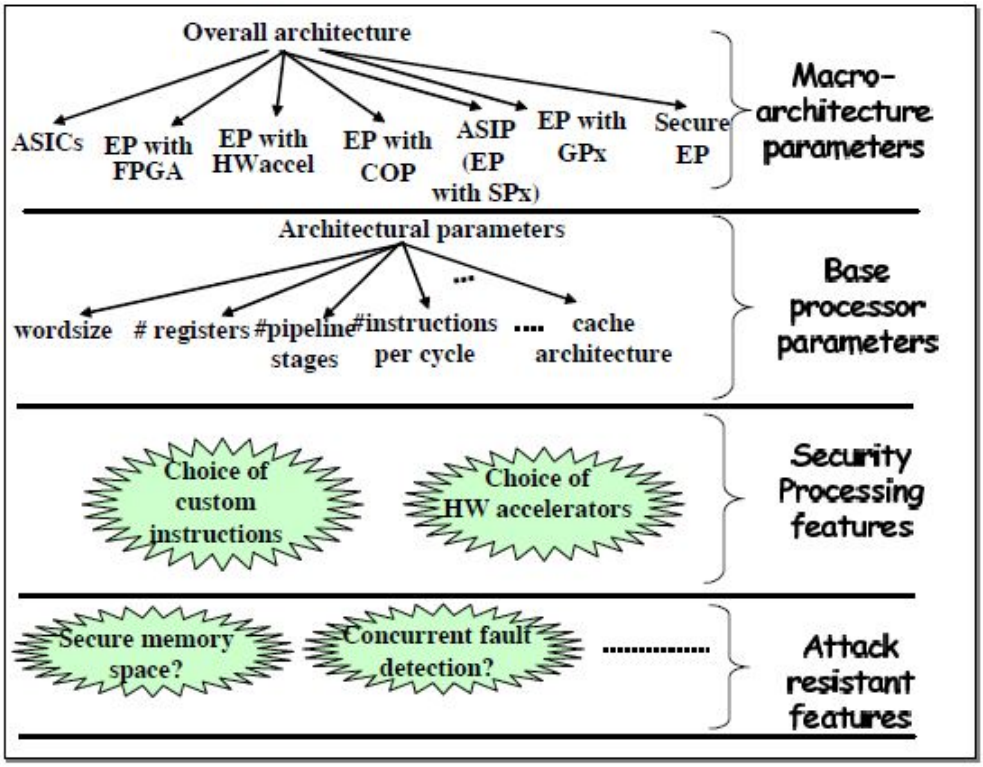

Figure 3.11: Architectural Design for a Secure Information Processing [Pau04]

offer the same security services. Therefore in this layer, the choice of cryptographic algorithms should be made with the consideration of hardware and software specific aspect. For instance, the full implementation of SSL protocol for a simple low end filed device is apparently not a wise decision [Pau04].

- *Attack Resistant Features.* When general security mechanisms used to ensure content confidentiality, data integrity and peer authentication are provided by an embedded system, security mechanisms to prevent other kinds of security attacks such as denial of service attack also should be designed.

3.6 Conclusion

In this chapter, I presented industrial application cases in the realm of home automation, OPC UA specifications and smart card products. At the same time, I emphasized the importance and significance of security in those applications. Moreover, I summarized the security mechanisms employed by smart card and OPC UA applications. A generic developer guide line for secure Android application design as well secure embedded system development is provided. Especially, based on the concept presented in section 3.5, before the development of my ap-

plication system, I have firstly analyzed the security requirements in my scenario and set clear security objectives in each layer of my system components. The five in section 3.5.2 mentioned security requirements are satisfied in my system. The *user authentication* is protected by the PIN as well as password-login mechanism employed by smart card and my *CommunicationStack* applet. Moreover, the applied TLS 1.2 secure handshake process provides not only a *secure network connection* to my system, it also can exam the identify of a communication peer. Furthermore, the during secure handshake generated master key together with the secure messaging approaches integrated in my system ensure the *content security*. Also I have applied smart card as the secure token, which perfectly serves the *secure storage*. At last, the *avalability* of services is protected by message rate control mechanism introduce by OPC UA specifications.

In next chapter, I am going to present the concept of my Smart Home system and the demonstration scenario.

4 Implementation Scenario

In this chapter, I will briefly describe my implementation scenario, including the system architecture overview, system component structures as well as functionalities and communication flows. Also the implementation tool support will be introduced.

4.1 Overview

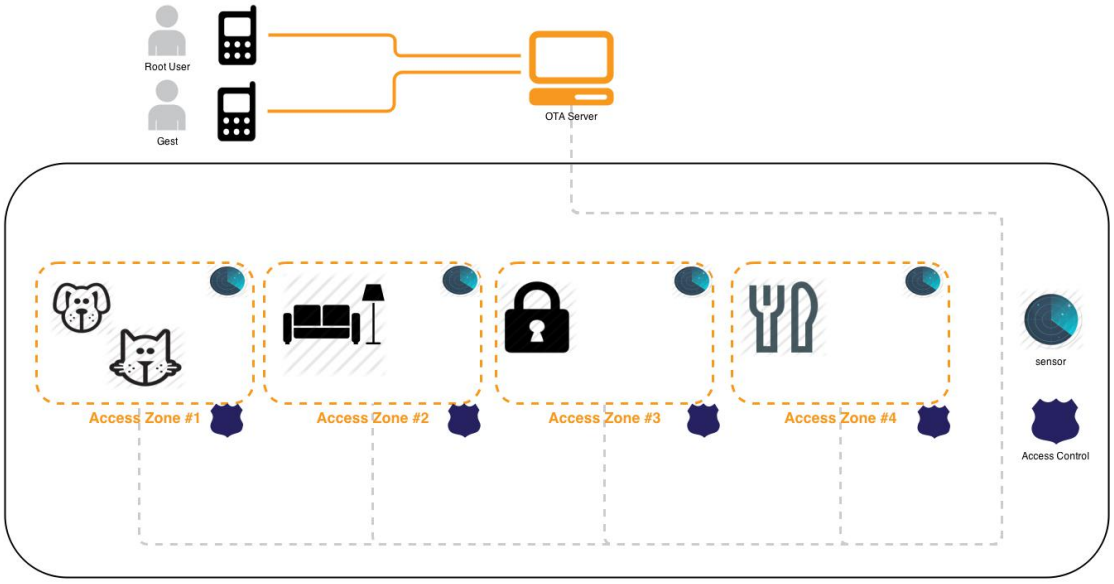

Figure 4.1: Smart Home Scenario

Figure 4.1 describes the basic structure and functionalities provided by implementation scenario - Smart Home. In the above-pictured automated home, following devices are introduced.

- Housing devices. Such as temperature sensors, coffee machine and digital door lock. They are integrated with OPC UA server application and provide house holder a comfortable and secure living condition by offering services such as:
  - General device state query functionality.
  - Door lock access control management.

  - Home temperature and luminance management and coordination.
  - Notification and alert.

- Smart Phone. With the help of the on smart phone installed Android application, *Smart Home App*, which provides OPC UA client functions, the house holder is able to remotely manage housing devices.

- Remote Administrator server (Over The Air Server) and smart cards. Smart cards, which are integrated in housing devices and smart phones, are installed with Java card applet *CommunicationStack*, that realizes the communication stack of OPC UA architecture. The duties of smart card and remote server peers are communication partner authentication and identification, public key management as well as secure message transmitting.

4.2 Component Structure

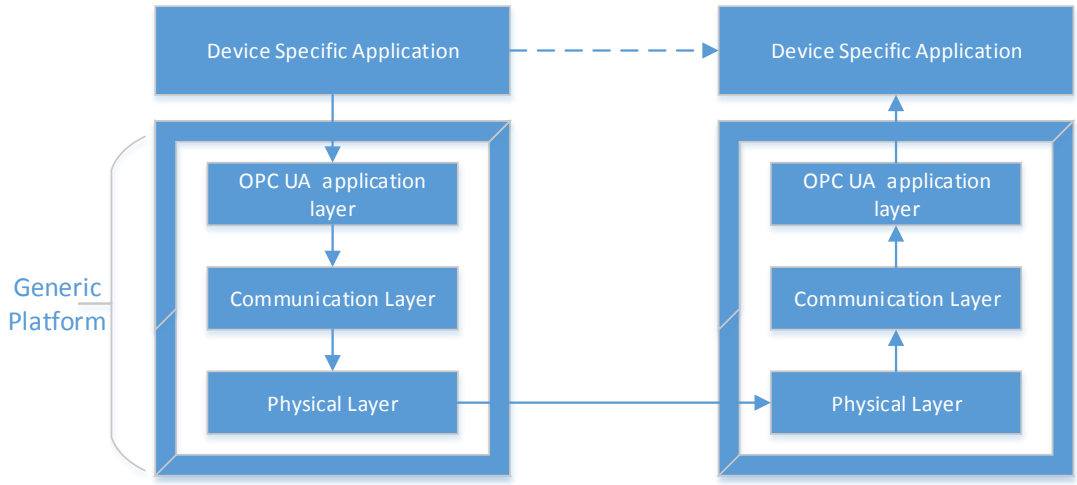

Figure 4.2: Smart Home System Component Structure

With inspiration offered by application described in section 3.1.4, in order to provide a generic application platform for devices manufactured by various vendors, my demonstration Smart Home system consists of components, [1] whose basic structure is illustrated in figure 4.2. As can be seen, components are composed of a device specific application and a generic communication platform, which includes three layers. From bottom to top, they are:

- Physical layer. In this layer secure electronic devices and facilities hardwares are deployed. Alternatively lower level component could be also in this layer of higher level component.

[1] Housing devices and smart phone are generically referred as Smart Home system components.

- Communication layer. Smart card, the on card integrated *Communication-Stack* applet and corresponding remote file/application management protocols together form this layer. The main responsibility of communication layer is to create a secure communication environment for OPC UA application layer.

- OPC UA Application layer. OPC UA client application (installed on smart phone) and OPC UA server application (installed on other homing devices) are placed in this layer and provide a common communication and connectivity interface for higher layer device specific applications.

Moreover this component platform also emphasizes the importance of secure messaging, peer authentication as well as authorization and provides corresponding secure mechanisms, which will be discussed in upcoming paragraphs.

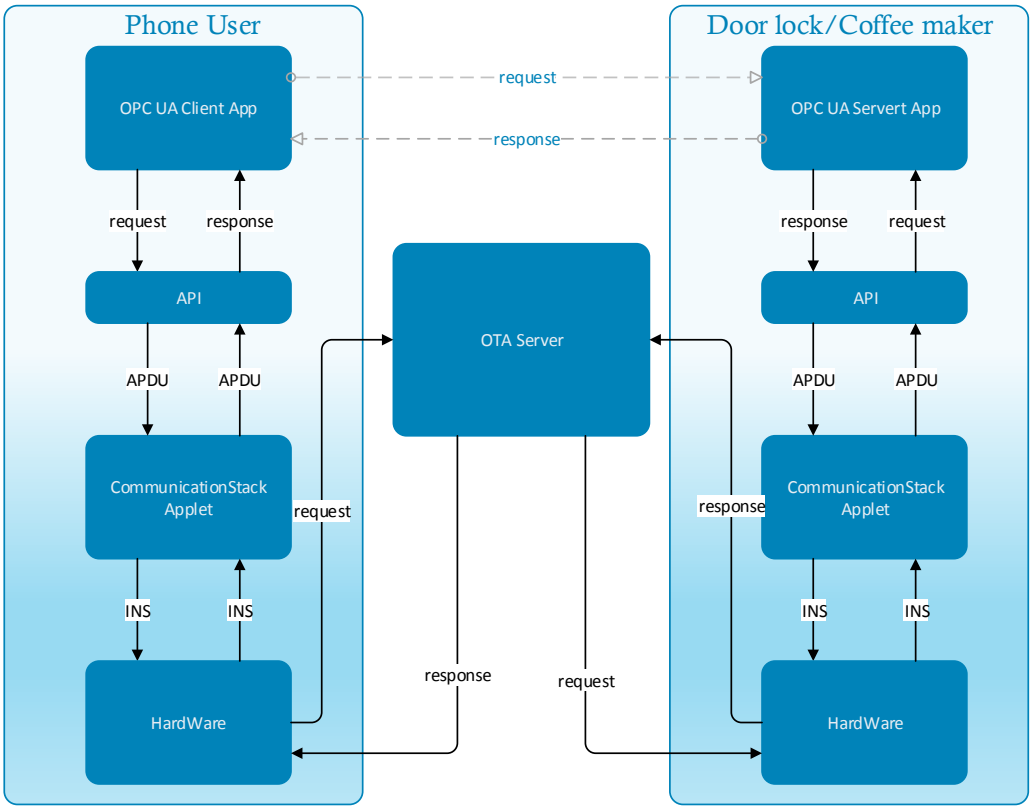

Figure 4.3: Smart Home Component Communication

4.3 Communication Flow

Figure 4.3 pictures the communication flow between two system components. The device hardware stands for the aforementioned component's *physical layer*. *CommunicationStack* together with the internal API construct the *communication layer*.

