

**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
Heinz Nixdorf Institut und Institut für Informatik
Fachgebiet Softwaretechnik
Warburger Straße 100
33098 Paderborn

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

Master's Thesis

Submitted to the Software Engineering Research Group
in Partial Fulfillment of the Requirements for the
Degree of
Master of Science

by
YUANKUI WANG
Dessauer Str. 4
33106 Paderborn

Thesis Supervisor:
Dr.rer.nat 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	Solution Idea	2
1.3	Thesis Structure	2
2	Fundamentals	5
2.1	OPC Unified Architecture Structure 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	12
2.2.3	Smart Card Software Components	13
2.2.4	Smart card File Management	14
2.2.5	Data Exchange with Smart Card	15
2.2.6	Secure Messaging	17
2.2.7	Life Cycle	17
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	19
2.3.4	Java Card Applet	20
2.3.5	Javacard Cryptography	20
2.4	GlobalPlatform and Remote Application/File Management	23
2.4.1	OPEN - GlobalPlatform Environment	23
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	27
2.6.4	Android Application Overview	29
2.6.5	Intent	30
3	State of Art	31
3.1	Smart Home Research	31
3.1.1	Overview	31
3.1.2	Smart Home Optimized for Energy Services	32
3.1.3	Smart Health Home	33
3.1.4	Agent-based Smart Home	33
3.1.5	Conclusion	35
3.2	OPC UA Application and Security policy	35
3.2.1	OPC UA applications	35
3.2.2	OPC UA secure policies	36
3.2.3	OPC UA design consideration	37
3.2.4	Conclusion	37
3.3	Smart Card Security	38
3.3.1	Smart Card in Access Control	38
3.3.2	Key Management	39
3.3.3	Threaten and countermeasures	41
3.3.4	Smart Card Secure Applications	44
3.3.5	Conclusion	45
3.4	Secure Android Design	45
3.4.1	Android Security Mechanism	45
3.4.2	Security Design	46
3.4.3	Conclusion	49
3.5	Embedded System Secure Design	49
3.5.1	Overview	49
3.5.2	Embedded System Secure Requirements	49
3.5.3	Software Security during Software Design Life Cycle	50
3.5.4	Secure Architectures	51
4	Implementation Scenario	53
4.1	Overview	53
4.2	Component Structure	54
4.3	Communication Flow	55
4.3.1	CommunicationStack Applet	56
4.3.2	OPC UA appliation Functionalities	57
4.3.3	Cell Phone Side Software Structure	57
4.3.4	Housing Device Software Structure	58
4.3.5	Implementation Tool Support	59

5 System Design	63
5.1 UICC Applet	64
5.1.1 Overview	64
5.1.2 Work Flow	64
5.1.3 Commands: Interface between Applet and CAD	67
5.1.4 Classes	71
5.2 Android Application	74
5.2.1 Utility Classes	75
5.2.2 Layout	75
5.2.3 Activity Classes	76
5.3 Smart Home Web Server	79
6 Implementation Test	81
6.1 Remote Management Testing	81
6.1.1 <i>UTE</i> Test Case	81
6.2 Android Application Testing	84
6.2.1 PIN Verification	85
6.2.2 Communication with Web server	85
6.3 Conclusion	85
7 System Security Analysis	87
7.1 Attack Tree Overview	87
7.2 Smart Card Security Compromise	87
7.2.1 Sub Attack Tree and its Content	88
7.2.2 Analysis and Countermeasures	89
7.3 Application Level Security Compromise	90
7.3.1 Sub Attack Tree and its Content	90
7.3.2 Analysis and Countermeasures	91
7.4 conclusion	92
8 Summary and Future Work	93
8.1 Summary	93
8.2 Future Work	93
Bibliography	95

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]	18
2.12	Javacard Applet Execution States[Wol08]	20
2.13	APDU command processing[Wol08]	21
2.14	Card Operation System Architecture[Glo11]	22
2.15	OpenMobileAPI architecture overview[sim14]	24
2.16	Android Stack Overview [Mar11]	26
2.17	Compiling process difference[Mar11]	28
2.18	Comparison between Java byte code and Dalvik executable file[Dav10]	29
3.1	DER Scheduler in Smart Home optimized for energy[Mic10] . . .	32
3.2	Overview of Smart Health Home structure[Vin02] (RCC: Remote Control Center)	33
3.3	MavHome agent architecture[Dia03]	34
3.4	keys presented in Smart Card Description Language from <i>Morpho</i>	39
3.5	the PSK key used to configure administration session	40
3.6	Differential Power Attack on a DES implementation[Pau04] . . .	43
3.7	Malware growth in period from August 2010 to October 2011 based on database from paper[Yaj11]	46
3.8	Detailed Android permission model[Han12]	47
3.9	Permission required by application <i>k9mail</i> [Han12]	48
3.10	Software Security Consideration in Software Design Life Cycle[Pau04]	50
3.11	Architectural design space for a secure information processing[Pau04]	51
4.1	Smart Home	53
4.2	Smart Home System Component Structure	54
4.3	OPC UA Client Server Structure Example	55
4.4	Client Structure	58

4.5	Communication Flow between an AP and corresponding APSD[Glo12]	59
4.6	Server Structure	60
5.1	Message Exchange	63
5.2	Handshake and message exchange between SD and Remote Server	65
5.3	Class Diagram	71
5.4	Android Application Class Diagram	74
5.5	Customized text color	75
5.6	Logo image used in web applications	78
5.7	Smart Home Web Overview	79
6.1	Push Short Message including all parameters which are necessary for the creation of communication channel	82
6.2	TLS client hello sent by smart card applet	83
6.3	Successful generation of master key between remote server and <i>CommunicationStack</i> applet	83
6.4	<i>cmd</i> Buffer in <i>CommunicationStack</i> applet.	84
6.5	Http Response Message received by remote server	84
6.6	Http Content Translation	84
6.7	Wrong PIN input	85
6.8	PIN is blocked	85
6.9	Using Android application remotely managing home devices . . .	86
6.10	Historical Record Overview	86
7.1	Attack tree overview	87
7.2	Smartcard Security Compromise	88
7.3	Application Level Security Compromise	91

List of Tables

2.1	OPC UA specifications	6
2.2	ISO/IEC 7816[Wol08]	13
2.3	Command APDU Structure	16
2.4	Command APDU Structure	17
2.5	package: <i>javacardx.crypto</i>	22
2.6	package: <i>javacard.security</i>	23
2.7	Minimum Android Recommendations[Dav10]	27
3.1	key data stored in smart card[Wol10]	40
5.1	SELECT command APDU	67
5.2	SELECT response APDU	67
5.3	Verify PIN command	68
5.4	Verify PIN response APDU	68
5.5	Reset PIN command	68
5.6	Reset PIN response APDU	68
5.7	Communication session creation command APDUs	69
5.8	Trigger Session Return Code	69
5.9	Close Session command APDU	69
5.10	Close Session Return Code	69
5.11	Process data command APDUs	69
5.12	Process data Return Code	70
5.13	Command data Type-Length-Value structure	70

1 Introduction

Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the most recent released industry standards set from OPC Foundation, which compared with his 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 and government identification, payment. Therefore it is meaningful and promising to develop OPC UA standard compliant application on embedded smart card secure device, for the purpose of secure remote control, enterprise resource planning and etc. Since the storage and compute capacity of a chip card is limited, I propose that the aforementioned application system should consist of two essential parts. Namely, the client/server application code, which is realized as for instance Android Application. And the communication stack, which is designed as a Javacard Applet applying Remote Application Management protocol from GlobalPlatform. The implemented demonstration scenarios and corresponding analysis in this thesis proof the feasibility of my proposal.

1.1 Motivation

According to the *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 mobile technology. Moreover by year 2017, 700 million new subscribers are expected to be added. The number of mobile subscriber will reach 4 billion in 2018. 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 of housing devices, exist not only in science fiction film but also could be realized using present-day's technology.

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 convicting exam-

ple here is the well known Machine to Machine communication technology, that is also referred as M2M technology. In M2M communication, machines are capable of exchanging gathered data with each other using wireless or wired networks, together accomplishing complex tasks, making wise scheduler decisions and etc. M2M technology is widely employed in different industry spheres such as factory automation, remote access control and device monitoring. It boosts the efficiency of corresponding processes, offers centralized service support and data management, minimizes system response time.

But in order to enjoy the aforementioned features, two tough issues must be resolved. First, how to achieve a common interface for the devices that build the system. And second how to guarantee system security under different communication environments with various data complexity and customer's requirement.

1.2 Solution Idea

In this thesis, I am going to address solutions for above mentioned questions and design a Smart Home system for the purpose of 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 housing devices, like 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 communication management with other devices.

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 a OPC UA specification compliant communication stack Applet on the UICC smart card, whose duties are: creation and management communication between OPC client/server application, entity authentication and secure message exchange.

1.3 Thesis Structure

In the second chapter I will present the fundamental technologies which will be frequently mentioned in this paper. Second 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 com-

ponents. To be more precisely, I will discuss how I design the Javacard Applet and the Android Application. 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 proof 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
- Smart card Technologies
- Javacard Language and applet
- GlobalPlatform and Remote Application/File Management protocol
- OpenMobileAPI
- Android Platform and Application

2.1 OPC Unified Architecture Structure Overview

Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the most recent 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 automation system.

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 provided by a machine as well as data information gathered by that device. 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.

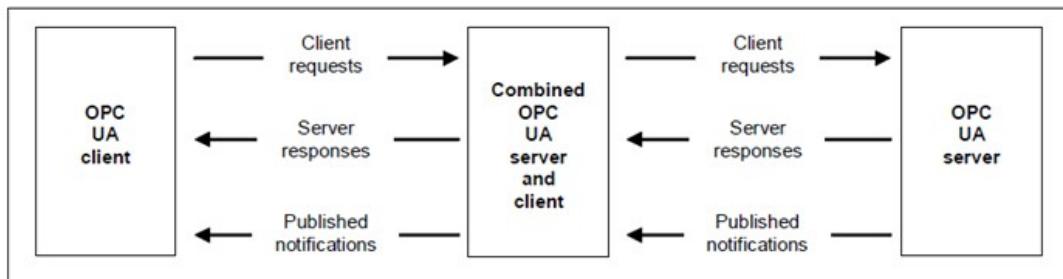


Figure 2.1: OPC UA Client Server Structure[OPC12]

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 subscription.

2.1.3 OPC UA Terminologies

In OPC Unified Architecture, the on server stored information, that can be visited 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 system parameters and etc.

Clients can query and accept information provided by OPC Unified Architecture 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 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 is normally predefined and customized by a client, to publish data to the 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.

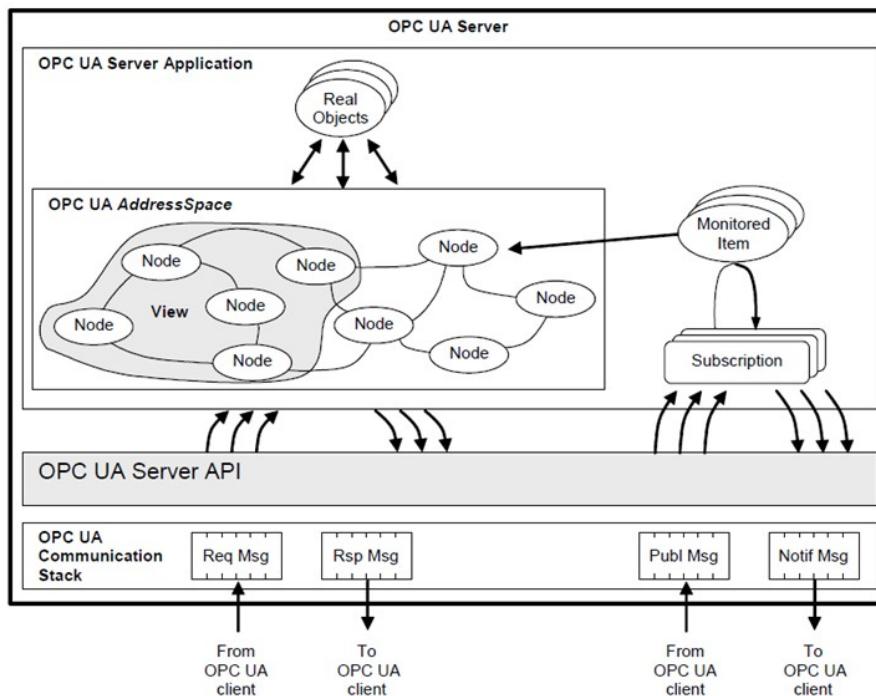


Figure 2.2: OPC UA Server Structure[OPC12]

2.1.4 Standard OPC UA Server

In figure 2.2, the structure of one standard OPC UA Server is described. It includes three main parts, server application, internal API and communication stack. In server application part, functionalities and services which are offered by OPC UA standards are realized, such as *Event Notification*, processing request from connected OPC UA client, data encryption and decryption. Moreover, *Real objects* here refers physical field devices and software applications that are maintained and managed by OPC UA server. *Nodes* in *Address Space* presents abstractly above mentioned *Real Objects*. *View*, which is pictured as a part of address space, presents *Objects* that can be browsed by particular clients. The main task for communication stack is to establish communication session based on secure channel between OPC UA client and server. Typically communication messages which are exchanged frequently among clients and servers are, request-, notification- message from client and corresponding response-, publish message from server. At last, an internal API connects the server application and the communication stack.

2.1.5 Standard OPC UA Client

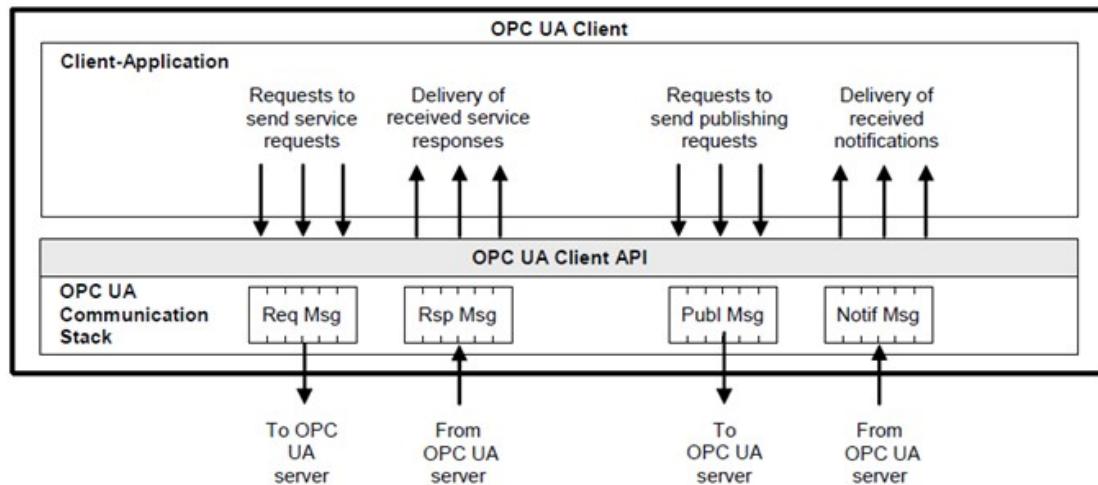


Figure 2.3: OPC UA Client Structure[OPC12]

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.