OPC UA client and server application communicate with each other with the help of an OTA server. And the *CommunicationStack* applet is in charge of creating and managing secure communications between OTA server and secure devices. A demonstration case is given as below:

When the householder wants to make some coffee, he will send the *makeCoffee* command to coffee maker with his cell phone in following steps as pictured in figure 4.3.

1. Phone user touches his screen and selects the *makeCoffee* command. Then OPC UA client application installed on the cell phone initially generates the *makeCoffee* request and forwards it to the internal API

2. The internal API translates the request into a remote APDU object and sends the APDU to the *CommunicationStack* applet. Before the *CommunicationStack* is able to forward the request APDU, it must create a secure communication channel with the remote server.

3. The remote server will then in this step firstly exam the identify of the message sender.

4. After a successful mutual identification, a secure channel will be established between smart phone and the remote server, the *makeCoffee* request is going to be transmitted to the remote sever with the help of this newly created secure messaging channel.

5. After receiving the command from message sender, remote server will find the correct message receiver and forward the command to him.

6. In the last step, the *makeCoffee* command will be received by the *CommunicationStack* installed in target coffee machine, whose *internal API* translates the request and forwards it to the OPC UA server application. The OPC UA server application will then control the coffee maker and perform the make coffee function. Alternatively it also generates a response message and notifies the phone user that its job is successful finished.

The two core system parts involved in above described scenario are *CommunicationStack* applet and *OPC UA client/server application code*

4.3.1 CommunicationStack Applet

CommunicationStack is integrated in UICC smart card, whose duties are realizing secure channel as well as session management, transporting data to receiver using TCP/IP connections or SMS. To be more specifically, the *CommunicationStack*'s responsibilities are:

- initiate session based Http connection, which is protected by TLS 1.2 protocol (proactive)
- trigger Http session based on received trigger SMS, that is proposed and send by OTA server (passive)
- rebuild broken communication channel
- message encryption as well as decryption
- message transmit

4.3.2 OPC UA Application Functionalities

In this paragraph, a brief introduction about functionalities offered by the OPC UA client and server application code is presented.

In conclusion, OPC UA server on secure housing device provides following services:

- processing client's subscription
- publishing notification.
- client authority management.
- historical data record.
- execution of client's command.
- managing the homing device on which the OPC UA server runs.

Basic OPC UA client functions as following are provided by smart phone applications:

- submitting subscription
- receiving published data from OPC UA server.
- sending command and configuration data.
- querying system historical record.
- providing user friendly GUI interface.

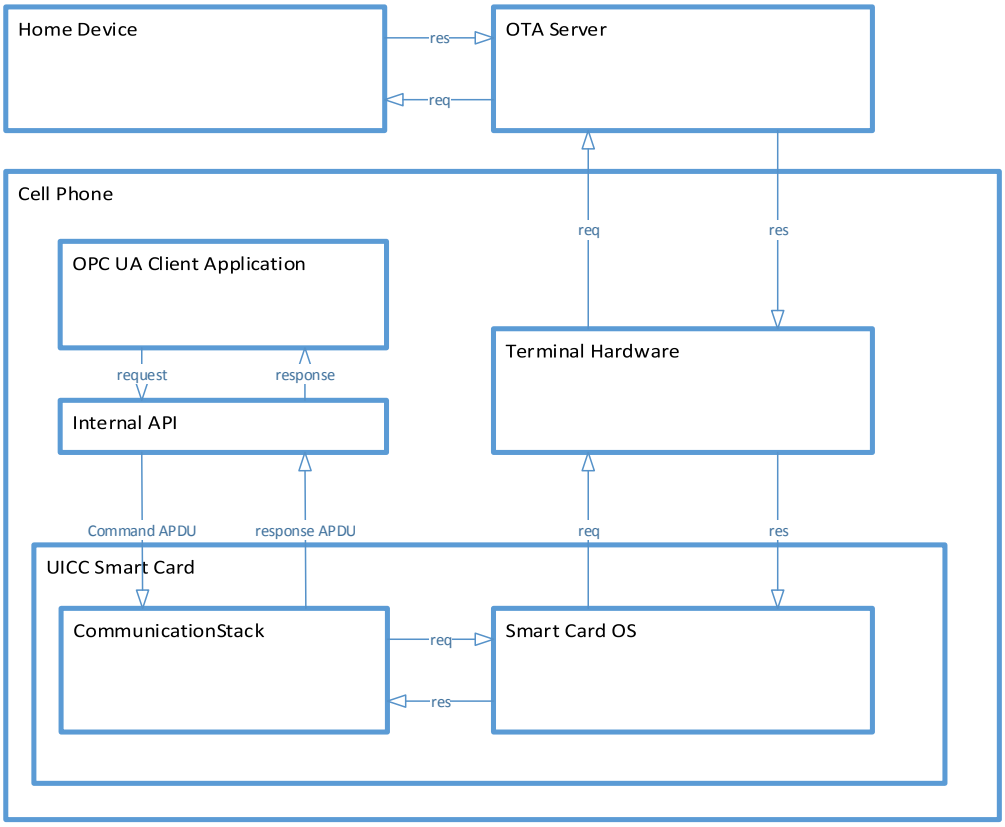

Figure 4.4: Cell Phone Component Architecture

4.3.3 Cell Phone Component Software Structure

As described in figure 4.4, the smart phone introduced in my application scenario consists of OPC UA client application code that realizes client application level functions, internal API, *CommunicationStack* applet, smart card and phone hardware.

The remote communication flow applied in my application scenario is compliant with the one provided by GlobalPlatform as shown in figure 4.5. GlobalPlatform defines the mechanisms for secure information exchange between a remote entity and a terminal, which is also known as Remote Application/File Management(RAM/RFM) over Http protocol.

Application Security Domain (APSD) issued by GloablPlatform represents the terminal device in the realm of RAM/RFM and the aforementioned remote entity is also referred as Remote Administration Server.

With these two concepts, smart card holding the appropriate Security Domain can act as a Http client and is capable of packing APDU format information into Http POST message and transmitting this Http message to the OTA server, who

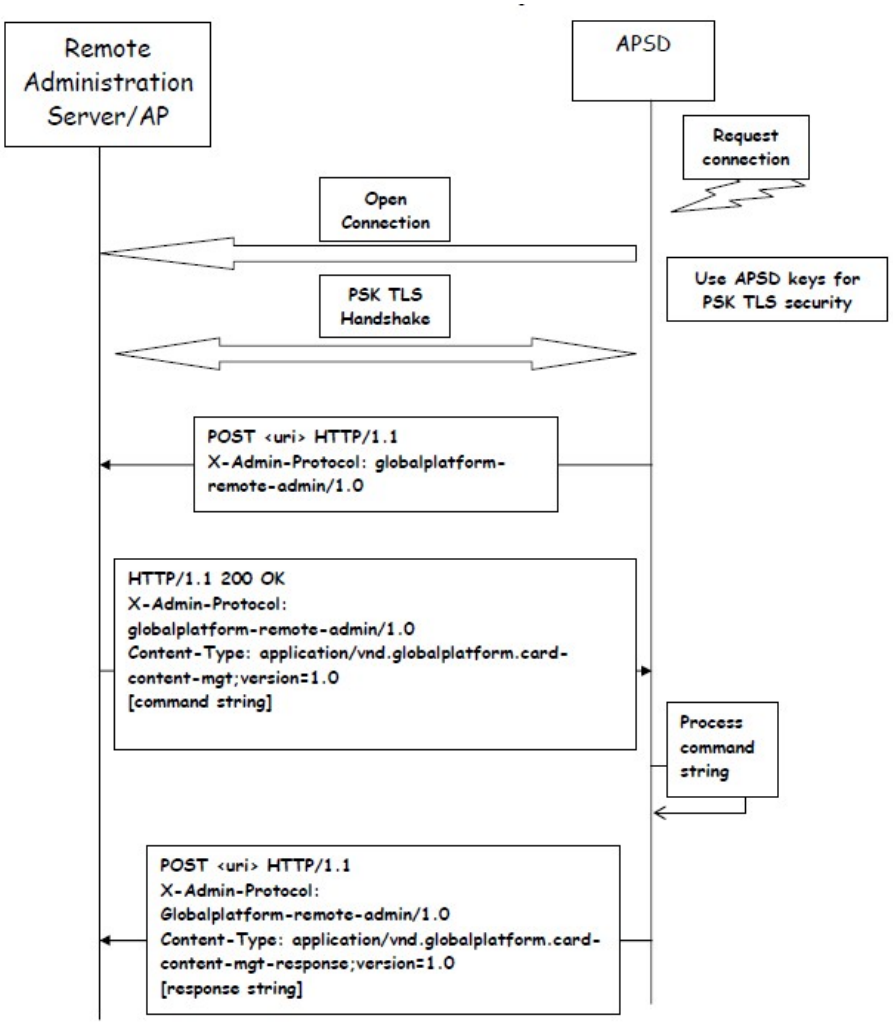

Figure 4.5: Communication Flow between AP and APSD [Glo12]

will forward the message to the target receiver [Glo12]. Moreover in oder to protect the confidentiality and integrity of exchanged information, I proposal that a *public key infrastructure* is also integrated in the OTA server, which manages the public keys of Smart Home housing devices and adds additional security protection to my system.

Figure 4.5 illustrates a typical RAM communication flow between administration server and corresponding security domain (Application Security Domain) on smart card. As can be seen, the request for open communication channel is usually initialized by security domain, which represents the phone user. After a successful secure handshake, the remote administration server and security domain are able to based on Http connection exchange request and response strings, which encapsulate APDU instructions. GlobalPlatform has also provided a list of APIs used to initialize authentication process, to configure cipher suits and to perform

secure messaging.

4.3.4 Housing Device Component Software Structure

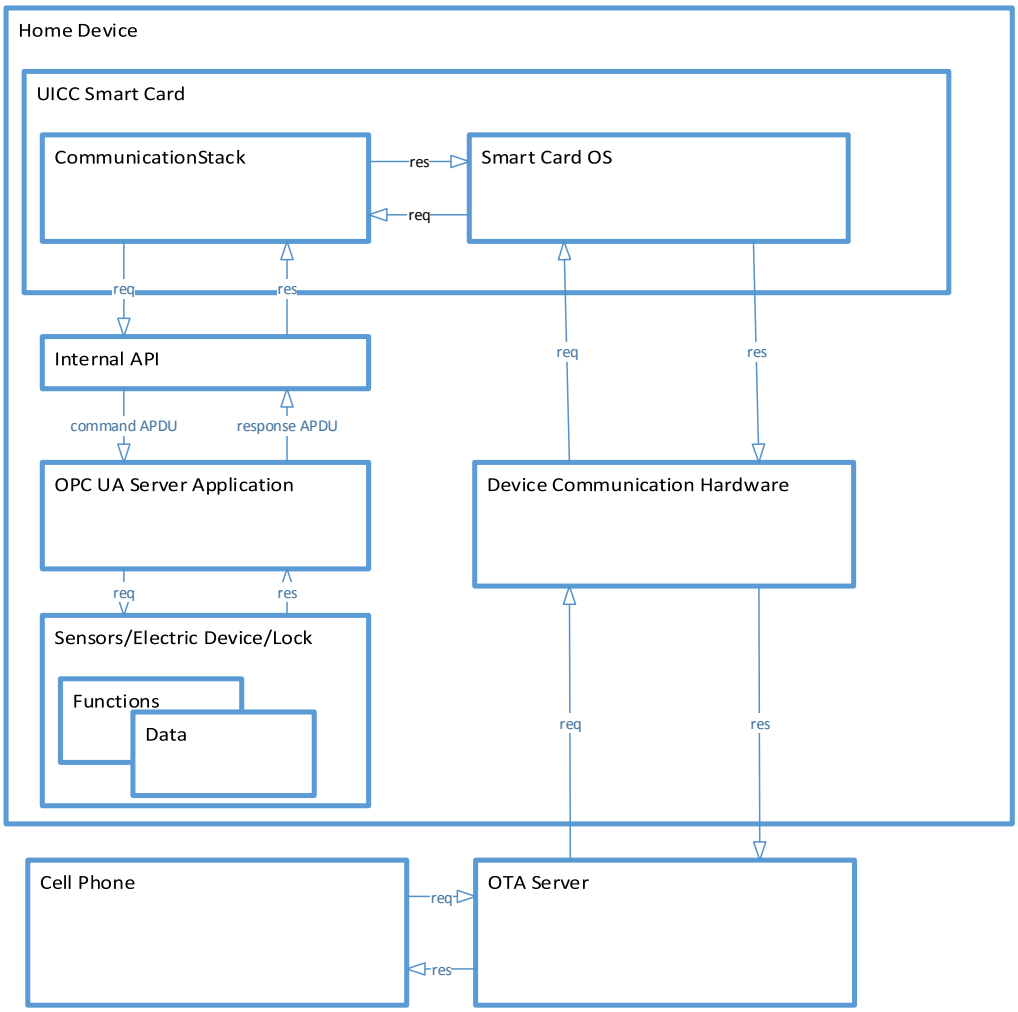

Figure 4.6: Housing Device Component Architecture

Housing Device here refers to sensors, household electrical appliances as well digital locks that together build up the Smart Home system. Each secure device provides services described in section 4.3.2. The device software structure is pictured as figure 4.6 and it consists of OPC UA server application code, which offers basic functionalities like subscription and notification mentioned before, an internal API, an on smart card integrated *CommunicationStack* applet and device specific functions as well as device data.

4.3.5 Implementation Tool Support

In this paragraph, I will shortly present the tools that I applied to design, program and test my Smart Home demonstrating system.

Java Card Application Design and Debug Tool

Morpho presents JACADE with full name, Java Card Applet Develop Environment, which is more than just a IDE but a complex selection of various class APIs and software modules, that can be applied to design complete Java Card applet as well as to debug Java Card source.

Java Card Applet Testing Environment

When it comes to running test with Java Card applet, tester has two options. Firstly, he can install the to be tested applet on a smart card and then connect this chip card with test computer using *Morpho Card Reader (MCR)*. Alternatively *Java virtual card* which is integrated with the to be tested applet could be also applied.

In either way, as next step *Universal Test Environment (UTE)* will be applied. *UTE* presented and developed by Morpho uses Java language developed test cases and test scenarios[2] to simulate desired use cases and observes corresponding smart card reactions. With the observed test result, tester will be able to analyze applet performance and debug corresponding application code. For instance *UTE* can integrate software models used to simulate security domain standardized by *Globalplatform* and perform the sophisticated user authentication process.

Android Application Design Tool

Eclipse IDE for Java Developer of version 3.7.1, together with Android SDK *seek-for-android* and ADT plug-in, are applied as Android application development environment. Moreover Android of version 2.3.5 is simulated and chosen as platform for the *Smart Home App*.

Smart Home Simulation Environment

In order to simulate the data managed by home electronic device, *MySQL* database is employed. And together with *PHP of version 5.5.12*, *Apache of version 2.4.9*, the web server for Smart Home is created. Android application *Smart Home App* communicates with this web server for the purpose of scenario testing and demonstration.

[2]Test scenario is a collection of relative test cases.

4.4 Conclusion

In this chapter, I briefly described the concept of my demonstration scenario and discussed the system components architecture as well as adapted communication flow. Moreover the supporting programming and testing tools are also presented. In the next chapter, I will detailedly describe how I design my *Communication-Stack* applet, *Smart Home App* and the Smart Home web application.

5 System Design

In this chapter, I will present the crucial information about how I designed and programmed my demonstration system.

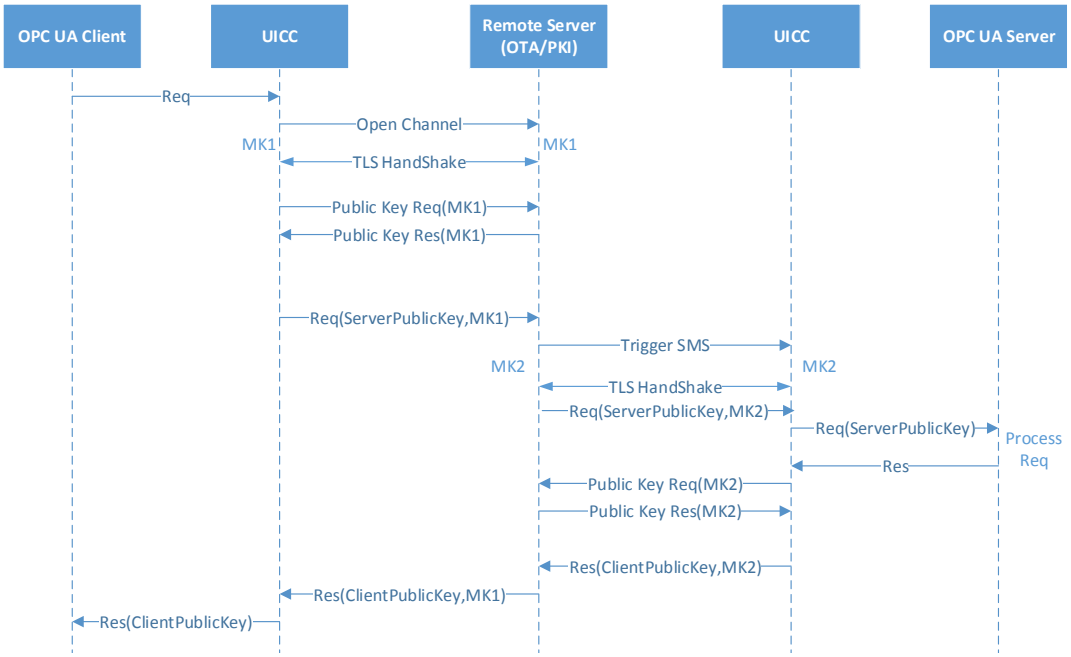

Figure 5.1: Request and Response Message Exchange Demonstration

Figure 5.1 illustrates the request and corresponding response message exchange process that occurs in the demonstration system. In my scenario, OTA server is also integrated with a *public key infrastructure*, in order to perform secure peer identification and messaging. Therefor this OTA server can also be generically referred as remote administrator server.

5.1 UICC Applet

5.1.1 Overview

The generic request and response message exchange process shown in figure 5.1 consists of following steps,

- *Request Initialization.* As the first step of communication flow, OPC UA client application usually generates the request message and forwards it to the smart card applet *CommunicationStack*.
- *Open Channel Phase.* Request sender initializes the *open-channel* command and sends this command to the remote administrator server.
- *TSL Handshake with Remote Server.* After a successful TLS 1.2 based peer identification, a secure communication channel between remote server and message sender is created. And the master key, which is generated during the TLS handshake, is applied to encrypt message exchanged between the aforementioned peers.
- *Require Target's Public Key.* When remote server identifies the connected communication partner, it will grant him the expected target's public key.
- *Secure Messaging.* Message sender will in this phase encrypt the request message with target's public key and send this encrypted message through remote server to the message receiver.
- *Request Forwarding.* In this step, remote server will send the request receiver a SMS to trigger the construction of a communication channel and use the newly created secure connection to forward the request message.
- *Request Processing.* Upon receiving the request, message receiver decrypts the message using his private key and performs actions based on the in request encapsulated commands. Alternatively, he will also send a response message to the requester.

5.1.2 Work Flow

The components involved in this scenario are:

- *CommunicationStack* applet and associated Security Domain defined by Globalplatform (APSD for short)
- OTA-PKI server which is also known as Remote Administration Server (RAS for short)

The communication between *CommunicationStack* and Remote Administration Server involves following steps:

- Open Communication Channel and Create Communication Session
- Secure Message Exchange
- Close Communication Session

Communication Session Creation and Message Exchange

This process as shown in figure 5.2 could be either initiated by Remote Administration Server by sending target applet a trigger SMS or be sponsored by the applet itself. In both cases, *CommunicationStack* applet sends the first *Open-Channel Request* message encapsulating supported cipher suits, random number and etc. During the PSK-TLS Handshake phase, *CommunicationStack* and remote server will agree on the to be used cipher suit and authenticate each other. After a successful PSK-TLS Handshake, the aforementioned communication peer is able to generate a master key and use this newly constructed key to encrypt exchanged messages, which can encapsulate APDU command or APDU response strings. The Http header is in compliance with GlobalPlatform standards and is shown as following.

Http Header Format

The Http message sent from remote sever to target applet abides following schema [Glo11]:

Http Request Schema

```
HTTP/1.1 200 OK [or HTTP/1.1 204 No Content CRLF]
X-Admin-Protocol: globalplatform-remote-admin/1.0 CRLF
[X-Admin-Next-URI:  <next-URI> CRLF]
[Content-Type: application/vnd.globalplatform.card-content-mgt
-response;version=1.0 CRLF]
[X-Admin-Targeted-Application: <security-domain-AID> CRLF]
[Content-Length: xxxx CRLF] or [Transfer-Encoding: chunked CRLF]
CRLF
[body]
```

The field *security-domain-AID* will be filled with the AID of target applet.

The Http response message sent from applet to remote server uses following schema [Glo11]:

Http Response Schema

```
POST<URI>HTTP/1.1 CRLF
Host: <Administration Host> CRLF
X-Admin-Protocol: globalplatform-remote-admin/1.0 CRLF
X-Admin-From: <Agent ID> CRLF
[Content-Type: application/vnd.globalplatform.card-content-mgt
-response;version=1.0 CRLF]
[Content-Length: xxxx CRLF] or [Transfer-Encoding: chunked CRLF]
[X-Admin-Script-Status: <script-status> CRLF]
[X-Admin-Resume: true]
CRLF
[body]
```

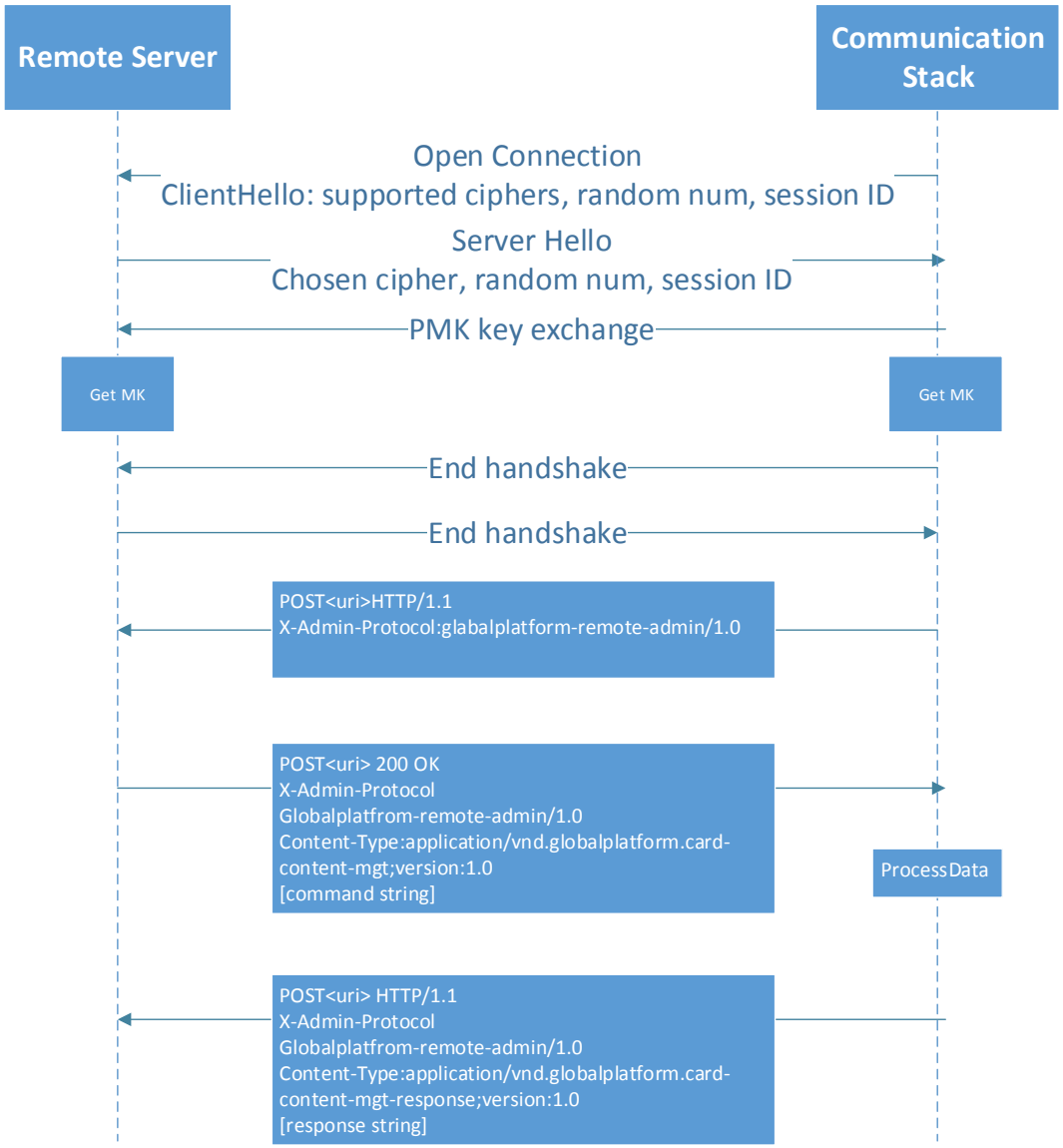

Figure 5.2: Handshake and Message Exchange between SD and Remote Server

The filed *X-Admin-Script-Status* could contain one of following values:

- *ok*, which means that the previous message is successfully received by applet.
- *unknown-application*, which stands for the error, that the target applet of previous message can not be found.
- *not-a-security-domain*, this errors occurs when target applet is not a Security Domain.
- *security-error*, as its name indicates, this values is returned when the security

of previous message can not be checked.