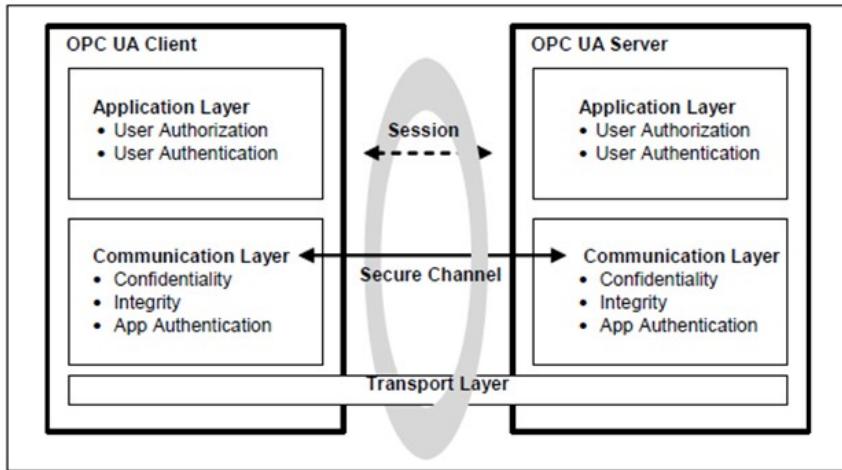


Figure 2.4: OPC UA Client Server Communication[OPC09a]

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 model*, with whose help the developer is able to configure the security level of the application to meet the need of reality. In the *security model*, peer authentication as well as authorization, data integrity and confidentiality, auditability(also known as traceability) and availability of services are guaranteed. Also OPC UA standard provides a set of countermeasures against attacks such as message flooding, eavesdropping, message spoofing, message alteration, message reply, server profiling, session hijacking and so on[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 confidentiality, message integrity and application authentication. On top of secure channel, is an application level session between OPC UA client and server, whose responsibility is to transmit data information and commands. It should be pointed out that, even a secure channel is out of work for some reasons, the session is still valid and OPC UA client and server involved in aforementioned session can still re-establish the broken secure channel. A secure transport layer is guaranteed by encryption and signatures methods provided by platform that supports OPC UA structure.

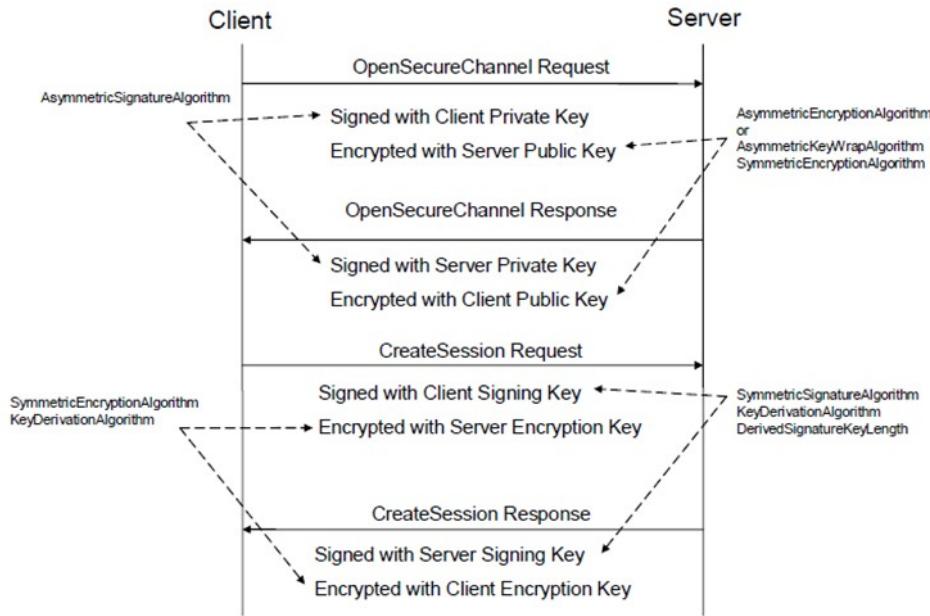


Figure 2.5: OPC UA Client Server Security Handshake[OPC09a]

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 the response from server. Messages exchanged during the secure channel construction phase are encrypted using asymmetric encryption and signature algorithms. But some security protocols that could be applied according to OPC UA standards, are not using an asymmetric message encryption algorithm to encrypt to request/response messages. Instead, they apply *AsymmetricKeyWrapAlgorithm* to encrypt symmetric keys and use symmetric encryption algorithm with encrypted keys to encrypt messages. After a successful construction of secure channel, OPC UA client sends *CreateSession* request and waits for the server response. Messages transported during this procedure are encrypted with symmetric encryption algorithms and signed with client/server signing key.

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: application layer, communication layer and transport layer. Even the terminologies of those layers use the same English words as the ones applied in ISO model, but they are not equal to the layers in ISO model. Figure 2.6 also pictures a precise functionality overview of each layers. To be more precisely, from top to bottom, the functionality of each

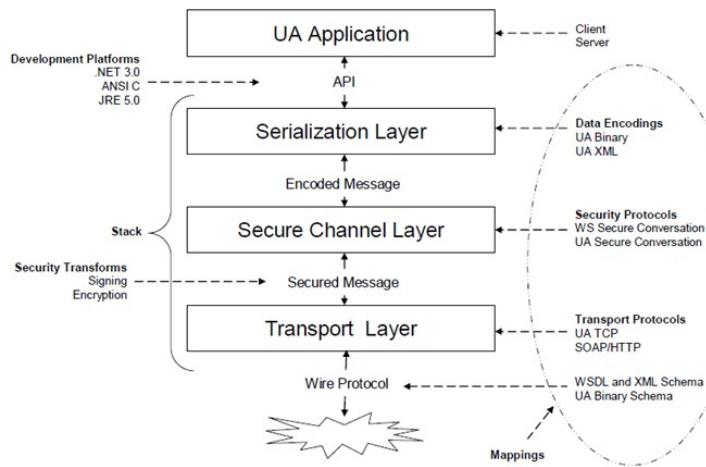


Figure 2.6: OPC UA Client Server Communication Stack[OPC09a]

layer is,

- Application Layer. Client or server application code is realized in this layer.
- Communication Layer. Serialization layer together with secure channel layer build the communication layer and their jobs are, dividing long message into pieces referred as message chunk, encrypting each individual message chunk and forwarding encrypted message chunks to the transport layer. When receiving message chunks from other, 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 a API.
- Transport Layer. The transport layer provides mechanism for message transmitting.

2.1.9 Other Competitor

WebSphere Message Broker Message Queuing Telemetry Transport (MQTT)[SID13] is another machine to machine (M2M) communication protocol. Compared with OPC UA standard, MQTT also supports UDP protocol in the transport layer. In OPC UA, only unidirectional, client to server, communication is provided, but in MQTT server to client communication is also possible without server implements client code. Moreover the communication overhead of MQTT is 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 model and supports more features including historical data record, alarm, notification as well as complete security policies. This is the reason why 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 devices 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.

2.2 Smart card

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 process, 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 signature along with other precious user information can only be accessed through a serial interface, which stands under strict control of the card operation system. This characteristic provides strong protection against unauthorized data access and ensures the confidentiality of on card stored information. Therefore smart cards are widely used in applications that require strong protection.

With sophisticated communication protocol using Application Protocol Data Units(APDU), smart card and Card Accepting Device(CAD) are able to process secure message exchange. Smart card is also able to process cryptographic algorithms on 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 speed performance is better.

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

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, which stores and protects

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 for application identifiers
ISO 7816-6	Interindustry data elements

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 for application identifiers
ISO 7816-6	Interindustry data elements

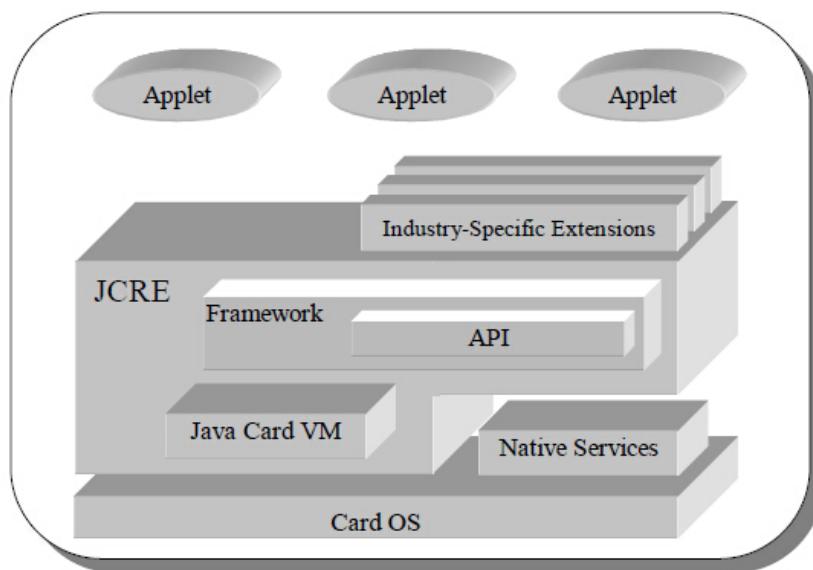


Figure 2.7: Smart Card Software Components[Sun98]

subscriber's confidential information. Moreover, as a 32bit 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.

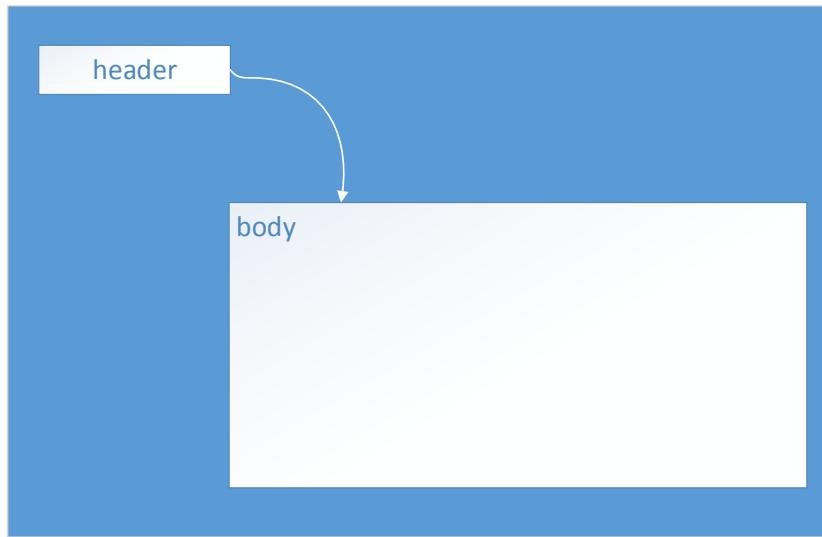


Figure 2.8: the internal structure of a file in smart card file management system[Wol08]

File Structure

As pictured in figure 2.8, files on smart card consist of two parts, the header, which encapsulates administrative information such as, file structure and access conditions. The body, that stores real user data and is linked with 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.

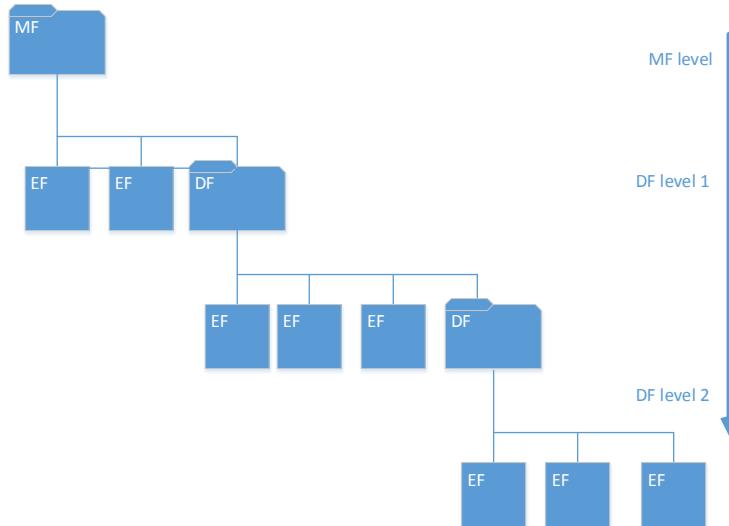


Figure 2.9: File Tree on Smart Card[Wol08]

PKCS#15

As already discussed, smart card not only is used as a storage media but also can 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 from PKCS#15 interpreter.

2.2.5 Data Exchange with Smart Card

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.

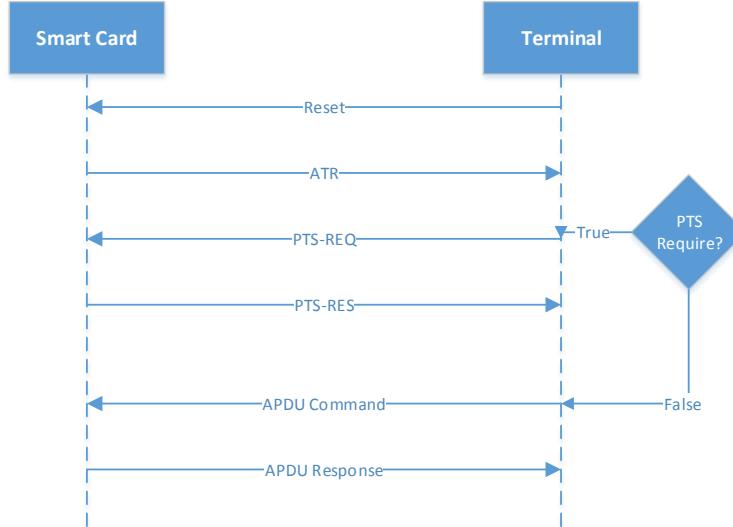


Figure 2.10: Communication between Smart card and CAD[Wol08]

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 a 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 Accessing device. Its structures abide 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 used to provide more command information	mandatory
P2	parameter 2 used to provide more command information	mandatory
Lc field	the length of data received by card	mandatory
data field	data sent to card	optional
Le field	the length of data sent by card	optional

Table 2.4: Command APDU Structure

data field	length decided by Le of preceding command APDU	mandatory
SW1	state word 1 also called return code 1	mandatory
SW2	state 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, attacker 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 secure message mechanism, both message sender and receiver must agree on the to be applied message cryptographic algorithms and corresponding pre-shared keys. In telecommunication domain in accordance with ISO/IEC 7816-4[Wol08], TLV(Type-lengthl-valuse)-formed data, which encapsulates relative user information, is used to perform secure messaging as 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 is taken by the card manufacturer. One unique *fabrication key*, *KF* for short, is applied in order to keep smart card from unauthorized modification.
- *Pre-personalization Phase.* After fabrication phase, smart card chip is going to be integrated on plastic frame by card suppliers. Moreover *fabrication key*, which is derived from *manufacturer key*, in phase one, is 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 fraudulent changes, *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 card issuer finishes this phase, he will add an *utilization lock* and the life cycle of smart card will come to next phase.
- *Utilization Phase.* Card holder will be able to use his smart card in this phase.
- *Invalidation Phase.* Card operation system will block the smart card in this phase. To be more specifically, operation such as writing and update will be disabled. There are two reasons that why a smart card lands this

phase. Reason one is because both *smart card PIN* and *smart card unblock PIN* reach its limited try counts, which indicates someone is trying to force brute guessing PIN. The other reason is, a particular application adds a *invalidation lock* on *master file* of smart card.