Close Communication Session

Whenever the communication channel is about to be closed, either because of session security issue, or due to the successful finish of communication, remote server will send the target applet a Http message with one of the following parameters:

- No *X-Admin-Next-URI* field is present in this Http message and message body is empty. This message will be recognized as final message from remote server and then the associating session will be closed.
- No *X-Admin-Next-URI* filed is presented but the body of this Http message is not empty. The receiver will process the data in body and close the communication session appropriately. But no response message will be generated.

5.1.3 Commands: Interface between Applet and CAD

Before concrete implementation of Java Card applet code, the interface, which in essence is a set of commond APDUs and corresponding response APDUs, between applet and CAD must be well defined. The CommunicationStack support two categories commond APDU:

- The *SELECT* Command APDU, which is used by JCRE to select *CommunicationStack* applet.
- Other command APDUs, which are introduced in order to provide functionalists such as: trigger communication session, process input APDU and etc. To be more specifically, following categories of APDU sets are designed :
  - PIN operation related APDU set
  - Communication session management APDU set
  - Data process APDU set

SELECT APDU

The header of this command APDU as shown in table 5.1 is fixed and *Lc* indicates the length of *CommunicationStack* AID. In *Data filed* real AID is saved.

Table 5.1: SELECT Command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0x00	0xA4	0x04	0x00	Length of AID	AID	N/A

Two categories of response APDUs as shown in table 5.2 are expected, one represents successful processing of *SELECT* command APDU and the other stands for failure.

Table 5.2: SELECT Response APDU

Optional data	Status word	Description
No data	0x9000	Successful processing
	0x6999	Failed to select CommunicationStack applet

Verify PIN operation

This APDU command and response set presented in table 5.3 and 5.4 respectively is used to let *CommunicationStack* applet verify the identity of terminal user. Moreover the PIN size is set from four bits to eight and the verify PIN operation only allows three times wrong PIN input before a successful identification.

Table 5.3: Verify PIN Command

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x11	0x00	0x00	length of Data field	PIN	N/A

As described in table 5.4, three categories of response APDUs are expected.

Table 5.4: Verify PIN Response APDU

Optional data	Status word	Description
No data	0x9000	Successful processing
	0x63C0	Verification failed
	0x6983	Blocked PIN

Reset PIN Operation

This APDU command and response set is introduced to offer the end user PIN change service.

Table 5.5: Reset PIN command

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x12	0x00	0x00	length of Data field	New PIN	N/A

Three categories of response APDUs are expected as shown in table 5.6.

Table 5.6: Reset PIN Response APDU

Optional data	Status word	Description
No data	0x9000	Successful processing
	0x6301	Verification is required first
	0x6984	Length of input PIN is wrong

Communication Session Creation

This APDU command and response set is employed to enable the *Communication-Stack* applet to establish communication channel with the remote administrator server. Commands listed in table 5.7 are supported. To be more precisely,

- *INS-0x01*. This command is applied to declare communication session variables.
- *INS-0x02*. This command is introduced to configure communication session parameters.
- *INS-0x03*. This command is used to create communication session.
- *INS-0x04*. This command is employed to retrieve session states.

Table 5.7: Communication Session Creation Command APDUs

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x01	0x00	0x00	data filed length	Session parameter	N/A
0xA0	0x02	0x00	0x00	data filed length	Session parameter	N/A
0xA0	0x03	0x00	0x00	N/A	N/A	N/A
0xA0	0x04	0x00	0x00	N/A	N/A	length of session state

Following return codes as shown in table 5.8 are expected in response APDU: .

Table 5.8: Trigger Session Return Codes

Status word	Description
0x9000	Successful processing
0x66AB	Array Index out of bounds exception
0x665E	Security exception
0x6600	Nullpointer exception
0x6C00	UnKnown exception

Close Communication Session

This APDU command and response set is used to correctly and appropriately close the communication channel.

Table 5.9: Close Session Command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x50	0x00	0x00	0x00	N/A	N/A

Following return codes as shown in table 5.10 are expected in response APDU: .

Table 5.10: Close Session Return Code

Status word	Description
0x9000	Successful processing
0x6032	Failed to close session

Table 5.11: Process Data Command APDUs

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x20	0x00	0x00	data filed length	desired target's public key	PK length
0xA0	0x21	0x00	0x00	data filed length	command data	response length

Process Communication Data

This APDU command and response set is used to perform command processing between cellphone and other Smart Home devices. Two types of commands as shown in table 5.11 as designed. The first command is applied in order to query target's public key from remote administrator server, and the second one is implemented to translate and process input command data. Moreover *command data* in data filed of command APDU adopts following TLV format as shown in table 5.13.

Table 5.12: Process Data Return Code

Status word	Description
0x9000	Successful processing
0x6A80	error in data filed
0x6A81	required device not found
0x6A82	required service not found
0x6A83	required record not found

Return codes as shown in table 5.12 are expected in response APDU.

Based on the in table 5.11 defined command structure, following APDU *A021000012901091089206 9C040102030402* can be understood as a command APDU, which has *class byte* as *A0*, *instruction byte* equals *21*, none *status words*, encapsulates a command data *9010910892069C0401020304* and expects 2 bytes response. The command data can be translated according to table 5.13 as a command sent by smart phone to the sensor whose unique id is *0x01 0x02 0x03 0x04*, for the purpose of querying current sensor value. Moreover any ill-formed command data will be ignored by the receiver.

5.1.4 Classes

My UICC applet shown in figure 5.3 is integrated in Morpho *SC5-01OS07* LTE SAT product and contains following classes:

- *CommunicationStack*, which is the main class of my applet, that implements *install*, *select*, *deselect* as well as *process* methods provided by *javac-*

Tag	Length	Name						Presence
’90’	0-n	command parameters						optional
		Tag	Length	Name				
		’91’	0-n	command from cell phone to home device				optional
				Tag	Length	Name		
				’92’	1-n	Read sensor value		optional
						Tag	Length	Name
						’9C’	1-n	device uid
				’93’	1-n	Set subscription		optional
						Tag	Length	Name
						’9C’	1-n	sensor uid
						’9A’	1-256	subscription value
				’94’	1-n	Get historical record		optional
						Tag	Length	Name
						’9C’	1-n	device uid
				’95’	1	Open main door		optional
				’96’	1	Coffee Maker add water		optional
				’97’	1	Coffee Maker add coffee		optional
				’98’	1	Coffee Maker make coffee		optional
				’99’	1-n	Grant main door access to		optional
						Tag	Length	Name
						’9B’	1-n	user uid
		’9D’	0-n	Command from home device to cell phone				optional
				Tag	Length	Name		
				’9E’	0-n	Notification sent by home device		optional
						Tag	Length	Name
						’9C’	1-n	device uid
						’9A’	1-256	value
				’9F’	0-n	Historical data		optional
						Tag	Length	Name
						’9C’	1-n	device uid
						”A0’	1-256	historical data

Table 5.13: Command Data Type-Length-Value Structure

ard.framework.Applet. Moreover in order to process remote APDU, this applet is also designed as UICC system applet, that extends interface *com.orga.javacard.componentinterfaces.JCISIMApplication*

- *sessionTrigger*, which implements Globalplatform and toolkitframework interfaces and offers functionalities such as, proactive and passive creation of communication session with remote administrator server, cipher suit negotiation and mutual authentication. *sessionTrigger* is developed based on *adminTrigger* class provided by Morpho.

Class CommunicationStack

As the main class of my Applet, *communicationStack* extends *UsimRemoteService* class provided by Morpho in order to be recognized as a system applet, which

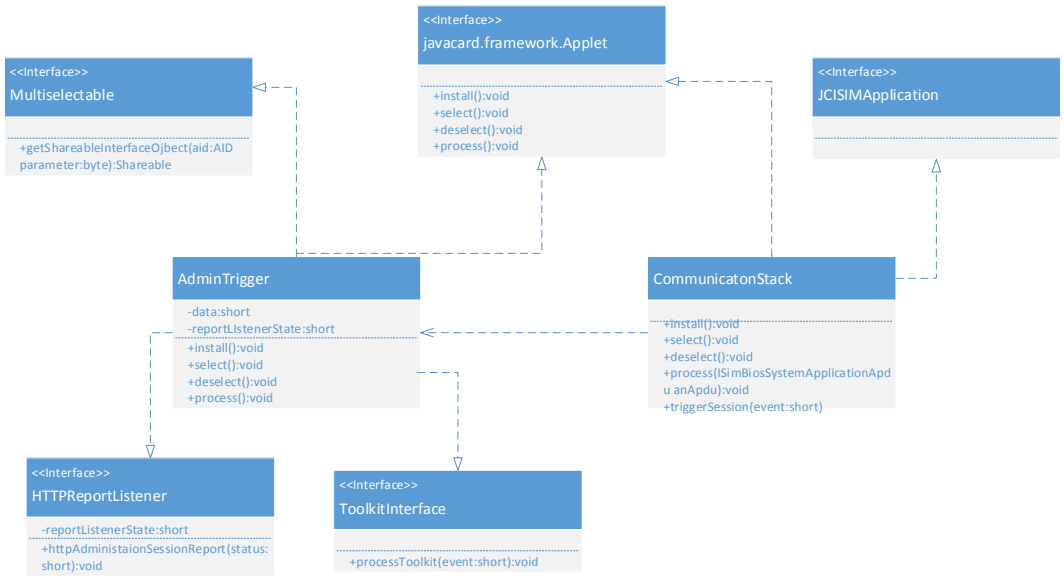

Figure 5.3: Class Diagram

is capable of performing remote application/file management behavior. Moreover this main class has implemented interface *JCISIMApplication*, which defines mechanisms to handle remote APDUs.

Also the PIN, which is required when using this applet, is defined and described as below,

```
private static final byte PIN_TRY_LIMIT = (byte) 0x03;
private static final byte PIN_MAX_SIZE = (byte) 0x08;
private static final byte PIN_SIZE = (byte) 0x04;
```

Every time if a user intends to use this applet, he must pass the PIN examination. System allows maximum three time wrong PIN input and PIN size is set from four bits to eight. Following two core methods provided by *communicationStack* is described.

Install Method *install* method is used to register my applet in JCRE and create a CommunicationStack implementation.

doVerifyPIN Handler

```
public static void install( byte[] bArray, short bOffset, byte bLength ) throws ISOException
```

process Method In this method, the received remote APDU command will be analyzed and upon on the encapsulated instruction filed, corresponding handler will be invoked based on the *command data* included in the same command APDU.

Following snippet defines how to get *class* filed, *instruction* filed , *command data* and alternatively an normal *APDU* object from a remote APDU object.

Remote APDU Operation

```
byte claMasked = (byte)(anApdu.getCla() & (byte)0xF0);
byte ins = anApdu.getIns();
byte[] cmd = (byte[]) anApdu.getCommandData();
short cmdOffset = anApdu.getCommandDataOffset();
byte[] apdu = anApdu.getApdu(); //return APDU object if available
```

Based on the acknowledgment of *claMasked* , *ins* data and *apdu* object, using below described switch statement, appropriate handler is invoked.

Switch Statement

```
switch(ins)
        {
        case INS_VERIFYPIN:
                doVerifyPIN(apdu);
        case ....
        }
```

In case when *ins* equals *INS_VERIFYPIN*, following *doVerifyPIN* is invoked.

doVerifyPIN Handler

```
private void doVerifyPIN(APDU apdu) {
        byte[] buffer = apdu.getBuffer();
        apdu.setIncomingAndReceive();
        if(pin.getTriesRemaining() == (byte)0) {
                ISOException.throwIt(SW_PIN_BLOCKED);
        ....
        }
```

In total six handlers are designed:

- *doVerifyPIN*. As already introduced, this handler performs the verify PIN operation. If the number of wrong PIN input reaches 3 times before a successful verification, *doVerfiyPIN* will block the PIN.
- *doResetPIN*. This handler realizes PIN change operation.
- *doUnlockPIN*. *doUnlockPIN* operation will be and only be invoked by card issuer for the purpose of unlock blocked PIN.

- *doGetPublicKey.* This operation queries target public key from *Public Key infrastructure* which is integrated in OTA server in my scenario.
- *doNotification.* After receiving notification data, *communicationStack* applet will invoke this handler in order to inform corresponding subscriber to fetch that command data.
- *doAppendMsg.* This handler configures and edits the outgoing response remote APDU.

Except from status words, if other information is expected to be transmitted back to the command APDU sender, the return data can be appended at the end of existing response data buffer using following schema.