Conclusion

From the above described smart card life cycle, it is not hard to find out that from the every begin of smart card manufacturing and deliver process, card manufacturers and issuers have taken it into account, that a secure card production, utilization and delivery must be guaranteed.

2.3 Java Card

Java Card technology not only adopts the distinguishing features from Java, such as productivity, security and portability[Sun98], it also makes Java technology available on smart card, where programmer must be faced with more harsh conditions, like limited memory resource and computing ability.

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 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 applet (.cap file) is generated by off-

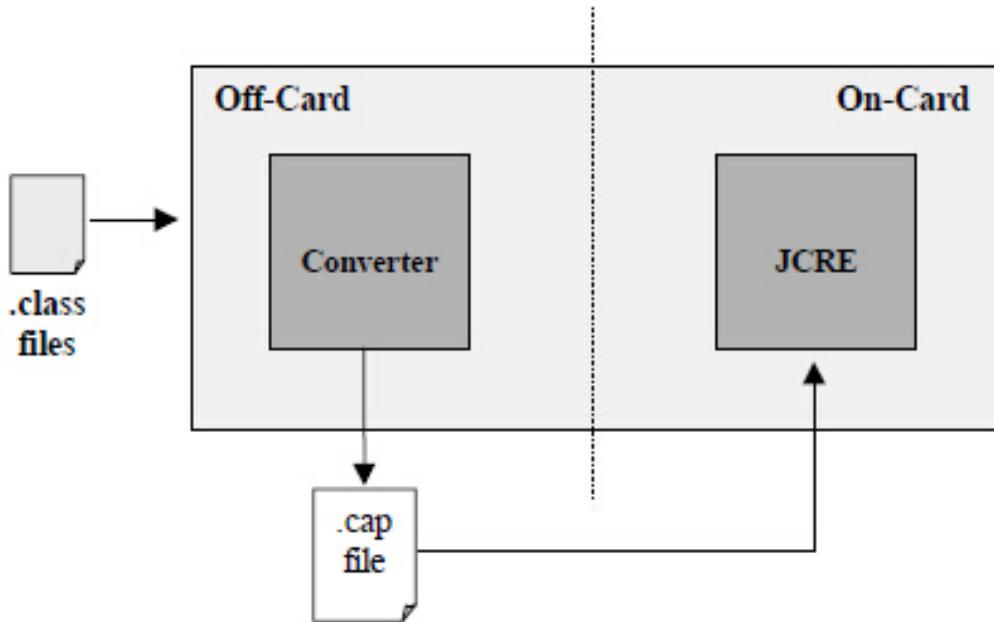


Figure 2.11: Java Card VM Components[Sun98]

card Converter based on inputed Javacard class file and executed by on-card Java Card Runtime Environment (JCRE).

2.3.1 Language Specification

Apart from above mentioned Java Card VM, compared with original Java language, Java card has many unique features. Since current smart card does not support multitasking, therefore threads are not backed up in Java Card. Also garbage collection is performed by VM, as a result function *finalize()* is not supported. Moreover, since on smart card, the memory space and processing ability 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 features, for example, private access modifiers as well as bytecode verification[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 ongoing transactions' integrity is protected with the help of following schema[Wol08]:

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

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 that involved in this broken transaction.

2.3.3 Persistent Object and Transient Object

In the realm of Java Card, all objects are preserved in nonvolatile memory, which means this persistent object exists on smart card 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 on Java card. To be precisely, for instance class array object stays in nonvolatile memory space but in contrast one actual 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 Java Card Applet should implement following four methods:

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

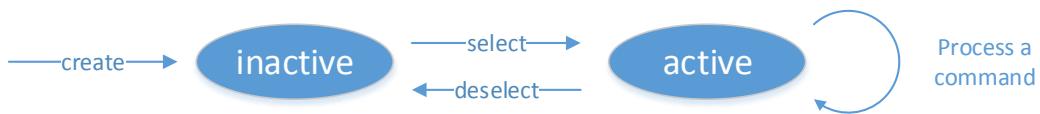


Figure 2.12: Javacard Applet Execution States[Wol08]

After finishing programming one Applet, it comes to next phase, namely applet installation. This process takes place usually at the factory or office under the control of card issuer. When one applet is installed on smart card, it only directly communicates with JCRC and other installed applet classes. It should 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 JCRC to *register* and *select* corresponding applet at run time.

Figure 2.12 pictures execution states transaction of Javacard applet. Furthermore figure 2.13 illustrates how JCRC selects and deselects one applet based on input APDU commands.

2.3.5 Javacard Cryptography

Javacard provides a list of APIs to support not only smart card application and data exchange security but also to reinforce other system's security by acting as

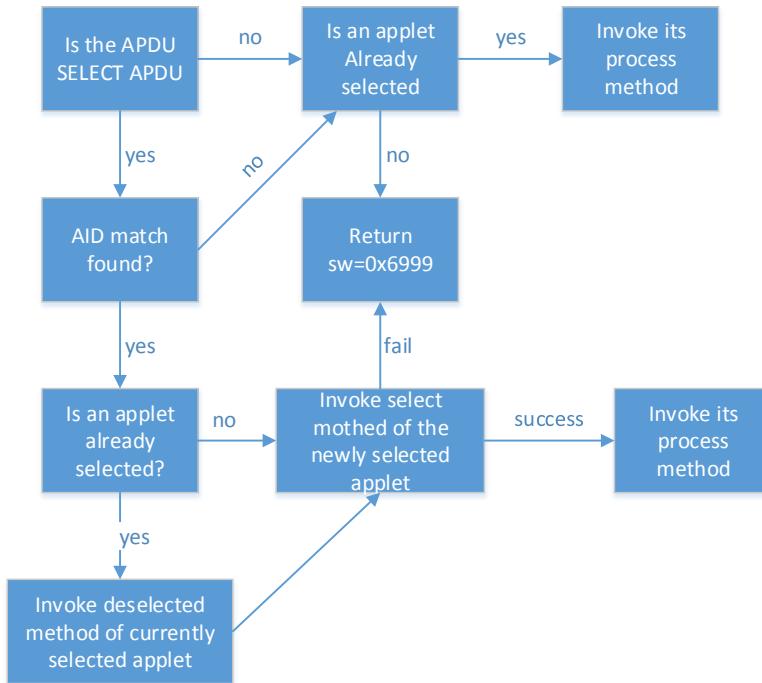


Figure 2.13: APDU command processing[Wol08]

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 guarantee the authorities of both communication partners.
- *Message Confidentiality*: The transferred information is encrypted using 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 based on the to be transferred data and integrated in that message.

Packages *javacard.security* and *javacardx.crypto* support aforementioned security mechanism with following class 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

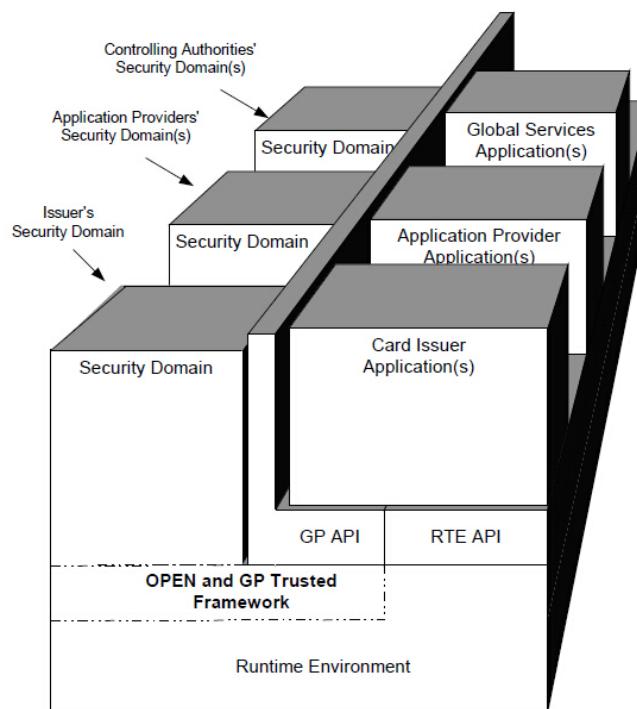


Figure 2.14: Card Operation System Architecture[Glo11]

Table 2.6: package: *javacard.security*

Class or Interface	Function Description
Key	Interface for all keys
SecretKey	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
RSA privateKey	Interface for keys used by RSA algorithm to sign data
RSA publicKey	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
CryptoException	Exception class

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 used as industry standard for managing Java Applet based applications on Javacard Operation System in several domains, for instance in communication industries and payment company[Glo11].

As shown in figure 2.14, the GlobalPlatform card architecture contains four essential parts. The runtime environment, that provides hardware-neutral API for card application and manages card memory spaces. The on card installed applications, which offers customers various functionalities and services. The security domain (SD), that is usually associated with particular application and known as on-card representatives of off-card authorities. 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 is

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 message regard of one application. Moreover the special logic channel named basic channel is always opened. In order to ensure system security, OPEN supports secure mechanisms such as, user authentication , resource availability guarantee and secure channel protocol .

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

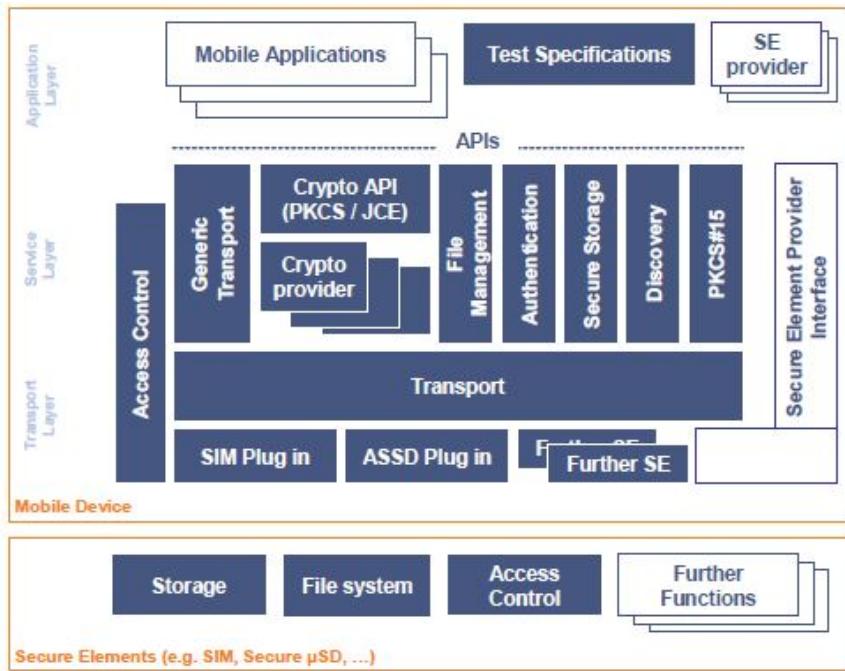


Figure 2.15: OpenMobileAPI architecture overview [sim14]

2.5 OpenMobileAPI

OpenMobileAPI, provided by SIMalliance, constructs an interface between terminal and chip card, which can be used by on terminal installed mobile application to access recourse stored on smart card as well as call function provided by smart card applet. Moreover OpenMobileAPI offers security mechanisms such as access control, that can be applied for mutual authentication between application and

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

- *Transport Layer*: This layer is in charge of providing secure elements access control services using APDUs and acts as cornerstone for other two layers.
- *Service Layer*: Abstract interfaces, that provide various functions such as secure storage, cryptographic services, are offered by this layer.
- *Application Layer*: Mobile applications which benefit from OpenMobileAPI lie in this layer.

2.6 Android

2.6.1 Overview

Android is an open source platform based on Linux and modified by Google, which is 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 and android developers are able to deploy their androids system 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 his customers, such as:

- *Security*: Linux, as the cornerstone of android, has been 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 more comfortable user experiences and better 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 provide its end users 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.

2.6.2 Android Software Stack

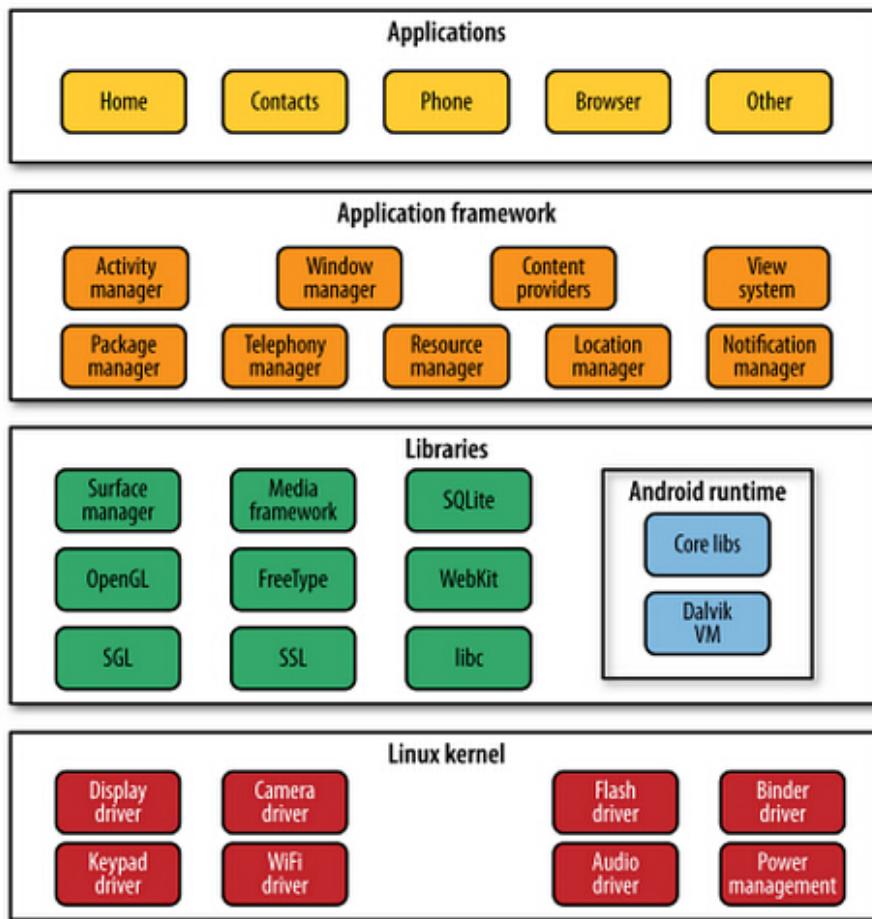


Figure 2.16: Android Stack Overview [Mar11]

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

Linux kernel:

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

Libraries:

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 specific designed virtual machine, namely Dalvik[Mar11] is deployed. There exists two reasons for not using standard Java VM[Mar11]. First of all, since Java VM is generally developed virtual machine, therefore some constraints from

mobile systems and devices are not concerned, such as *processing gap* and *battery gap*[Pau04]. Processing and battery gap refer the limited process capabilities and battery lifetime provided by mobile device. Secondly, when 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 the license free and mobile platform specific virtual machine, Dalvik.