Editing Response Data

```
byte[] result = {....};
short length = (short) result.length;
byte[] resBuff = anApdu.getResponseBuffer();
short resOffset = anApdu.getResponseBufferOffset();
Util.arrayCopyNonAtomic(result, (short)0, resBuff, (short)resOffset, (short)length);
anApdu.setStatusword(State_Word);
anApdu.appendResponse(resBuff,(short)resOffset, (short)length, true);
```

Byte array *result* represents the appended data and *resBuff* is the outgoing buffer.

Class SessionTrigger

sessionTrigger class extends *HTTPReportListener* interface in order to receive notification upon completion of a session. *triggerSession* method as shown below provides core mechanism to create a communication session configured by input parameter *byte[] data*.

Trigger Session

```
public void triggerSession(byte[] data, short dataOffset, short dataLength )
 {
final short FAMILY_HTTP_ADMINISTRATION = (short) (GPSystem.FAMILY_HTTP_ADMINISTRATION << 8)
GlobalService globalService = GPSystem.getService(null, FAMILY_HTTP_ADMINISTRATION);
HTTPAdministration httpAdmin = (HTTPAdministration)globalService.getServiceInterface
(GPSystem.getRegistryEntry(null),FAMILY_HTTP_ADMINISTRATION, null, (short)0, (short)0);
httpAdmin.requestHTTPAdministrationSession(data, dataOffset, dataLength);
}
```

5.2 Android Application

In order to simulate and realize application level OPC UA functionalities, I have designed the following Android application.

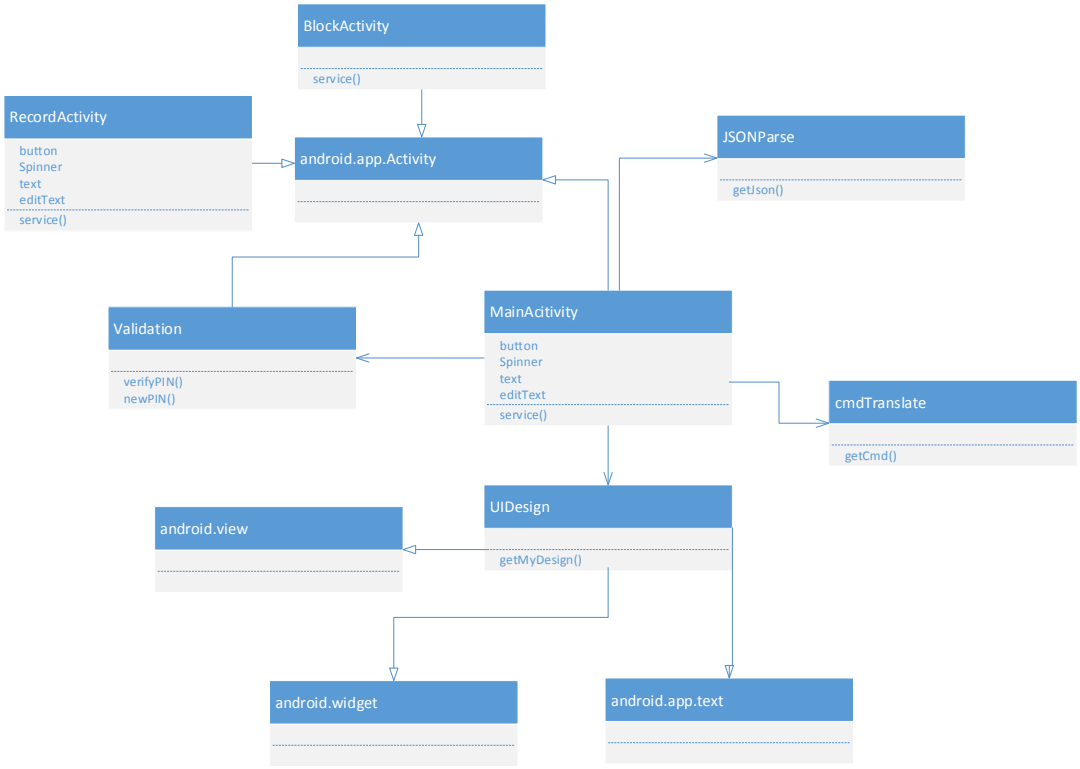

Figure 5.4: Android Application Class Diagram

5.2.1 Utility Classes

The Application class diagram is shown in figure 5.4. And in order to encapsulate commonly re-used functions, following utility classes are employed.

- *CmdTranslate*. This class provides methods such as *getCmd*, *getCmdTlv* and *byteArrayToHexString*, which are used to translate user commands into byte arrays based on rules defined in section 5.1.3.

- *JSONParser*. With support provided by *org.apache.http* and *org.json* packages, *JSONParse* class offers mechanisms to generate Http POST and GET request. Moreover method *makeHttpRequest* in this class returns a Json object including query result as well as return code.

- *UiDesign*. Based on package *android.graphic* and *android.text*, this class provides the approach to realize customized UI layout. For instance, in method *getStyle(String text, int begin, int end, String Farbe)*, the text between *begin* and *end* position is granted with a given color, *Farbe*. A concrete usage case is demonstrated in figure 5.5

Figure 5.5: Customized Text Color for the Command Result

5.2.2 Layout

Under *root\res\layout* directory, the *xml* files that define layout frameworks of my application activities are presented. To be more specifically,

- *welcome.xml*. The UI-layout of welcome screen is described, where user can input his password and username.
- *main.xml*, which defines the layout of application main screen.
- *list_item.xml* and *record.xml*. Those two xml files together picture how to present historical record in a list arranged by date.
- *block.xml*. The PIN block screen is defined by *block.xml*

The screen-shots of my Android application will be presented in next chapter *Implementation Test*.

5.2.3 Activity Classes

Activity class extends *android.app.Activity* and implements interface *SEService.CallBack* provided by Simalliance OpenMobileAPI with whose help one activity is able to send instructions and receive call-back from smart card.

Seek for Android

Seek for Android is a smart card API that implements Simalliance OpenMobileAPI standards which grants Android application the ability to communicate with secure elements such as smart card. Package *org.simalliance.openmobileapi* is the code realization of aforementioned specifications.

In order to communicate with the smart card, the *SEService* constructor must be generated, which provides the method *getReaders* returning a list of available connected secure elements. If one *reader* is selected, then with the help of *openSession* method, one particular session will be created between the selected secure

element and the application. At last this *session* is capable of getting access to a logical channel on the connected secure element by applying *openLogicalChannel* method with AID as input parameter. Following code snippet demonstrates the service and logical channel creation process [sim14].

Service and Logical Channel Creation

```
void performSelectOpt(SEService service) {
Reader[] readers = service.getReaders();
cardreader = readers[0].getName();
boolean isPresent = readers[0].isSecureElementPresent();
Session session = readers[0].openSession();
final byte[] hellosimcard = new byte[] {AID};
logicalChannel = session.openLogicalChannel(hellosimcard);
performOpenOpt(service);
}
```

After a successful creation of the logical channel, one Android application is able to send command to and get response from a secure element using *bytep[] result = logicalChannel.tramsmit(cmdApdu).*

Layout Adoption

Activity class adopts the layout defined in section 5.2.2 during the activity creation phase.

Layout Adoption

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// set Layout
setContentView(R.layout.main);
```

AsyncTask

Since the *HttpResponse* message could be delivered to *activity* which sends the corresponding *HttpRequest* message with latency, therefore for the purpose of simplified UI thread operations, the interface *AsyncTask* is employed in my application.

Following code snippet demonstrates class, that is used to perform the sensor subscription value update operation.

AsyncTask HttpRequest

```
class RecordUpdate extends AsyncTask<String, String, String> {
protected void onPreExecute() {}
protected String doInBackground(String... params) {
runOnUiThread(new Runnable() {
public void run() {
...
```

```
JSONObject json = jsonParser.makeHttpRequest(url_update_record, "GET", params);
...
```

Intent

As already in section 2.6.5 introduced, intent builds data sharing system between activities. Following code segment describes how *ValidationActivity* class perform page jump based on *performVerifyOpt* result.

Intent

```
Intent intent = new Intent();
verifyResult = performVerifyOpt(scservice);
if (verifyResult == 1){
intent.setClass(Validation.this, MainActivity.class);
startActivity(intent);
finish(); }
if (verifyResult == 3){
intent.setClass(Validation.this, BlockActivity.class);
startActivity(intent);
finish(); }
```

Activities Overview

Following four activities together build my Android application.

- *ValidationActivity.* Whose major task is to perform user authentication based on input user-name and password.
- *BlockActivity.* After three times of failed user identification, my application will invoke this *BlockActivity* and accordingly block the PIN.
- *MainActivity.* This class realizes client functions defined in section 4.3.2
- *RecordActivity.* With the help of *RecordActivity*, user is capable of querying and viewing system historical data.

AndroidManifest

AndroidManifest encapsulates all necessary information which is required by Android OS. Such as composing components information, component access permissions and necessary applied libraries [Goo]. In order to provide a secure application environment, in my *AndroidManifest*, only the *ValidationActivity* can be launched by Android system and always runs as the initial task. Moreover, none *intent-filters* are declared for my activities, which means activities from my application can only be started or visited by the explicit intent sent by *ValidationActivity*, the AndroidManifest snippet is shown as below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        android:versionCode="1"
        android:versionName="1.0" package="org.simalliance.openmobileapi.test">
 <application android:label="@string/app_name" android:debuggable="true">
  <activity android:name=".ValidationActivity"
                  android:label="@string/app_name">
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
 </activity>

 <activity android:name=".MainActivity"
        android:label="@string/app_name"
        android:exported='false'>
 </activity>
....
 </application>
</manifest>
```

Figure 5.6: Logo Image Designed for Web Applications

5.3 Smart Home Web Server

Smart Home web application is also designed for the purpose of testing my smart card applet, Android application as well as simulating Smart Home housing devices, especially the data information maintained by them. Apache 2.4.9 together with PHP of version 5.5.12 and MySQL database build my web application.

Figure 5.6 is the logo that I designed for my web application. And Figure 5.7 illustrates the web page that shows state information of three housing devices. Each housing device has its own table in MySQL database. Moreover, user can navigate to other pages by clicking the blue tabs on top right of this page. For instance, by clicking *Record* button, user is able to bowser historical record of each home device. Detailed design information has limited relation with the subject of this thesis, therefor will not be presented.

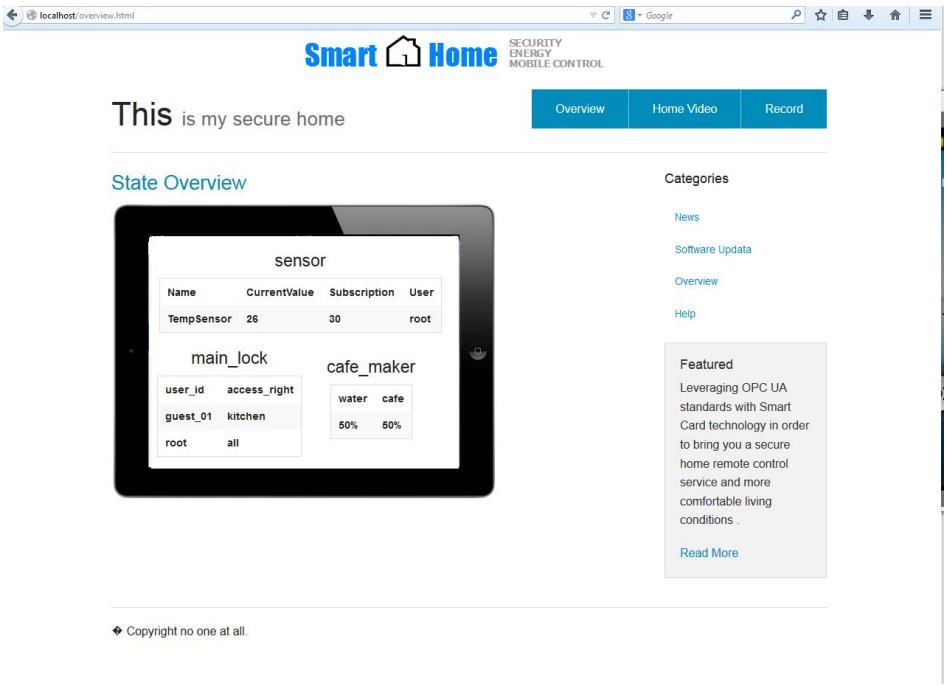

Figure 5.7: Smart Home Web Overview screen-shot

5.4 Conclusion

In this chapter, I have presented how I designed and implemented my different system components with focus on *CommunicationStack* applet and *Smart Home App*. Moreover a brief introduction of my Smart Home web application is also given. In the next chapter, I am going to present how I designed system implementation tests, which are introduced together with the test results for the purpose of proving the reliability and feasibility of my proposal.

6 Implementation Test

In this chapter, a brief description of demonstration testing cases as well as test results is presented. To be more specifically, two test scenarios related with *CommunicationStack* Java Card applet and simulated remote administrator server are designed. Also, the test cases demonstrating communication between Smart Home web server and Android application *Smart Home App* are introduced.