Moreover other libraries which provide services to application framework layers are also included in this Libraries layer, 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-rendering engine.

Application framework

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

Application

As the top layer of entire android software stack, application layer with the help of both native and third party reside component from application framework layer, provides various services to 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 described above, Dalvik VM compared with general Java virtual machine, takes the constraints that are specific about mobile device into account, which means this android virtual machine concerns the 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.

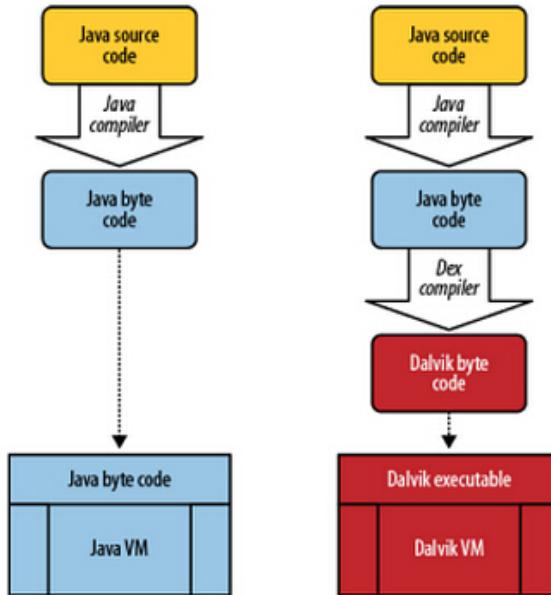


Figure 2.17: Compiling process difference[Mar11]

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* 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 colored blue in left side of above-mentioned figure, is heterogeneous, which means all kinds of constant pools are included in this heterogeneous constant pool, even if they could 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.

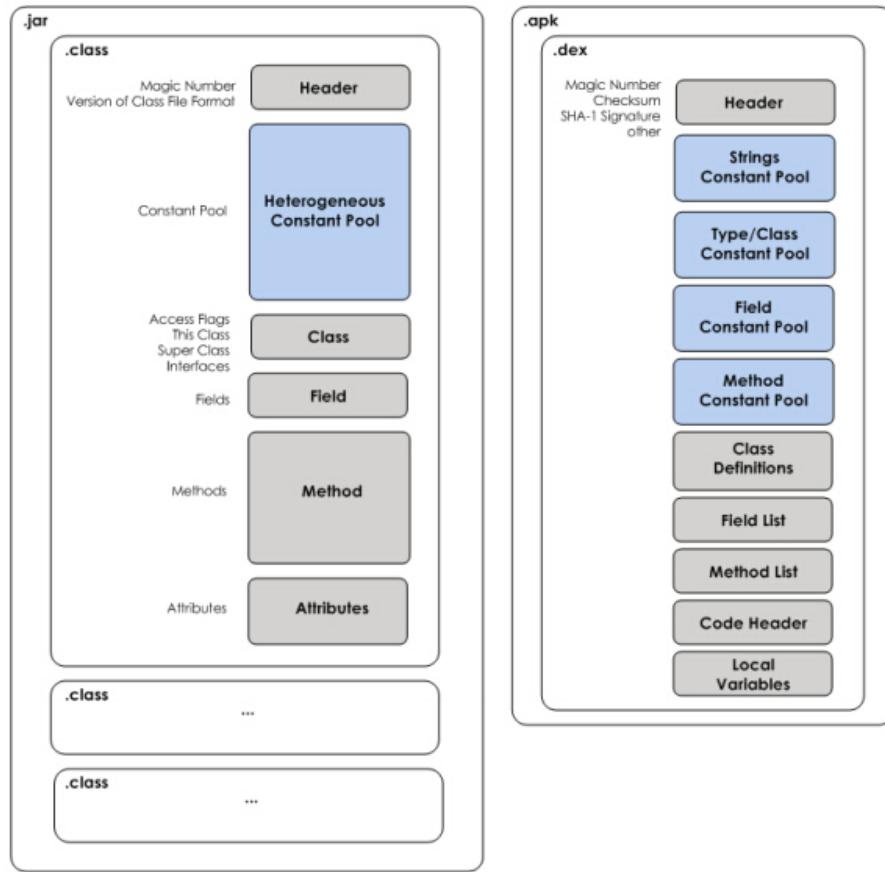


Figure 2.18: Comparison between Java byte code and Dalvik executable file[Dav10]

2.6.4 Android Application Overview

One Android application consist of four components, they are activities, services, content providers and broadcast receivers[Han12]. Let's look into them a little deeper.

- *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 specific 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 various applications.
- *Broadcast Receivers* The last component is broadcast receiver, who is in charge of asynchronously communication with android system broadcast and

other applications. For example, when the battery life runs almost out, broadcast receiver will then be informed by the android OS.

2.6.5 Intent

For the purpose of inter- and intra- application communications, 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 understood as a well-formed message format that builds the Android message passing system[Eri11].

3 State of Art

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

The second topic covered by this chapter is the study of security. In more details, I have researched security technologies and mechanisms employed in OPC UA standards, Smart card system and Android Application design. With the acknowledgment of aforementioned studies, I have learned the potential threads to my application scenario and the corresponding countermeasure. 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 scenario system, especially with the consideration that some of system components could be just simple low level filed devices which have limited resources and power but still have security demand.

3.1 Smart Home Research

3.1.1 Overview

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

With the development of Smart Home technology, nowadays' Smart Home is not only in charge of monitoring and controlling lighting and heating inside the building, but also capable of connecting almost every electronic devices, prediction of inhabitant behaviors as well as making scheduler decisions. Functionali- ties 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 put 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 suggestion based on various parameters such as distributed energy resources(DER), with the prospect of energy and resource saving.

Decision Support Tool

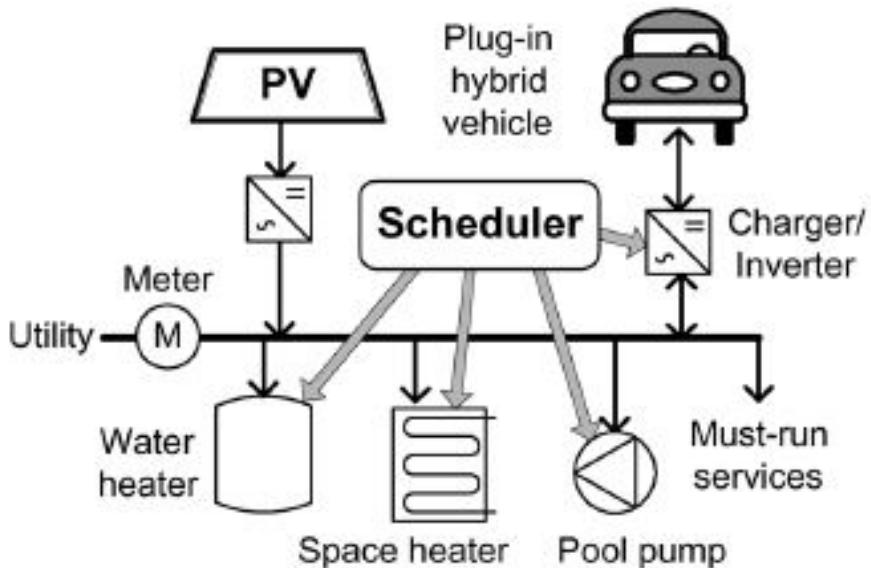


Figure 3.1: DER Scheduler in Smart Home optimized for energy[Mic10]

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 for 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 [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 by[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 i th 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 at home or elder residents.

The charming features of this Smart Home system are the combination of telemedical system with communication technologies and customizing services such as, teleconsulting, telediagnosis, real time imaging as well as distance medical education[Vin02], which together 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 guideline for the design of Smart Health Home system and summarizes precious experience.

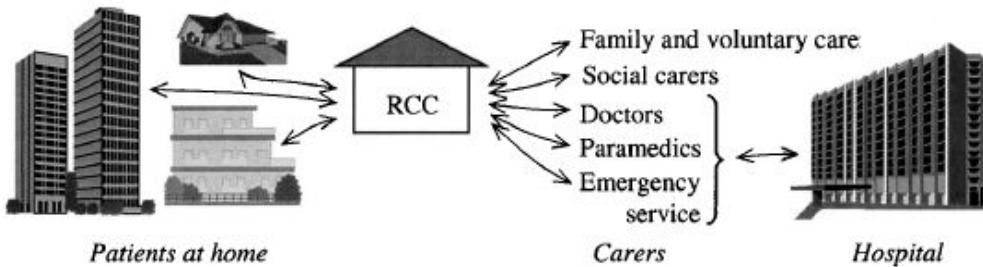


Figure 3.2: Overview of Smart Health Home structure[Vin02] (RCC: Remote Control Center)

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, that

can predict householders' behaviors, make appropriate decisions. The achieved prediction can help residents to experience more comfortable and convenience living condition. MavHome (Managing An Intelligent Versatile Home)[Dia03] builds the best example.

MavHome architecture

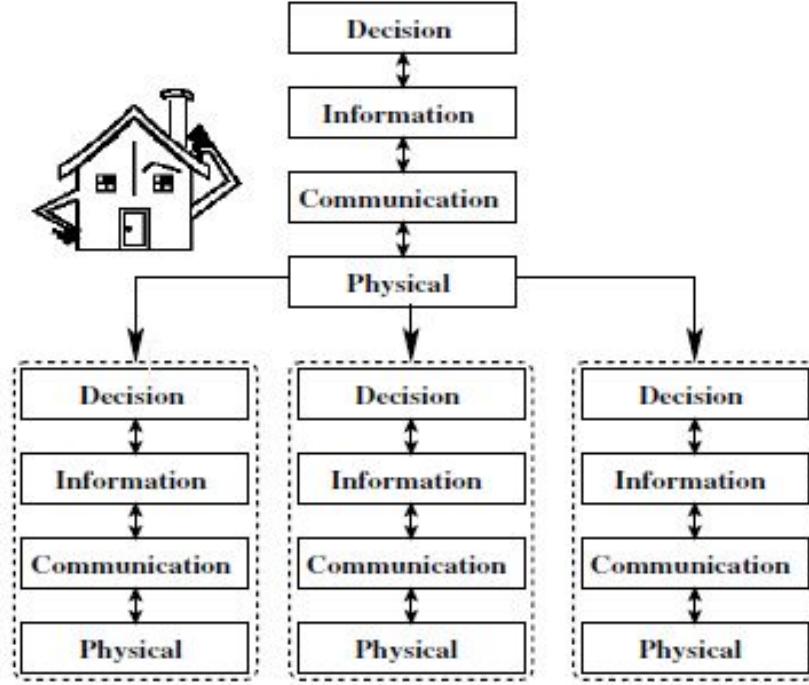


Figure 3.3: MavHome agent architecture[Dia03]

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

- *Physical layer*, where hardwares that together build Smart Home system are deployed. Moreover, underlaying agents can also act as physical layer for other agents.
- *Communication layer*, this layer provides communication services for agent by using functionalities offered by physical layer.
- *Information layer*, as higher layer of Smart Home system, the responsibilities for this layers are gathering and maintaining information which is used by decision layer.
- *Decision layer*, in this layer agents make decision as well as learn resident's preference and correct the unwanted system behavior.

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

Also described above, Smart Home is the representative application which is based on industrial automation, remote control, 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 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 idea of this thesis applying OPC UA standards with smart card technology came into world.

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, secure concerned, IEC standard compliant, promising industry standards set, which provides a service oriented architecture and is being widely applied in industry application such as critical control system and industrial automation. Application examples are given in following paragraphs, each of which has different focuses. To be more specifically, the charming demonstrated characteristics of OPU UA standard set are: secure consideration, real time data exchange and management, scalability.

OPC UA and ICS

With the develop of industrial automation, Industrial control system (ICS) has became a hot topic, but most manufactures putted much more effort in designing automation and manufacturing process and neglected the communication security

issues. As a consequence, Cyber security of ICS now is drawing more and more attentions from industry. OPC UA offers manufactures who are applying ICS system not only 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 communication protocol stack layer.

OPC UA and Smart Grid

Smart Grid is now concerned as the future of electricity energy industry and therefore how to achieve an intelligent electricity distribution as well as behavior coordination between customs and suppliers is a hot topic[Seb11]. Electricity industry is searching out for a standardized communication technology to solve aforementioned issues. OPC UA standards set that includes *alarm and even* model, *data access* model as well as *historical data access* model, is without doubt one of the best candidates. With the help of all these 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 policies

At the beginning of design phase, the OPC foundation has taken the construction of secure and robust communication protocol as the center and therefore developed a enhanced consistent security model which has clear and definite objectives in each layer of OPC UA transport and communication stack. In 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 introduced, in order to process mutual authentication among OPC UA applications from different logic levels and hardware devices.

OPC foundation uses the term *secure policies*[OPC09a] to describe the user authentication, user authorization and application authentication 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 provides distinguishing services to different clients, who might have various demands for security. Meanwhile one

client can also accept a list of profiles, each of them is assigned by one server with particular security objectives.

3.2.3 OPC UA design consideration

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

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

- *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 caused by unexpected physical device failure or intentionally server denial of service attack initiated by third party and etc.
- *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 sever. Alternatively server can also ban or block the client for a period of time, who is 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 robust random number generation functions.
- *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. And 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 proof the feasibility of applying OPC UA as communication protocol for Smart Home proposed in this thesis and the study of application cases help me to learn the this point better.

3.3 Smart Card Security

Integrated circuit cards, ICCs for short, was firstly designed and patented in Germany[Joh02]. Since then this credit size-, embedded with circuit chip- card is being applied in more and more various industry domains, such as secure 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 market. This *contactless smart card* is able to communication with card access 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 temper resistant nature always contributes to make them to be one of the most popular secure token medias.