6.1 Javacard Applet and Remote Administrator Server Tests

The aim of this testing scenario is to prove the reliability of applying Smart Card and Globalplatform standardized remote application/file management protocols to provide a secure dual authentication mechanism and messaging platform for other high-level applications. My Java Card Applet - *CommunicationStack* together with correspondingly designed *UTE* test cases are employed and build the testing environment. For reference, *UTE* is introduced in section 4.3.5 together with other testing tools.

6.1.1 UTE Test Cases

Two *UTE* test cases are programmed. They are:

- *Trigger Communication Channel with SMS*. In this test case, I simulate the open communication channel process between smart card and remote administrator server. This process is the cornerstone for a successful remote application management.

- *Secure Messaging*. After a successful identification between communication peers and creation of communication channel, in this test scenario, smart card and remote server are going to perform secure messaging.

Trigger Communication Channel with SMS

In this test suit, *UTE* testing case acts as the remote administrator server and encapsulates information necessary for the construction of a secure Http channel in a *TriggerPushSMS* object and sent this *TriggerPushSMS* to target smart card applet using simulated GPRS connection. The structure of *TriggerPushSMS* is

pictured in figure 6.1 and includes parameters which can be categorized as following:

```
Push SMS:
SPI1: 12 --> [CC] [NO CIPHER] [CNTR IF HIGHER]
SPI2: 01 --> [POR] [POR NO RC/CC/DS] [POR NO CIPHER] [SMS DELIVER]
Ciphering Key Identifier: 15
Signature Key Identifier: 15
Toolkit Application Reference: 380011
Ciphering Key: 0102030405060708
Signature Key: 0102030405060708
Counter: 0000000003
Number of concatenated SMS: 100
Single SMS length: 120
Mactype: 00
------------------------------
ConnectionParameters:
Bearer description: Type=GPRS, Parameter=010101010102
Buffer size=05DCh (1500)
Network access name: 066D6F7270686F03636F6D [morpho.com]
Text: [admin]
Text: [root]
Interface transport level: protocol=TCP, UICC in client mode, port=9096
Other Address: type=IPv4 address, address=80.66.10.152

------------------------------
Security Parameters:
PSK-Identifier: 89390100000129506903
Key Version/Key Identifier: 4001
------------------------------
Retry Policy Parameters:
Retry counter: 0002
Timer value: 00:00:03
------------------------------
HTTP POST Parameters: unknown:
8A1138302E36362E31302E3230303A38303830
8B03303037
8C0B2F4345482F6365686F6370
```

Figure 6.1: Screen-shot of Push Short Message including parameters which are necessary for the communication channel construction

- *Connection Parameters*, which includes network access name, bearer description and other parameters describing the simulated remote server.
- *Smart Card Keysets*, that consist of key identifiers as well as indicator used for the select of cipher keys and signature keys. As described in section 3.3.2, in order to retrieve a particular cipher key from Smart Card key management system, information such as key identifier must be provided.
- *Http connection parameters*. Parameters such as retry counter, timeout values are included in this short message.

After a successful receiving and processing of aforementioned push *TriggerPushSMS*, *CommunicationStack* will preform TLS handshake with the remote server.

The TLS *client-hello* message is demonstrated by figure 6.2. The version of TLS protocol, random number as well as proposed cipher suits are encapsulated. The corresponding TLS *server-hello* message is presented by figure 6.3 and in this demonstration case as shown cipher suit *TLS_PSK_WITH_NULL_SHA256* is chosen. At last, the master key, shown in figure 6.4 is generated between remote server and card applet.

```
Command:    80 14 00 00 0F
Data:       81 03 01 43 01 02 02 82  81 83 01 00 37 01 FF
Le:         00
Info, using regular expression as expected response!
SW1_SW2:    90 00 (NORMAL_ENDING_OF_THE_COMMAND)
--

SEND DATA (send data immediately)
Terminal -> UICC
00 (Command performed successfully)
Channel data length: FFh (255)

struct {
  type = handshake;
  version = TLSv1.2;
  struct {
    type = client_hello;
    struct {
      version = TLSv1.2;
      random =
      struct {
        gmt_unix_time = D0A9E3ED;
        random_bytes = 4085246B0841D0298B6283522F5ABB04B6EE0E4F61EF2E3BE17AA557;
      } Random;
      sessionId = <empty>;
      cipherSuites = {
        TLS_PSK_WITH_AES_128_CBC_SHA256
        TLS_PSK_WITH_NULL_SHA256
      };
      compressionMethods = {
        00
      };
      extensions = {
        struct {
          type = max_fragment_length;
          value = 01
        } Extension;
      };
    } ClientHello;
  } Handshake;
} TLSPlaintext;
```

Figure 6.2: Screen-shot of TLS client-hello-message sent by smart card applet

```
verifyClientHello: TLSv1.2 Ciphersuites Compression Extensions -> passed
struct {
  type = handshake;
  version = TLSv1.2;
  struct {
    type = server_hello;
    struct {
      version = TLSv1.2;
      random =
      struct {
        gmt_unix_time = 49FECD57;
        random_bytes = 15E044CD99324EB34B6B83DD1D634CDB8E41B0EA3F9EA40AA8163996;
      } Random;
      sessionId = C20982D575DE69A01464536EC174FBD61E9DC8F1505CA681901E6BFADC7CB05B;
      cipherSuite = TLS_PSK_WITH_NULL_SHA256;
      compressionMethod = 00;
      extensions = {
        struct {
          type = max_fragment_length;
          value = 01
        } Extension;
      };
    } ServerHello;
  } Handshake;
} TLSPlaintext;
```

Figure 6.3: Screen-shot of TLS server-hello-message sent by remote server

```
verifyClientKeyExchange: PSK Identitiy -> passed
verifyChangeCipherSpec:  Value 0x01 -> passed
verifyFinished: Verify-data -> passed
Master Secret:
struct {
  master_secret = 590407CDA02F0A60F8F56D4D59A0E386C6A25BBCD59530BCB6E86237AA4988F1BC025FE8E076DBBA890768E651879A01;
};
Key Block:
struct {
  client_write_MAC_secret = C4601C7A81F5480E62B4C391EA19BB1BD1346D7B86C109BC1A37CBFB33156175;
  server_write_MAC_secret = 6C6163E7A3A548DE516148D789D70BB5F936A883DF35CDBF9FF699B506265E07;
  client_write_key = ;
  server_write_key = ;
};
```

Figure 6.4: Screen-shot of successful generation of master key between remote server and *CommunicationStack* applet

Secure Messaging Exchanging

Based on the in previous test created secure channel and Http Connection, in this test case, smart card applet and remote server perform secure message exchange using newly generated master key.

```
        cmd = new Blob();

        hr = new HttpResponse();
        hr.setStatusCode(StatusCode.SC_200_OK);
        hr.setHeaderField(HeaderField.X_ADMIN_PROTOCOL, HttpConstants.ADMIN_PROTOCOL);
        hr.setHeaderField(HeaderField.X_ADMIN_NEXT_URI , "/CEH/cehocp");
        hr.setHeaderField(HeaderField.CONTENT_TYPE, HttpConstants.ADMIN_CONTENT_TYPE);
        hr.setHeaderField(HeaderField.TARGET_APPLICATION, hr.targAppAid(aidInstanceRfm)); // D27600002841070101020002B0002601
        cmd.addBlob(ConversionUtil.byteArray("A0250000020206"));
        hr.setContentField(new ContentField(cmd));
```

Figure 6.5: Screen-shot of Http message configuration in test case

The *UTE* test code snippet shown in figure 6.5 generates a Http response message using header defined in section 5.1.2 to the *CommunicationStack* applet, whose AID is set as *D276000028410701010200028B0002601*. The body of this message includes a command APDU, whose content is *A0250000020206*.

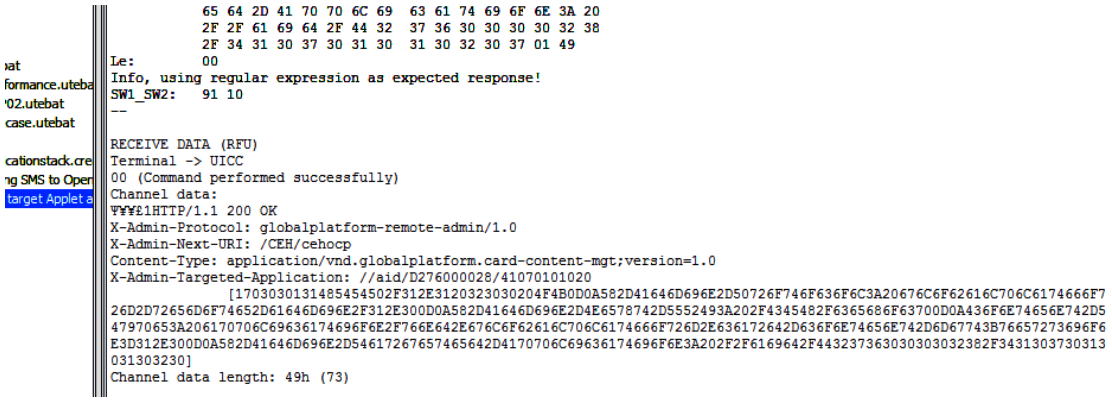

Figure 6.6: Screen-shot of Http message generated by simulated remote administrator server

As illustrated in figure 6.6, the simulated remote server successfully generated this Http response message and transmitted it to the Java Card applet.

After receiving this Http Message from remote server, my applet is able to retrieve the encapsulated command data, to be more precisely, as illustrated in figure 6.7, the command data *A0250000020206* can be retrieved. Furthermore as shown in figure 6.8, at the end of this testing scenario, remote server receives a Http message from *CommunicationStack* applet and the content of this message is apparently secured, whose clear test is presented in figure 6.9. The Http header I applied in those test cases are compliant with the ones introduced in section 5.1.2.

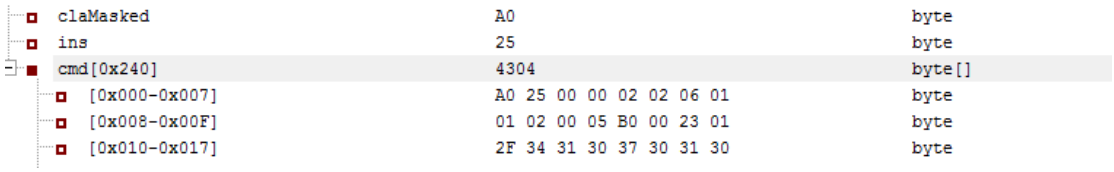

Figure 6.7: Screen-shot of cmd Buffer in *CommunicationStack* applet

6.2 Android Application Tests

In this section, how my Android application interacts with Smart Home web server as well as with smart card applet will be demonstrated.

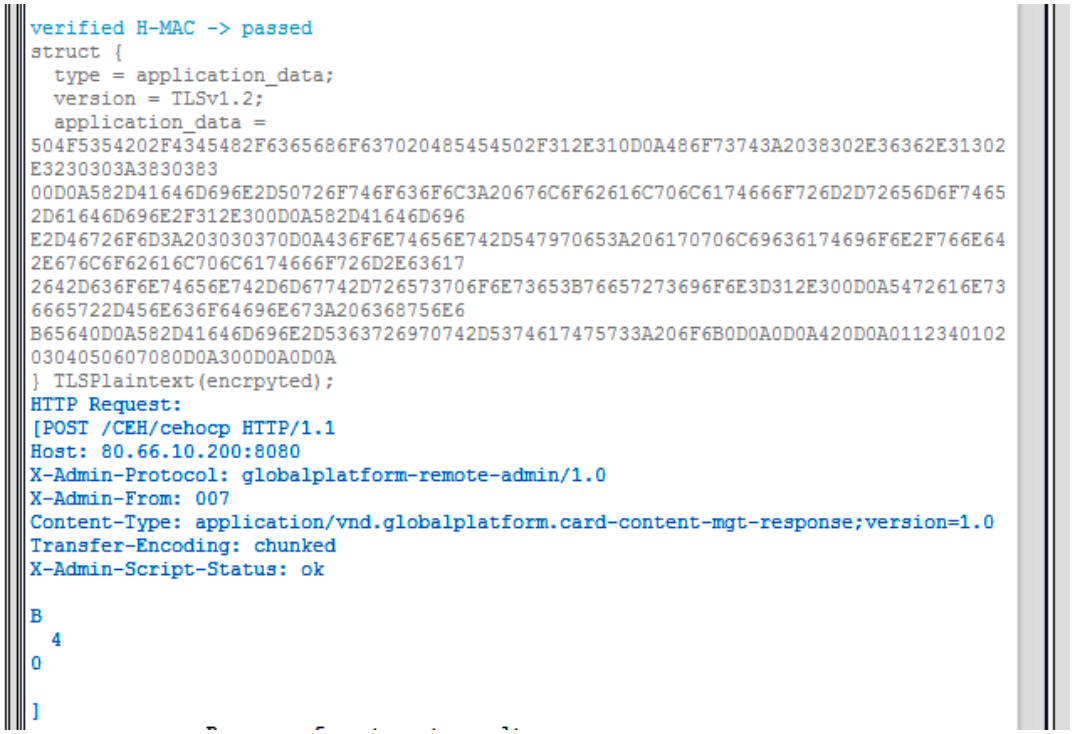

Figure 6.8: Screen-shot of Http message received by remote server

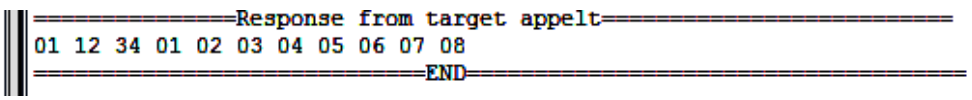

Figure 6.9: Screen-shot of translation of Http content of figure 6.8

6.2.1 PIN Verification

As already introduced, in order to use functionalities provided by *Smart Home App*, phone holder must authenticate himself by inputing PIN for Android application. But after three times wrong PIN input as shown in figure 6.10, PIN will be locked as shown in figure 6.11. Only the Smart Home service provider can unblock the locked PIN.

6.2.2 Communication with Web Server

After a successful identification, Android application user now can enjoy functions provided by Smart Home application, such as remote housing device management and querying historical data, as presented in following.

Moreover, as shown in figure 6.12, house holder can select the name of housing device in the first drop-down menu and choose associating functions from the second drop-down menu. To be more precisely, following housing devices and functions are simulated,

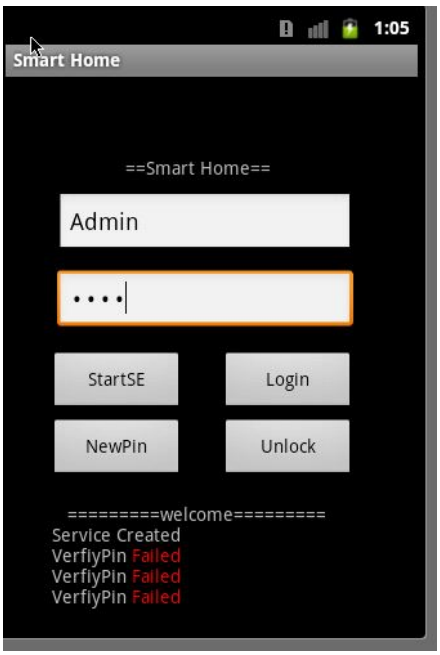

Figure 6.10: Screen-shot of Wrong PIN input

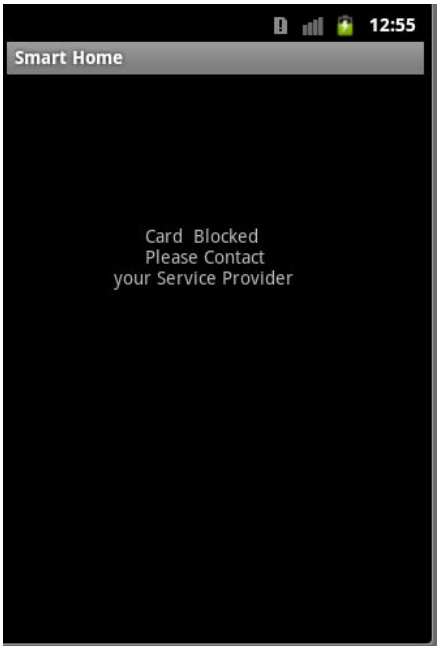

Figure 6.11: Screen-shot of blocked PIN

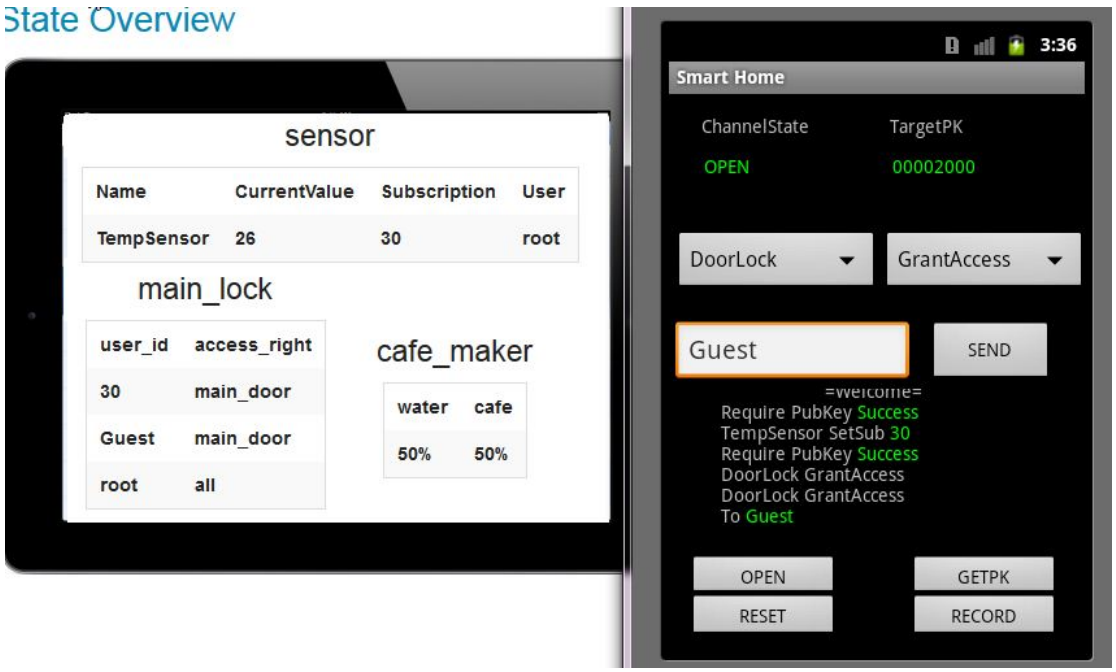

Figure 6.12: Using Android application remotely managing home devices

- Temperature Sensor, which provides functions for getting current sensor value and setting subscription .

- Coffee Maker, which offers add water, add coffee and make coffee services.
- Digital Door Lock, which can receive *openDoor*, *closeDoor* and *grantAccessToOthers* commands.

Meanwhile the four buttons in the bottom of *Smart Home App* main screen as shown in figure 6.12 provide following functionalities,

- *OPEN*, which is used to inform smart card to create secure channel with the remote administrator sever.
- *GETPK*, which is used to query public key of desired housing device selected by the first drop-down menu.
- *RESET*, which is used to reset the current state of this Android application
- *RECORD*, which is used to achieve Smart Home historical records as shown in figure 6.13.

Figure 6.13: Historical Record Overview

6.3 Conclusion

In conclusion, as the testing result shows, my proposal is practical and secure. The demonstration system is able to create secure connection between communication partners. The created communication channel is based on Http connection secured by TLS 1.2 standard. With the help of Morpho Smart Card OS and RAM/RFM standards from GlobalPlatform, my Java Card applet *CommunicationStack* is

capable of performing successful mutual identification as well as secure messaging with the remote administrator server. Moreover, all exchanged messages are well-formed and carefully encrypted. The demonstrated Smart Home takes care of the integrity of exchanged data, confidentiality of user credentials and at the same time provides traceability records.

7 System Security Analysis

In this chapter, I will perform a secure analysis based on the demonstration system. To be more specifically, I am going to introduce a conceptional attack tree graph, with whose help the potential threats and attacks aiming at my system are analyzed. Moreover for each type of threads, I will prove that there exists at least one secure countermeasure in my system.

7.1 Attack Tree Overview

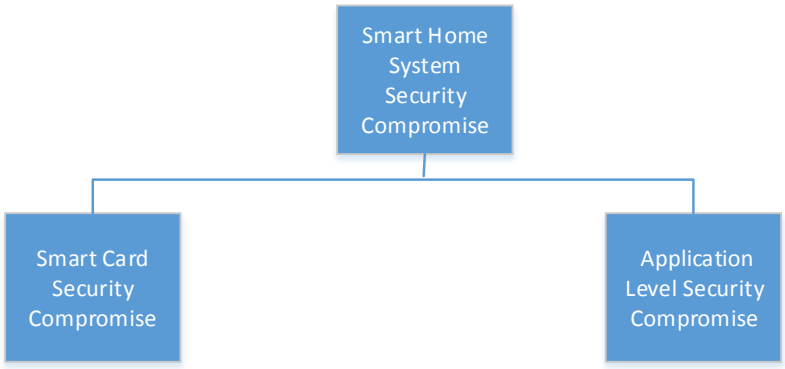

Figure 7.1: Attack Tree Overview

As shown in figure 7.1 the attack tree is categorized into two main sub attack trees. The first type group describes attacks that the third malicious party could apply to harm the smart card employed in my scenario. And the second one contains higher application level related threats, namely the security threats to the OPC UA standards and applications.

7.2 Smart Card Security Compromise

In this section, I will firstly present the attack tree in the domain of smart card and based on this attack tree perform the secure analysis. At last, countermeasures designed and integrated in my system are going to be described.

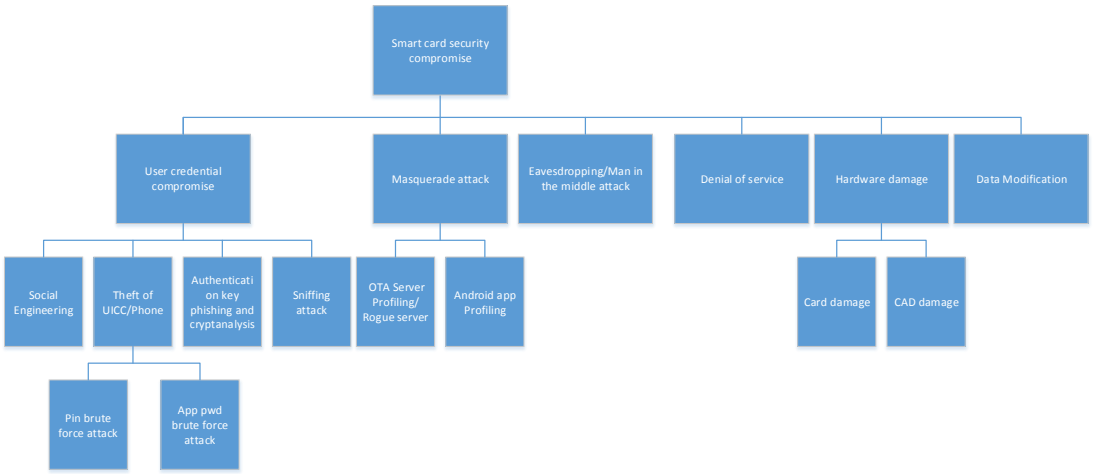

Figure 7.2: Smartcard Security Compromise

7.2.1 Sub Attack Tree and its Content

Figure 7.2 illustrates the first category of potential threats. Since a number of attacks are listed in the figure which makes it hard to read detailed subcategories, therefore this sub attack tree is also presented as following,

1. User credential compromise
   a) Social Engineering
   b) Theft of UICC/Phone
      i. Smart card PIN brute force attack
      ii. Android Application password brute force attack
   c) Authentication key phishing and cryptanalysis
   d) Sniffing attack
2. Masquerade attack
   a) OTA server profiling/Rogue server
   b) Android application profiling
3. Eavesdropping/Man in the middle attack
4. Denial of service
5. Hardware damage
   a) Card damage
   b) CAD damage
6. Data Modification

7.2.2 Analysis and Countermeasures

Security mechanisms and countermeasures against potential threads to the smart card are described as following,

- *against user credential compromise.* Any security system must have a human as the administrator or user, therefore the secure storage and usage of user credential shows its importance in the realm of computer security. As shown in the attack tree, following potential threads are presented,
  - Social engineering, this attack refers to the process of psychological manipulation of user to for instance give out their password or to believe a fraud. In order to protect the end user, no one can get access to my system without the UICC smart card and the corresponding user password, which is used to unlock Android application. In the worst case, which means, user gives his smart phone and password to the attacker, the system is also capable of recording any behaviors taking place during the period of fraud and providing forensic evidence.
  - Theft of phone or smart card. Even when the attacker manages to steal the smart phone, but without the acknowledge of the PIN and user password, which are used to unlock the smart card and Android application respectively, he can not get access to the system. Moreover both smart card and my Android application has the mechanism to block itself after three times of wrong input, which means brute force attack is not an option for the adversary.
  - Key phishing and cryptanalysis/ Sniffing. As introduced in section 3.3.2, smart card is the secure credential storage media and the keys used by smart card consist of various of information, such as key number, key indicator and so on. Consequently, even if the attacker has a segment of credential message, which is captured as a result of key phishing or sniffing, he can never figure out what encryption keys are applied without knowing the corresponding smart card key management system structure.
- *against masquerade attack.* In order to fraud the smart card and to get the credential information stored by it, the attacker could masquerade himself as either the remote administrator server or the Android application designed by me. But any of these two approaches will not work, because the smart card needs to perform TLS secure handshake before exchange any other information with the remote server as standardized by the *GlobalPlatform*. Since fraud server can never pass the secure handshake, therefore the adversary is not able to get any useful information from smart card. Moreover if an Android application claims himself as the *Smart Home App*, it must provide the corresponding application signature, as described in essential secure element control protocols, which are also provided by the *GloabalPlatform*.