3.3.1 Smart Card in Access Control

In present days, *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 concerned with our daily life. Especially when electronic devices, which can carry sensitive and 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 early between two identification peers.
- 2 *possession of an object*, this object could be anything that is predefined. One example could be a key that can open a particular door.
- 3 *biometrics identification*. First two points have their own limitations, because third part could steal the object, that is used for identification, or copy user's password. Moreover it also can happen, that a password is too complex to remember or identification object is hard to carry with. Compared with them, biometrics identification depends on however the human body characteristics, such as fingerprint, signature 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 the a chip card there usually exit a signature signed by the card holder or a card holder photo, that could be used to perform biometrics identification.

Nowadays with the help of standards such as *GPC_2.2_B_RAM_over_HTTP*, which is introduced in section *fundamental technologies*, 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 kept from unauthorized modification. After 3 times wrong PIN input, a smart card will be locked until the *unlock PIN* operation is successfully carried out. Moreover if *unlock PIN* operation also fails three times in a roll, the PIN will never be unblocked[CHA].

3.3.2 Key Management

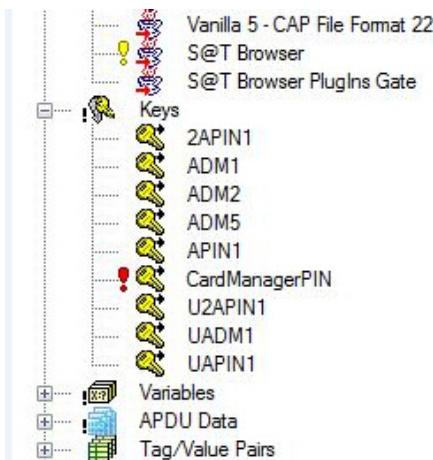


Figure 3.4: keys presented in Smart Card Description Language from *Morpho*

The above introduce PIN is not the only key that are stored on 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 one *Morpho* smart card product, a list of keys and key sets are used, for example, Card Manager PIN and Administrative keys (*ADM1*, *ADM2*, *ADM5*).

Moreover key data in smart card not only simply record the actual key value but also must provide information such as key version number, the usage of key and so on. Table 3.1 describes a typical key data stored on smart card. Likely the

3. STATE OF ART

usage of a key also needs more information than just input the actual key value as shown in figure 3.5

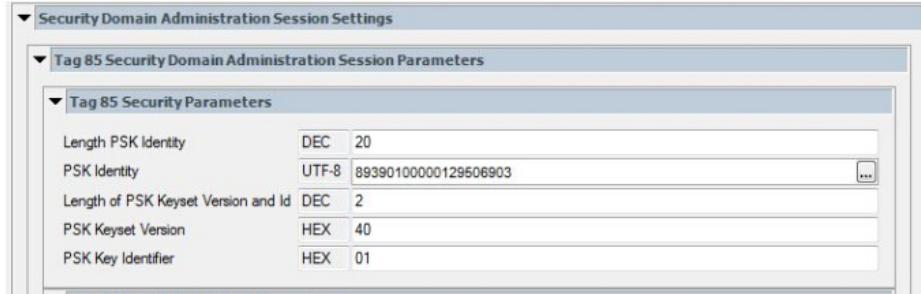


Figure 3.5: the PSK key used to configure administration session

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* shows its importance. The core responsibilities of key management is 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. 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 secret.

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 smart card. As a consequence, keys used in smart card are derived from unique card number, which is given during the card production process, with the help of triple DES and AES cryptographic algorithms[Wol10].

Dynamic Key Dynamic key is normally applied to secure communication between peers and therefore it is also known as session key or temporary key. The generation of dynamic keys begins with generation of a random number that is proposed by one communication partner or unique data which is able to specify one particular session. The rest process may be different and let's take *ANSI X 9.17* standard as an example[Wol10].

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

This process described above is one-way process and can not be reversed. To be more specifically, T_i and Key_{gen} are time-, 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 Threaten and countermeasures

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

- *Cardholder* In other word, the owner of smart card, who as customer daily uses smart card for different purpose 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 park fee.
- *Terminal* The card access 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 phone company. 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.

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[Hoo11]:

- Logical attacks. Whenever we develop a software, there could exit 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 some command 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*. Ill-designed smart card software, that attempts to break smart card system and steal information.
- Physical attacks. This type of attack aims to reverse engineer the 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]. For instance:
 - *Micro-probe station*. Using Micro-probes, attackers 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 for some ROM memories and with a step further analyze 0 and 1 in those memories[Hoo11].
- 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

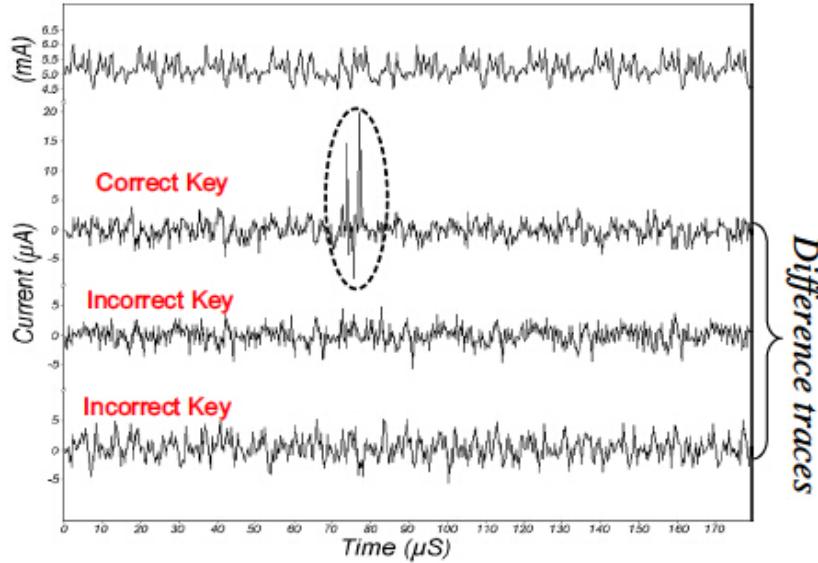


Figure 3.6: Differential Power Attack on a DES implementation[Pau04]

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 runtime depends also on the input key. Therefore there is an opportunity for attacker using observed time gap in key computation to get secret key information[Joh02]
- Other Attacks[Rak14].
 - Denial of service. This kind of attack aims at interrupting normal smart card operation and consuming smart card system resources to impact the smart card service availability.
 - Eavesdropping. Through eavesdropping exchanged information is captured and analyzed.
 - Cover transaction. Fake session or session hijacking tries to forge a communication session between smart card and the other connection peer, in order to gain information related with peer authentication, authorization and message exchanged between two partners.

Countermeasures

Logical Attacks Countermeasure In order to prevent this category of attacks, the number of bugs in smart card application should be minimized. Software and card manufacturers can apply *structure design* by dividing application into small

3. STATE OF ART

units which are easy to be tested and understood. Alternatively various company related in smart card industry should work together to propose standardized interface and protocols to regulate the 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 150nm, 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 manufactures are now able to build chip card consist of multi layers. vulnerable Components, for instance ROM memory, are built in 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 attack are not able to get correct timing gap anymore[Phi06] .
- Against Power Consumption Attacks by applying digital noise. Alternatively chips 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 for 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, therefore more and more smart cards are applied in secure applications and products. For instance:

- In Payment System. Recently the concept electronic purses and electronic payment systems become 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, smart card is now widely used as secure token in various computer secure domains and adds an additional protection layers on the system that 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 to 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 aimed at smart phone platforms such as the most popular Android is reported. Figure 3.7 shows a dramatic increasing amount of Malwares that are targeted at Android from 2010 to 2011[Yaj11]. Especially when taken into account that present day's 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 developer to design an android software with large security consideration.

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.

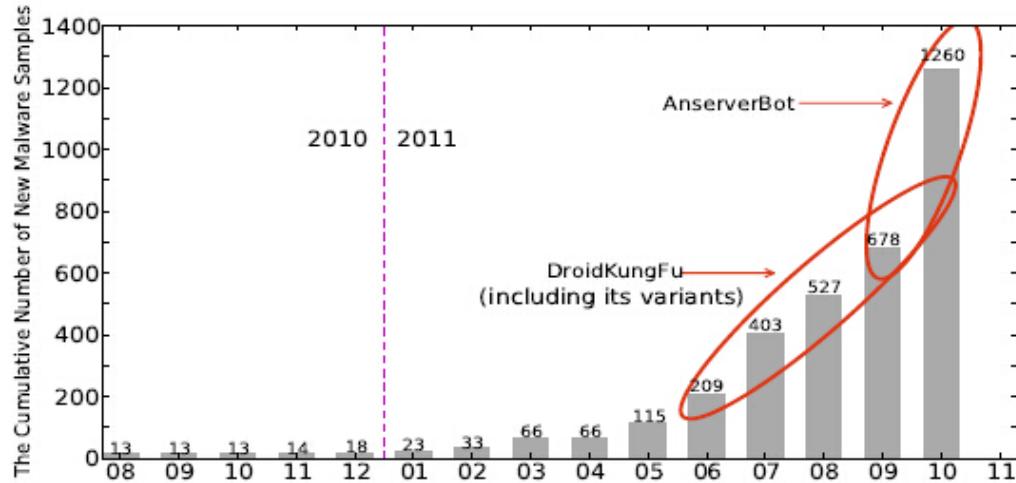


Figure 3.7: Malware growth in period from August 2010 to October 2011 based on database from paper[Yaj11]

- *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 protected from the others[Avi09].
- *Access Permissions* Permissions are given during the applications' installation time. As illustrated in figure 3.8, Android permission system restricts application's access right to private data of end user as well as of other applications, to smart phone hardwares and to Android OS.
- *Authorized Signatures* Applications are signed with certificates and the according private keys are protected by application developer. Every time when one application claims his identity, his signatures will be checked, in order to stop potential masquerade attacks[Avi09].

3.4.2 Security Design

With the acknowledgment form previous subsection, we can conclude 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, which has none explicit target, this implicit intent can be intercepted by a Malware. As a result, this malicious application may manage to access to sensitive data and perform attacks such as *activity hijacking* and *service hijacking*[Eri11].

Therefore it is necessary for application developers to apply protection techniques in order to protect their application from attacker and reverse engineering.

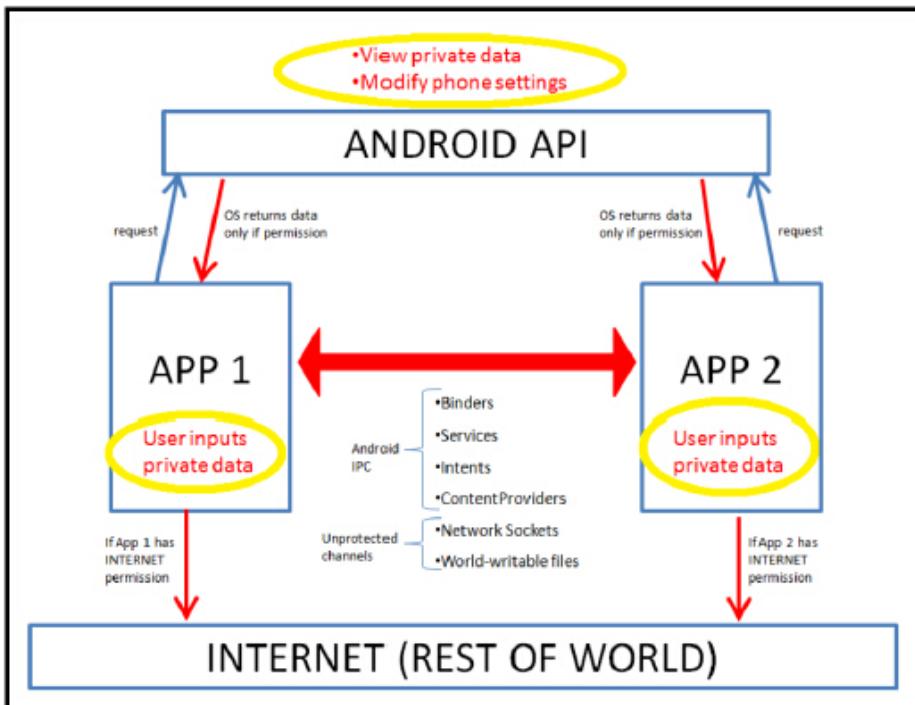


Figure 3.8: Detailed Android permission model[Han12]

Application Components Secure

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

Component Securing

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

Custom Permissions for Access Control

As pictured in figure 3.9, Android OS provides a list of default permissions, which service 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 should be added to the protected components in project's *AndroidManifest.xml*[Kei13].

Custom Permission Snippet

3. STATE OF ART

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]

```
<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 be 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 is 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, therefore it always 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 identifiers that is

related to sensitive information such as datasets[Kei13]. Content provider 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, to be more specifically[Kei13],

'Content Path Securing'

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

3.4.3 Conclusion

In conclusion Android is a secure-designed platform with sophisticated access control mechanisms and other security policies. But still Android application developers should apply security techniques to protect their project from malicious ill-formed applications.

3.5 Embedded System Secure Design

3.5.1 Overview

In this last section, 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 constraints must be taken into account during the design phase of a secure embedded system.

3.5.2 Embedded System Secure Requirements

The *Trinity of Trouble*[Pau04] three factors, namely *complexity*, *extensibility* and *connectivity* contribute together to make the embedded system more vulnerable and make the system secure requirements complexer. In the following a brief discussion about embedded system secure 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 verify the embedded system 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 important role in a secure embedded system. Message integrity and confidentiality must be ensured. Popular solutions are to use cryptographic algorithms such as symmetric chip, asymmetric chip and hash algorithms to protect message from illegitimately 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 applying anti-Dos attack mechanisms, which are already introduced in this thesis.

3.5.3 Software Security during Software Design Life Cycle

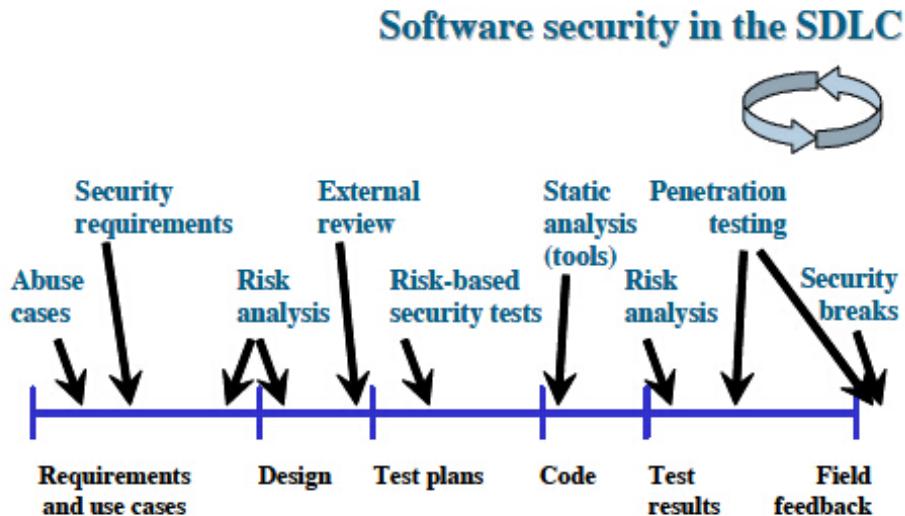


Figure 3.10: Software Security Consideration in Software Design Life Cycle[Pau04]

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

3.5.4 Secure Architectures

In order to keep an embedded system from malicious parties' attack, especially when present day's 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

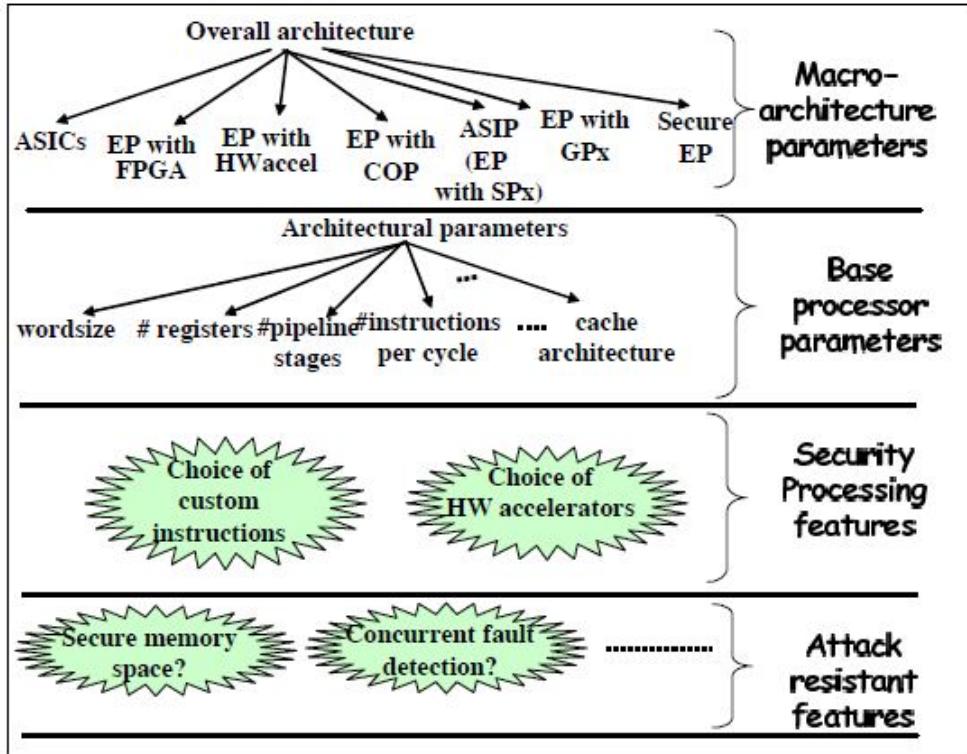


Figure 3.11: Architectural design space for a secure information processing[Pau04]

When the hardware architecture characteristics of an embedded system are confirmed. Following two secure aspect as shown in last two rows in 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 offer the same security services. Therefore in this layer, the choice of cryptographic algorithms should be made with consideration from hardware and software specific aspect. For instance, the full implementation of SSL protocol for a simple low end PAD device might not be a wise decision[Pau04].

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

3. STATE OF ART

- *Attack Resistant Features* When general security mechanisms used to ensure confidentiality, integrity and authentication are provided by embedded system, attempts to prevent such as denial of service attack also should be made.

4 Implementation Scenario

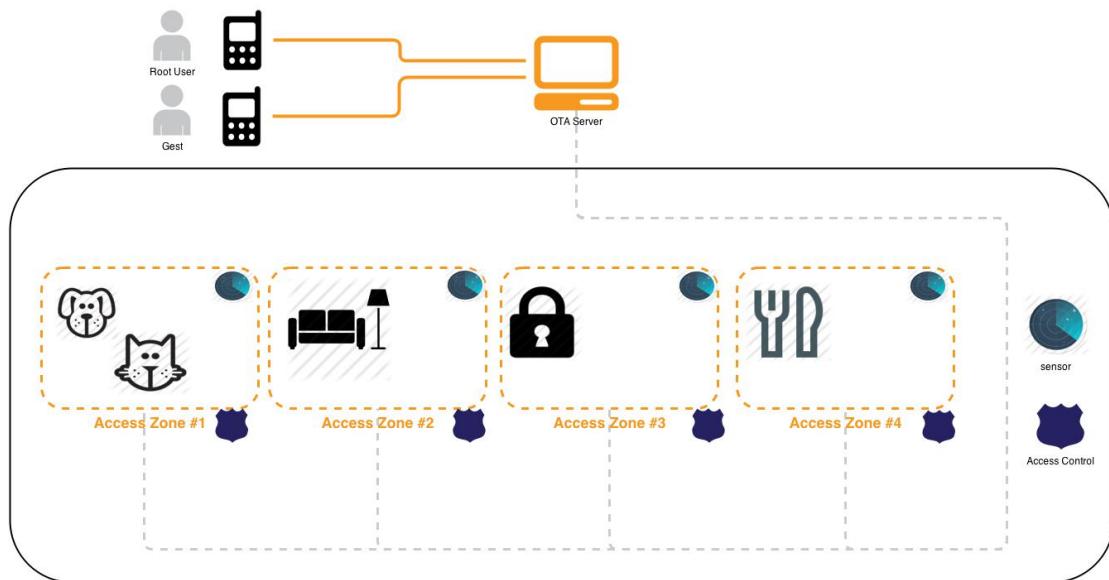


Figure 4.1: Smart Home

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

4.1 Overview

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

- Housing devices. Such as temperature sensors, coffee machine and digital door lock. They provide house holder a comfortable living condition by offering services such as:
 - General device state query functionality.
 - Door lock access control management.
 - Home temperature and luminance management and coordination.

4. IMPLEMENTATION SCENARIO

- Notification and alert.
- Smart Phone. The house holder with the help of smart phone is able to manage housing devices.
- Remote Administrator server, whose duties are authenticate communication partners, public key management and secure messaging.

4.2 Component Structure

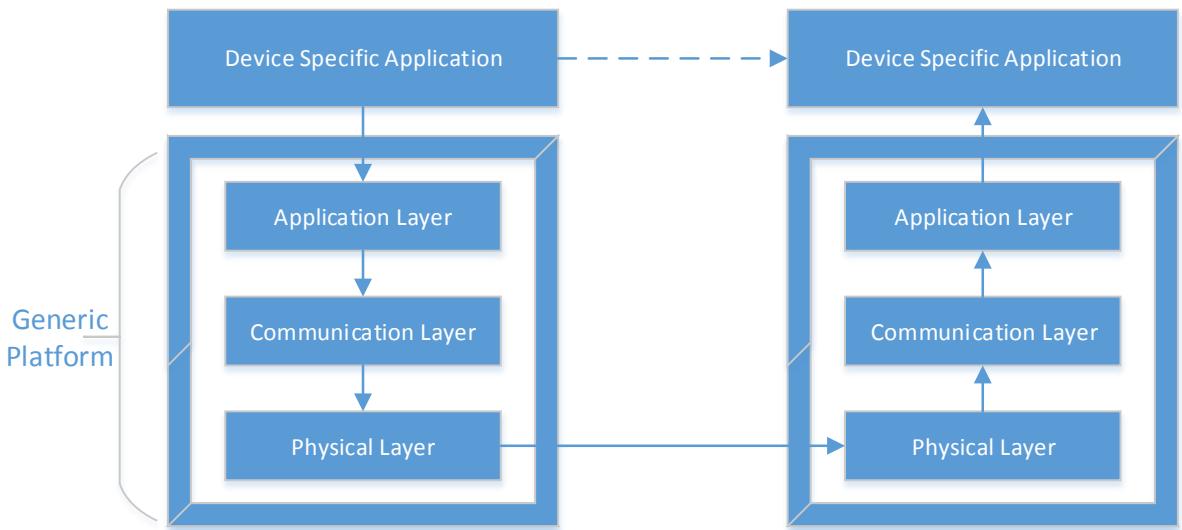


Figure 4.2: Smart Home System Component Structure

With inspiration offered by application described in section 3.1.4, in order to provide a generic application platform for vendor specific devices, 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 three layers. From bottom to top, they are:

- Physical layer. In this layer electronic devices and facilities are deployed. Alternatively lower level component could be also in this layer of higher level component.
- Communication layer. Smart card 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 application layer.

¹All housing devices that build the system and smart phone are generically referred as system components.

- 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 interface for higher layer device specific application.

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.

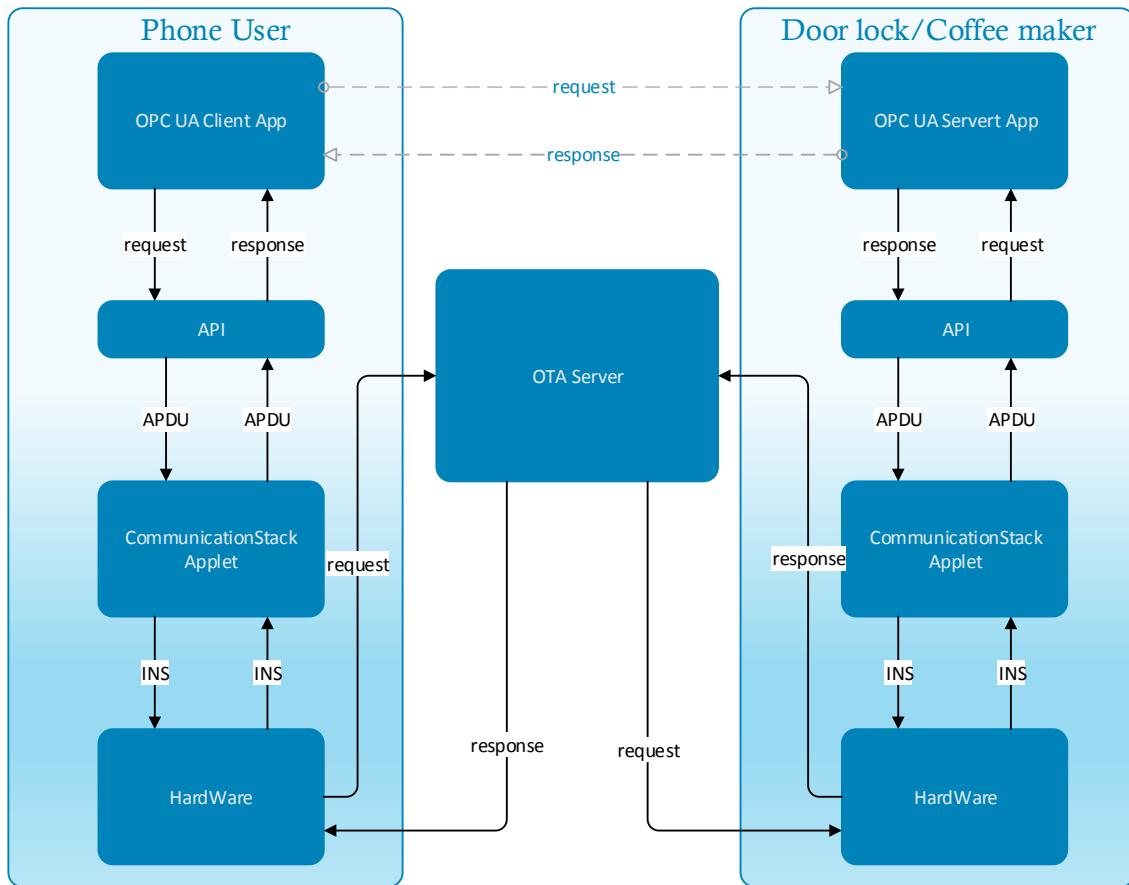


Figure 4.3: OPC UA Client Server Structure Example

4.3 Communication Flow

Figure 4.3 pictures the communication flow between two system components. OPC UA Client and Server application communicate with each other with the help of an OTA server. And the communication stack is in charge of creation

4. IMPLEMENTATION SCENARIO

and managing secure communications between OTA server and secure devices. Moreover OPC UA client application is able to communicate with OPC UA server application using Short Message Service(SMS) or TCP/IP based web service. One 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. OPC UA Client application installed on cell phone initially generates the *makeCoffee* request and forwards it to the internal API, such as openmobile API.
2. The internal API translates the request into remote APDU object, and the *CommunicationStack* applet will firstly try to connect with remote OTA/administrator server and to require the target coffee maker information.
3. The server will then in this step firstly check the identification of the message sender.
4. After a successful identification, OTA server is able to confirm the card-holder's identify and to grant necessary information to him and help card-holder transmit his request to targeted message receiver.
5. At last the *makeCoffee* command will be received and performed by OPC UA server application installed on coffee maker. Alternatively it also generates a response message and notifies the phone user that his 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

Communication stack 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 Communication stack's responsibilities are:

- initiate Http session based on TLS(proactive)
- trigger Http session based on trigger SMS send by OTA server(passive)
- rebuild broken communication channel
- message encryption as well as decryption
- message transmit

Moreover thanks to the self-containment structure, smart card itself does not dependent on other external resources, which could be extreme vulnerable to potential secure attack, and therefore provides a better hardware security and OS security.

4.3.2 OPC UA application Functionalities

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

- processing client subscription/publishing environment data.
- secure message exchange with client.
- authority management.
- historical data record.
- execution client's command.
- Managing corresponding homing device.

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

- submitting subscription/receiving published data.
- secure message exchange with server.
- sending command/configuration data.
- querying system historical data.
- providing user friendly interface.

4.3.3 Cell Phone Side 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, communication stack 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 Management/File(RAM/RFM) over Http protocol and PSK TLS security.

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

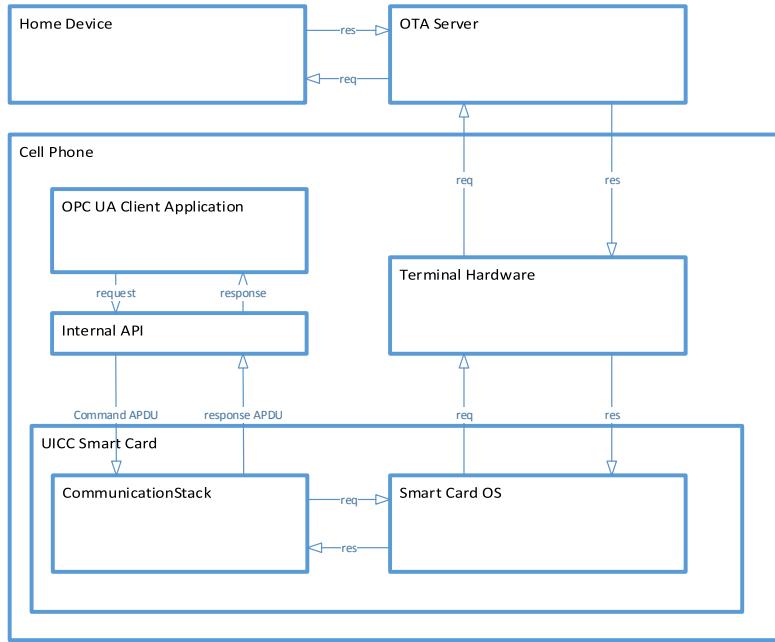


Figure 4.4: Client Structure

With these two concepts, smart card holding the appropriate Security Domain can act as Http client and is capable of packing APDU formate information into Http POST message and transmitting Http message to OTA server, which will then forward this Http message to target receiver.[Glo12]

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 lists of API used to initialize authentication process, to configure algorithm and keys used for message encryption and decryption, to perform secure messaging and etc.

4.3.4 Housing Device Software Structure

Housing Device here refers to sensors, electric device as well digital locks that together build up the smart home. Each secure device provides services described in 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 communication stack and device specific functions as well as device

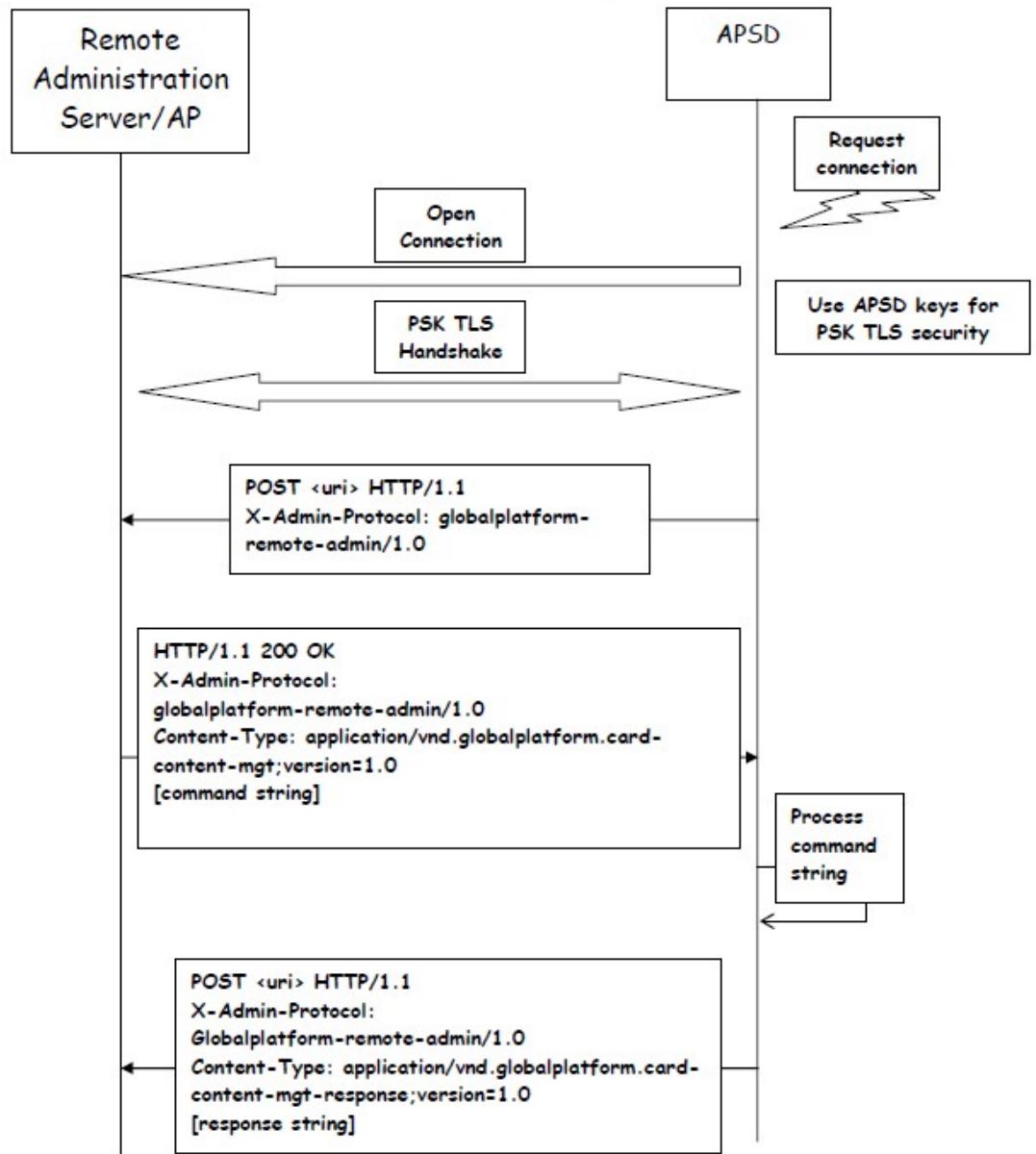


Figure 4.5: Communication Flow between an AP and corresponding APSD[Glo12]

data.

4.3.5 Implementation Tool Support

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

4. IMPLEMENTATION SCENARIO

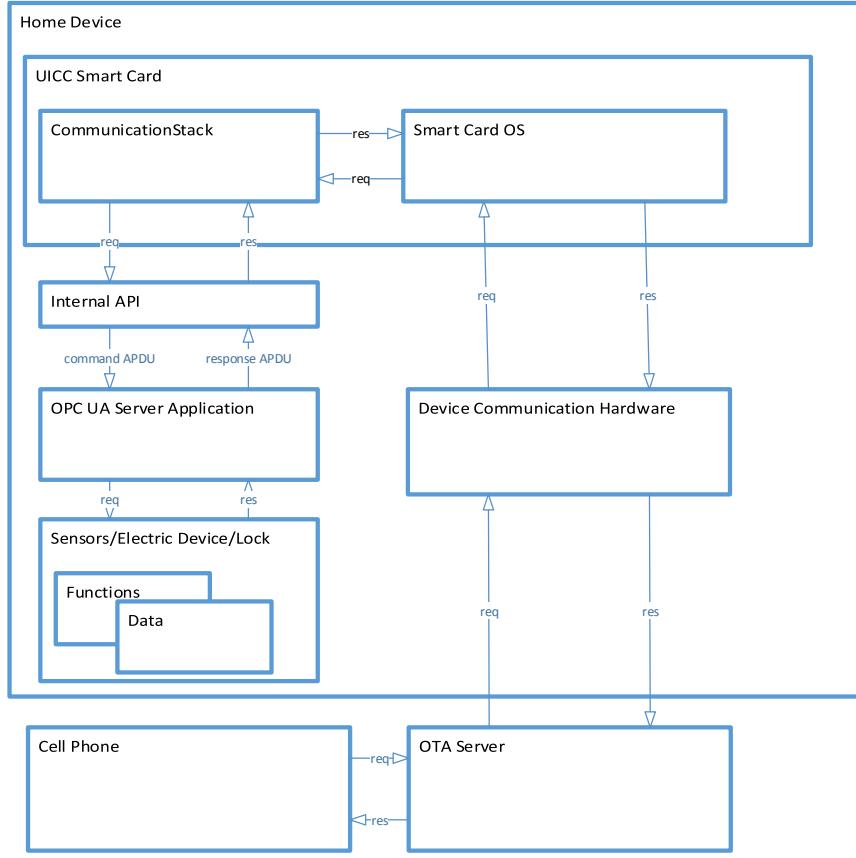


Figure 4.6: Server Structure

as well as to debug Java 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 test Applet also could be applied.

In either way, as next step *Universal Test Environment (UTE)* will be applied. *UTE* provided by Morpho uses Java languages developed test cases and test scenarios² to simulate desired use cases and observes corresponding smart card reactions. With the observed test result, tester will be able to analyze Applet's performance and debug corresponding application code. For instance *UTE* can integrate software models used to simulate security domain offered by *Globalplatform* and perform the sophisticate user authentication simulation process.

²Test scenario is a collection of relative test cases.

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 demonstration platform.

Smart Home Simulation Environment

In order to simulate data managed by home electronic device, *MySQL* database is introduced. 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 from previous subsection will then communicate with this web server for the purpose of scenario demonstration.

5 System Design

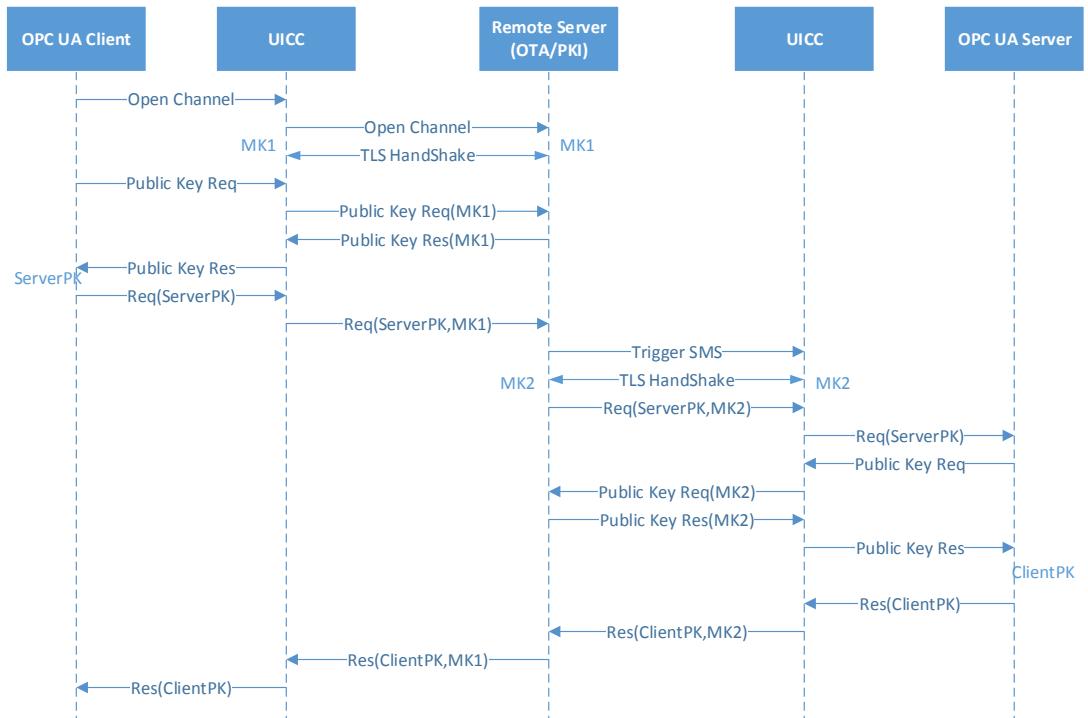


Figure 5.1: Message Exchange

In this chapter, I will present the crucial information about how I designed and programmed my system. To be more specifically, I am going to describe the design of my UICC Applet, Android Application as well as Smart Home Web Server.

As described in previous chapters, my demonstration system consists of three main parts, namely the on card integrated communication stack, Android application and Smart Home web service. Figure 5.1 illustrates the request and corresponding response message exchange process that occurs in the demonstration system. In my case, OTA is also integrated with a *public key infrastructure*, in order to perform secure peer identification and messaging. Therefor this OTA is also referred as remote administrator sever.

5.1 UICC Applet

5.1.1 Overview

The whole message exchange process consists of following phases:

- *Open Channel Phase* Whenever a secure housing device attempts to contact with the other, a communication session between Remote Server and this device must be created firstly.
- *TSL Handshake with Remote Server* After a successful creation of communication channel, a mutual peer identification through TLS handshake between Remote server and aforementioned homing device will be performed.
- *Require target's public key* When Remote Server identify the connected communication partner, it will grant him the expected target's public key.
- *secure messaging* Message sender will in this phase encrypt the message with target's public key and send this encrypted message through Remote Server to message receiver.

5.1.2 Work Flow

The components involved in this scenario are:

- CommunicationStack applet and associated Security Domain from 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 could be either initiated by Remote Administration Server by sending target applet a trigger SMS or be sponsored by applet itself. In both cases, *CommunicationStack* applet sends the first *OpenChannel Request* message. During the PSK TLS Handshake phase, APSD and remote server will agree on the to be used cipher suit and authenticate each other. After a successful PSK TLS Handshake, remote server will then grant authenticated user the public key of his

target applet and forward his following Http message, which encapsulates APDU command or response strings as body. The Http header is in compliance with GlobalPlatform standards and shown in following.

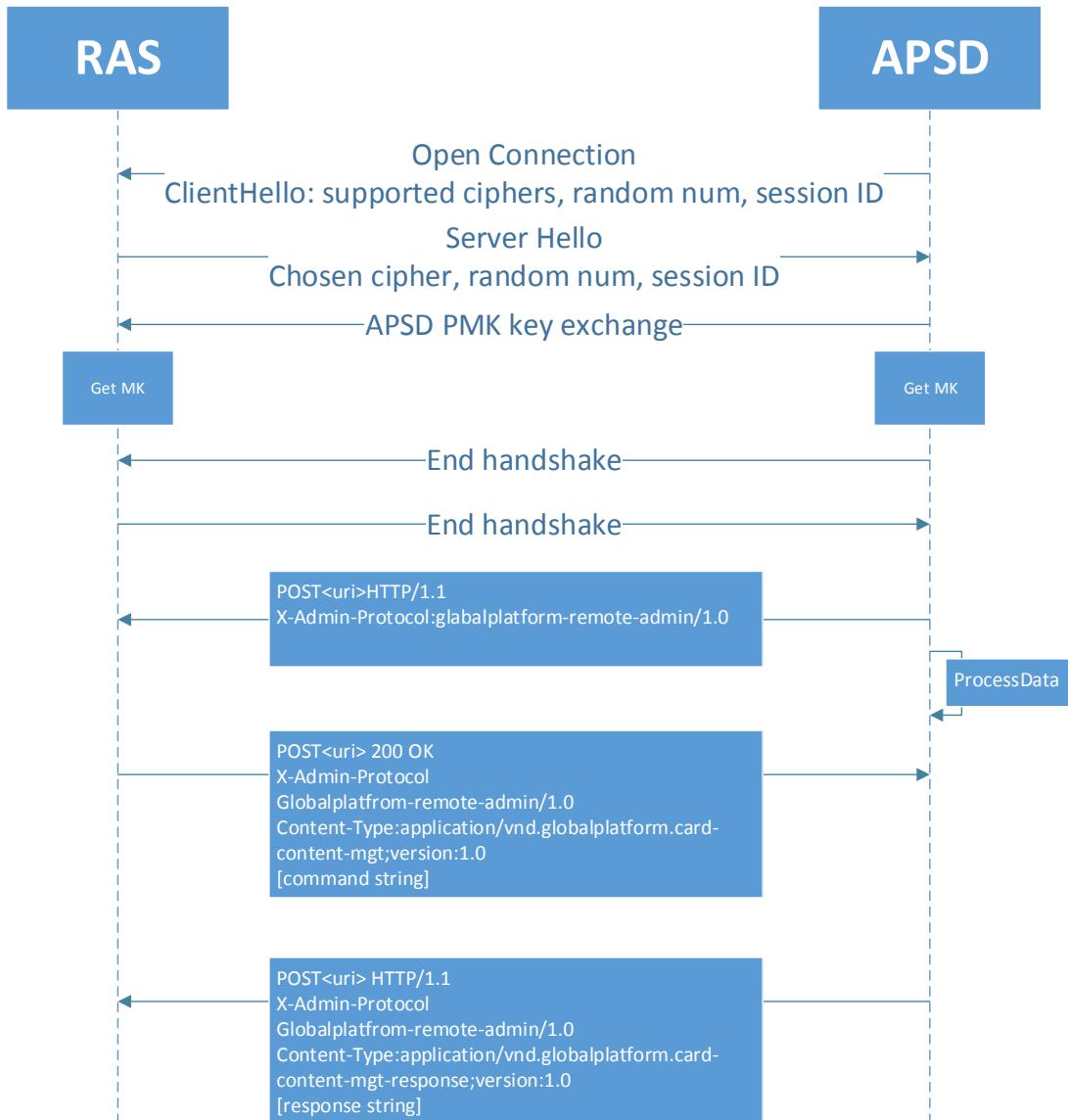


Figure 5.2: Handshake and message exchange between SD and Remote Server

Http Header Format

The Http message sent from remote sever to targeted applet obeys following schema[Glo11]:

Http Request Schema

5. SYSTEM DESIGN