Since any fraud application is not capable of providing the correct application signature, this masquerade attack will not succeed.

- *against eavesdropping and MitM attacks.* Since the communication between smart card and the remote server is session-based, even when an attacker is able to capture some communication messages, he still can't reply this message to the victim.

- *against DoS attack..* The remote administrator server could be a victim of denial of service attack, when the adversary floods messages to the server and tries to harm the system. But the remote server applied in my scenario has the message exchanging rate control mechanism, any partner that floods the message will be banned by the system for a period of time.

- *against Hardware damage.* Nowadays' smart cards are designed with the in section 3.3.3 introduced protecting mechanisms against physical attacks, therefor attacks such as power difference analysis will not succeed. The major consequence of smart card hardware damage is that, users have to require a new card from the card provider and to bear the inconvenience brought by card damage.

- *against data modification.* Data here refers not only to the information stored on the smart card but also the messages exchanged between smart card and the remote server. As already introduced, smart card is a perfect storage media and independent of other external device. Moreover the exchanged messages are protected from the attacker by using checksum algorithm, which means any information modified by a third party is going to be ignored.

7.3 OPC UA Application Level Security Compromise

In this section, the second category of threads is discussed and corresponding countermeasures are presented.

7.3.1 Sub Attack Tree and its Content

The second category of threads which focus on the OPC UA application is pictured as figure 7.3 and the graph content is described as following,

1. Message replay attack
2. Man in the middle attack
3. Stealth and Hijacking attack
   a) Session fixation

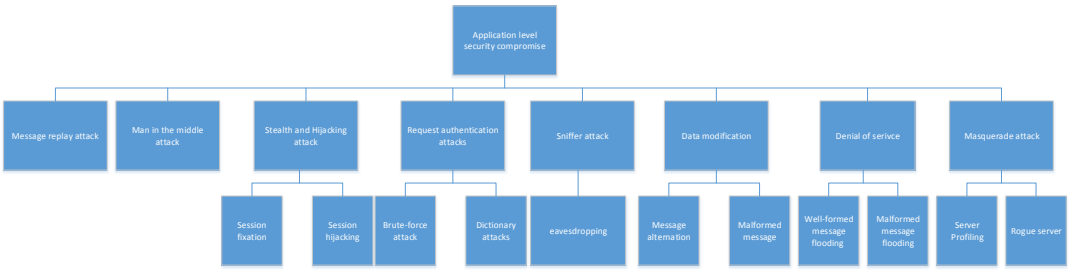

Figure 7.3: OPC UA Application Security Compromise

b) Session hijacking

4. Request authentication attacks
   a) Brute-force attack
   b) Dictionary attacks
5. Sniffing attack - eavesdropping
6. Data modification
   a) Message alternation
   b) Malformed message
7. Denial of service
   a) Well-formed message flooding
   b) Mal-formed message flooding
8. Masquerade attack
   a) Server profiling
   b) Rogues server

7.3.2 Analysis and Countermeasures

In total eight categories of threads, which aim at the OPC UA client and server application code applied in my scenario, are analyzed. They are,

- *Message replay attack and MitM attack.* Even when an adversary captures a message exchanged for instance between housing device and smart phone, he can never successfully reply this message, because each request and response message is protected by its session ID, secure channel ID and alternatively sequence number.

- *Stealth and Hijacking attack.* In order to perform session hijacking attack, the attacker must firstly compromise the secure channel based on which the session is created. But the employed secure channel is standardized by Globalplatform and protected with TLS handshake. Therefore this category of attack is prevented.

- *Request authentication attack.* No matter attacker applies brute force attack or dictionary attack, he only has three password or PIN input chances, which means it is extremely impossible for him to guess the authentication passwords or PIN.

- *Sniffing.* Encryption algorithms are always applied in my system, therefore the adversary is not able to understand the message he sniffed and the system confidentiality is protected.

- *Data modification.* Both symmetric and asymmetric signatures are employed in OPC UA standards, any by third party illegitimately modified message can not pass the signature examination test and will be ignored.

- *Denial of service.* In order to protect the service availability of my system, the message exchanging rate is controlled.

- *Masquerade attack.* Masquerade attacks are impossible in my system for the following reasons. Firstly if a malicious application claims itself to be a OPC UA client or server partner, it must provide necessary secure information such as secure keys and certificates before performing any other actions. Moreover, the adversary also has to be able to perform dual authentication with the remote administrator server. Even when the fraud application manages to receive a message, it doesn't possess the required private keys to decrypt the content.

7.4 Conclusion

In conclusion, my proposal system is strong against the known attacks presented in the above pictured attack tree by adopting the combination of the magic bullet - smart card and the newly released OPC UA specifications.

8 Summary and Future Work

With the development of M2M and home automatic control technologies, the in this thesis described Smart Home system will be in popular demand in the near future. But since present-day system is frequently exposed in pervasive computing and ubiquitous networking environment, the concept of security is drawing more and more attentions.

The security feature of smart card as secure token makes this chip sized circuit card the perfect storage media for user credential information. Meanwhile smart card OS also provides security mechanisms such as key management and secure APDU messaging protocol, which contributes to add an additional secure protection to the system that employs smart card. Moreover by applying well standardized industrial protocols such as RAM/RFM over Http standards from GlobalPlatform and the Simalliances OpenMobileAPI, smart card is capable of constructing secure session based connections with remote server and perform dual authentication as well secure messaging with CADs.

The newly released OPC UA specifications provide not only a sophisticated object modelling rules, which can be applied to design domain specific models, but also offer a complex secure policies that have clear security objectives in each lay of system architecture and help to protect OPC UA application from adversary's attacks. The in this thesis proposed combination of smart card with OPC UA protocols aims to create a generic communication and connectivity platform for various vendors manufactured secure devices with strong security supports and consideration.

In the future, if the study of my proposal goes further, it should be also considered that how to apply the newly proposed LTE technology to benefit the system users with qualified high bandwidth network. Moreover it is also worth thinking, whether it is practical and how to apply Near Field Communication embedded devices in the Smart Home system, for the purpose of providing enhanced secure communication environment, integrating small payment services and etc.

At last, I am grateful for the chance provided by Morpho card Paderborn to write this thesis. Especially I would like to thank Dr. Simon Oberthür, Dr. Stefan Sauer and Castern Rust, who have given me professional advices for my thesis.

Bibliography

[Ahm11] Ahmadou A. SERE, Julien Iguchi-Cartigny, Jean-Louis Lanet: *Evaluation of Countermeasures Against Fault Attacks on Smart Cards.* April,2011

[And11] Andrew Hoog: *Android Forensics: Investigation, Analysis and Mobile Security for Google Android.* July 2011

[And13] Andreas Frejborg, Martti Ojala, Otso Palonen, Jouni Aro: *OPC UA Connects your Systems.* 2013

[A.T13] A.T. Kearney: *Mobile Economy 2013.* 2013

[Avi09] Avik Chaudhuri: *Language-Based Security on Android.* 2009

[Bar13] Bartlomiej Uscilowski: *Security Response: Mobile Adware and Malware Analysis.* October,2013

[Bru99] Bruce Schneier, Adam Shostack: *Breaking Up Is Hard To Do: Modeling Security Threats for Smart Cards.* October,1999

[CHA] CHAN, Siu-cheung Charles: *An Overview of Smart Card Security*

[Dav10] David Ehringer: *The Dalvik virtual machine architecture.* March 2010

[Den] Dennis A. Seidel: *MQTT vs OPC-UA: Das HTTP der Industrie 4.0?* `http://dennisseidel.de/the-http-for-industrie-4-0-mqtt-vs-opc-ua-german`, . – [update;September,2013]

[Dia03] Diane J. Cook, Michael Youngblood, Edwin O. Heierman, Karthik Gopalratnam, Sira Rao, Andrey Litvin, Farhan Khawaja: *MavHome: An Agent-Based Smart Home.* 2003

[Dim02] Dimitar Valtchev, Ivailo Frankov: *Service Gateway Architecture for a Smart Home.* April 2002

[Eri11] Erika Chin, Adrienne Porte Felt, Kata Greenwood, David Wagner: *Analyzing Inter-Application Communication in Android.* June,2011

[Fra11] Franke Feinbube from HPI University Postdam: *Android*. November 2011

[Glo11] GlobalPlatform: *GlobalPlattform Card Specification*. January 2011

[Glo12] GlobalPlatform Card: *Remote Application Management over HTTP Card Specification v2.2 Amendment B*. March 2012

[Goo] Google: *API Guides*. `http://developer.android.com/guide/topics/manifest/manifest-intro.html`,

[Han12] Hannah Gommerstadt, Devon Long: *Android Application Security*. 2012

[Hoo11] Hoon ko and Ronnie D. Caytiles: *A Review of Smartcard Security Issues*. June,2011

[Jah13] Jahanzaib Imtiaz, Jrgen Jasperneite: *Scalability of OPC-UA Down to the Chip Level Enables Ïnternet of Things¨*. July 2013

[Jea09] Jean-Louis Carrara,Herv Ganem,Jean-Franois Rubon,Jacques Seif: *The role of the UICC in Long Term Evolution all IP networks*. January 2009

[Joh02] John Abbot: *Smart Card: How Secure Are They?* March, 2002

[Kei13] Keith Makan, Scott Alexander-Bown: *Android Security Cookbook*. December,2013

[Li 04] Li Jiang, Da-you Liu, Bo Yang: *SMART HOME RESEARCH*. August 2004

[Mar11] Marko Gargenta: *Learning Android*. March 2011

[Mic10] Michael Angelo A. Pedrasa, Ted D. Spooner ,Iain F. MacGill: *Coordinated Scheduling of Residential Distributed Energy Resource to Optimize Smart Home Energy Services*. September 2010

[Nim] Nimmi Kalinati: *A Study of Smart Card and its Security Features*

[OPC09a] OPC Foundation: *OPC Unified Architecture Specification Part2 Security Model 1.01*. February 6.2009

[OPC09b] OPC Foundation: *OPC Unified Architecture Specification Part3 Address Space Model 1.01*. February 6.2009

[OPC09c] OPC Foundation: *OPC Unified Architecture Specification Part4 Services 1.01*. February 6.2009

[OPC12] OPC Foundation: *OPC Unified Architecture Specification Part1 Overview and Concepts 1.02.* July 10.2012

[Pau04] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan and Srivaths Ravi: *Security as a New Dimension in Embedded System Design.* 2004

[Phi06] Philip Keerseblick, Wim Vahnoucke: *Smart card (In-)Security.* May,2006

[Rak14] Rakesh Kumar Jangid, Uma Choudhary, Swapnesh Taterh: *Security Measurement in Secure Smart Cards.* May,2014

[RSA99] RSA Laboratories: *PKCS #15 v1.1: Cryptographic Token Information Syntax Standard.* December 1999

[Seb11] Sebastian Lehnhoff, Wolfgang Mahnke, Sebastian Rohjans, Mathias Uslar: *IEC 61850 based OPC UA Communication - The Future of Smart Grid Automation.* August 2011

[Sib08] Sibylle Meyer, Eva Schulze: *Smart Home fr ltere Menschen.* February 2008

[sim14] simalliance: *Open Mobile API specification.* February 2014

[Ste] Stefan-Helmut Leitner, Wolfgang Mahnke, Ragnar Schierholz: *Secure Communication in industrial automation by Applying OPC UA*

[Sun98] Sun Microsystems: *Java Card Applet Developers' Guide.* July 1998

[Vin] Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delhoche, Christophe Loge: *The Smart Home Concept: our immediate future*

[Vin02] Vincent Rialle, Florence Duchene, Norbert Noury, Lionel Bajolle, Jacques Demongeot: *Health 'Smart' Home: Information Technology for Patients at Home.* Number 2002

[Wik] Wikipedia contributors: *Constrained Application Protocol.* `http://en.wikipedia.org/wiki/Constrained_Application_Protocol`, . – [update;February 8,2014]

[Wol08] Wolfgang Rankl und Wolfgang Effing: *Handbuch der Chipkarten - 5. deutsche Auflage.* 2008

[Wol10] Wolfgang Rankl and Wolfgang Effing: *Smart Card Handbook Fourth Edition.* 2010

[Zhi00] Zhiqun Chen: *Java Card Technology for Smart Card.* June 2000