```
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 targeted 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]
```

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

- *ok*, which means that the previous message is successfully received by applet.
- *unknown-application*, which stands for the error, that the targeted applet for previous message can not be found.
- *not-a-security-domain*, this errors occurs when targeted applet is not a Security Domain.
- *security-error*, as its name indicates, this values is returned if 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 successful finish of communication, remote server will send target applet Http message using following parameters:

- No *X-Admin-Next-URI* field is present in this Http message and message body is empty, which will be recognized as final message from remote server and then the session will be closed.

- No *X-Admin-Next-URI* filed is present but in this Http message body 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 Javacard applet code, the interface, which in essence is a set of command APDUs and corresponding response APDUs, between applet and CAD must be well defined. The CommunicationStack support two categories command APDU:

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

SELECT APDU

The header of this command APDU is fixed and *Lc* indicates the length of CommunicationStack AID. In *Data filed* real AID is saved. Two categories of response

Table 5.1: SELECT command APDU

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

APDUs 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 is used to let CommunicationStack applet verify the identity of terminal user. Moreover the PIN size is set from four bits

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

to eight and the verify PIN operation only allows three times wrong PIN input before a successful identification.

As described in tabl 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	After 3 times wrong input, PIN is block

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.

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 used to let CommunicationStack applet initiate the request to establish communication channel and create communication session above it. Commands listed in table 5.7 are supported.

Following return codes are expected in response APDU:

Table 5.7: Communication session creation command APDUs

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

Table 5.8: Trigger Session Return Code

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

Table 5.9: Close Session command APDU

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

Close Communication Session

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

Following return codes are expected in response APDU:

Table 5.10: Close Session Return Code

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

Process Communication Data

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

This APDU command and response set is used to provide functionalities to perform command processing between cellphone and other smart home device.

Following return codes as shown in table 5.12 are expected in response APDU.

Moreover *command data* in data filed of command APDU adopts following TLV format as shown in table 5.13.

5. SYSTEM DESIGN

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	

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

Table 5.13: Command data Type-Length-Value structure

Based on the predefined TLV structure, following APDU *A0210000129010910892069C040102030402* can be translated as command sent by smart phone to sensor whose unique id is *01020304*, for the purpose of querying current sensor value. And the expected response data length is set as two bits. 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 *javacard.framework.Applet*. 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.

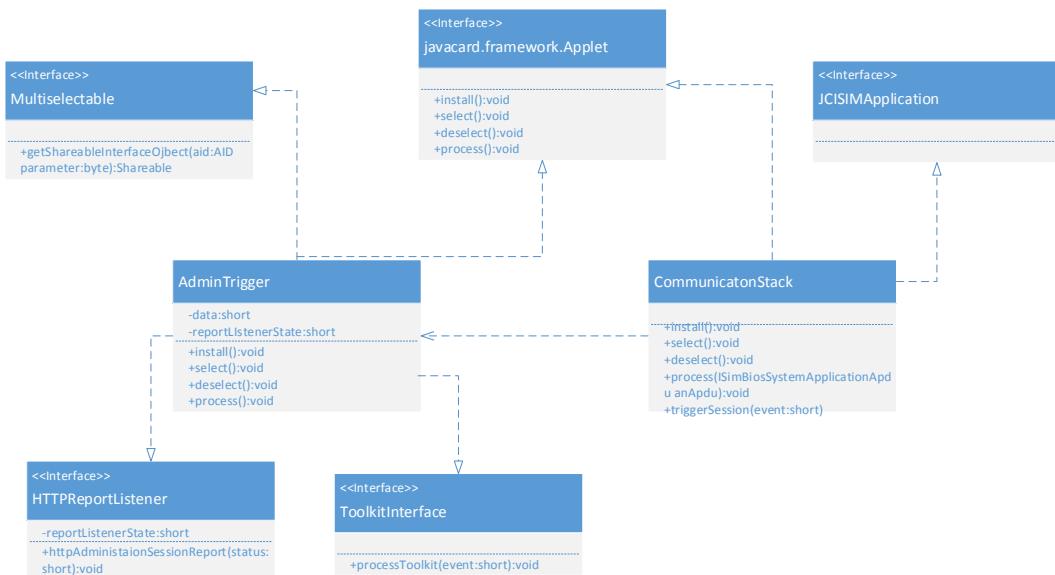


Figure 5.3: Class Diagram

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 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 PIN configuration is defined in this applet and described as below,

5. SYSTEM DESIGN

```
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;
```

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 wrong PIN input reaches 3 times before a successful verification, *doVerifyPIN* will block the PIN.
- *doResetPIN*. This handler in charge of reset PIN operation.
- *doUnlockPIN*. *doUnlockPIN* operation will be and only be invoked by card issuer.
- *doGetPublicKey*. This operation queries target public key from *Public Key Infrastructure* which is integrated in OTA server in my scenario.
- *doNotification*. After receiving command data, *communicationStack* applet will invoke this handler in order to inform corresponding Android application to fetch that command data.
- *doAppendMsg*. This handler configures the outgoing response remote APDU.

Except from statue 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.

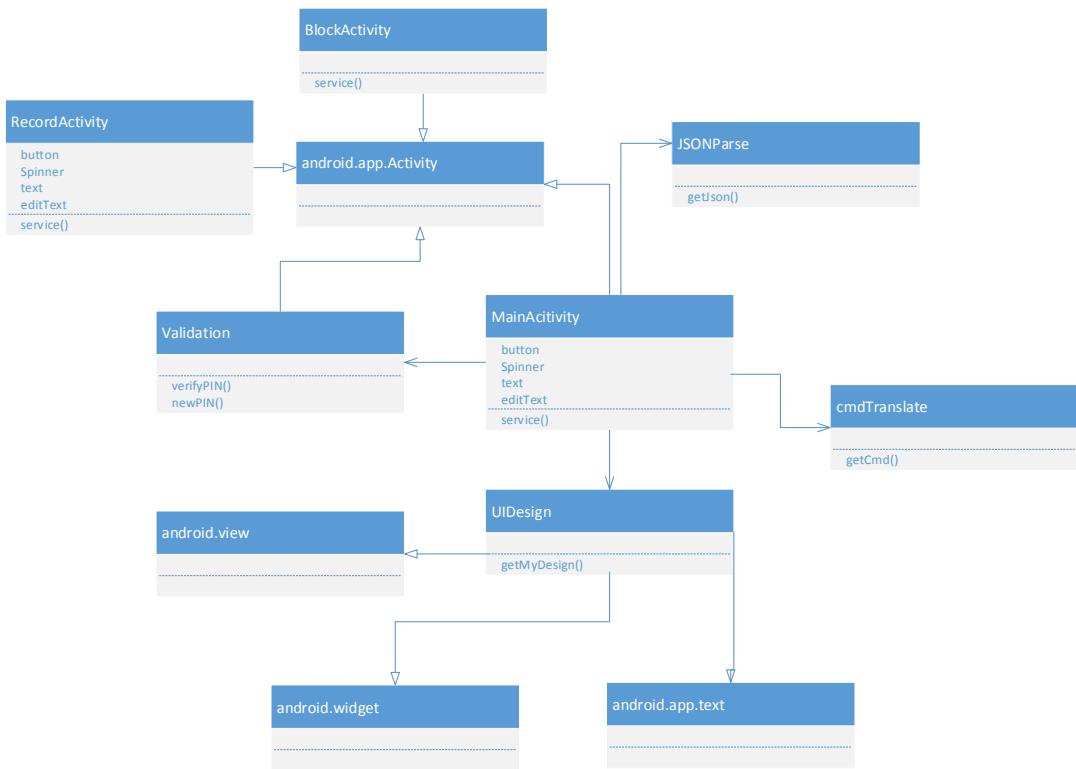


Figure 5.4: Android Application Class Diagram

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 provide methods such as *getCmd*, *getCmdTlv* and *byteArrayToHexString*, which are used to translate user commands into byte arrays based on rules defined in table 5.13.
- *JSONParser*. With supports 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.graphics* 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

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 should input his password and username.
- *main.xml*, which defines the layout of main screen.
- *list_item.xml* and *record.xml*. Those two xml files together picture how to present historical record in a list arranged by modified data.
- *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 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 connect 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 desire 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 logical channel, one Android application is able to send command to and get response from a secure element using *bytep[] result = logicalChannel.transmit(cmdApdu)*.

Layout Adoption

Activity class adopts the layout defined in 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 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 introduced in 2.6.5, 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 realize user authentication based on inputted user-name and password.

- *BlockActivity*. After three time wrong failed user identification, my application will invoke this *BlockActivity* and accordingly block the PIN.
- *MainActivity*. This class realizes client functions defined in 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 quired by Android OS. Such as composing components, access permissions and necessary libraries[Goo]. In order to provide a secure application environment, in my *AndroidManifest*, only the *ValidationActivity* can be launched by Android system and always run as the initial task. Moreover, no *intent-filters* are declared for my activities, which means they can only be started or visited by the explicit intent sent by *ValidationActivity* as show 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 used in web applications

5.3 Smart Home Web Server

Smart Home web application are also designed for the purpose of testing my smart card applet, Android application as well as simulating Smart Home 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 applications.

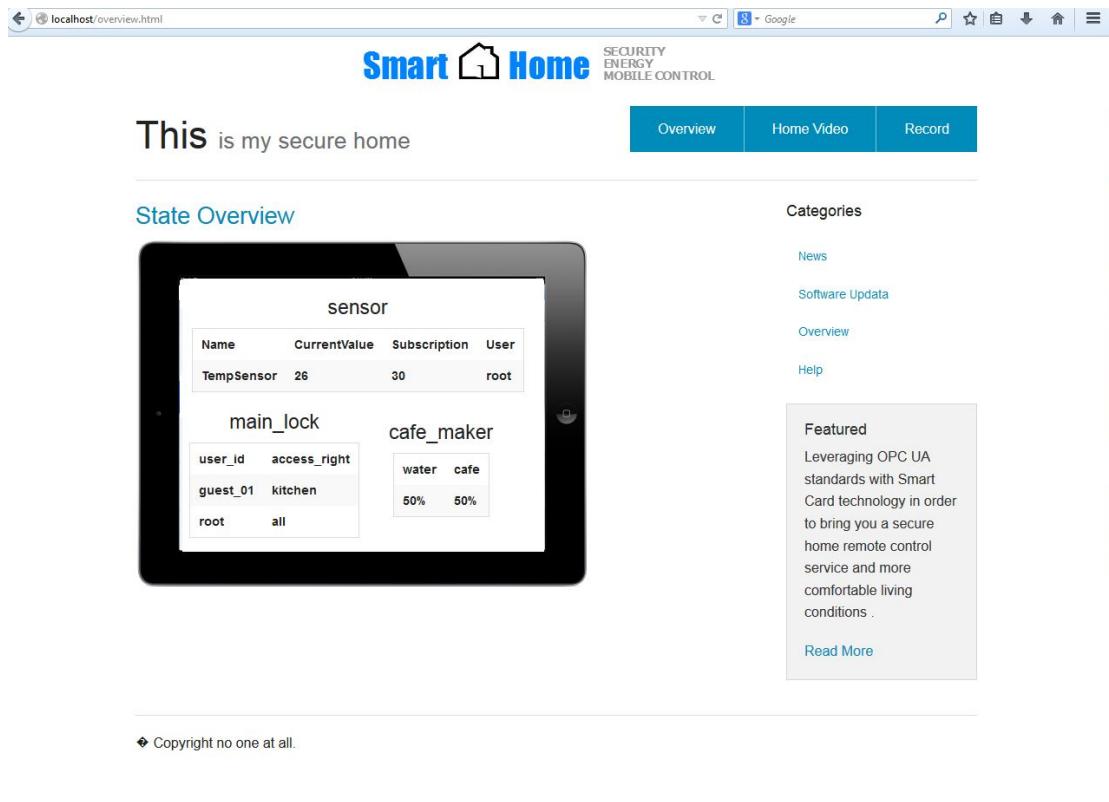


Figure 5.7: Smart Home Web Overview

Figure 5.7 illustrates the web page that shows all three housing devices states. 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 browse historical record of each home device. Detailed design process has limited relation with the subject of this thesis, therefore will not be presented.

6 Implementation Test

In this chapter, a brief demonstration testing cases are presented. To be more specifically, two testing scenarios related with *CommunicationStack* applet and simulated remote server are designed. At the same time, the testing cases demonstrating communication between Smart Home web server component and Android application are designed.

6.1 Remote Management Testing

The aim of this testing scenario is to prove the feasibility of applying Smart Card and remote application/file management protocols proposed by Globalplatform to provide a secure dual authentication and messaging platform for other higher leveled applications. My Javacard Applet - *CommunicationStack* together with correspondingly designed *UTE* test cases are employed and build the testing environment.

6.1.1 UTE Test Case

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 also the cornerstone for a successful remote application management.
- *Secure Messaging Exchanging* 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 a remote administrator server and encapsulates information necessary for the construction of a secure Http channel in a *TriggerPushSMS* object and sent this SMS to target smart card applet using simulated GPRS connection. The SMS structure is pictured in figure 6.1 and includes parameters which are categorized as following:

- *Connection Parameters*, which includes network access name, bearer description and other parameters describing the simulated remote server.

6. IMPLEMENTATION TEST

```
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: Push Short Message including all parameters which are necessary for the creation of communication channel

- *Security Parameters* In security parameters, proposed TLS 1.2 connection information, such as suggested cipher suit, is provided
- *Smart Card Keysets*, that consists of key identifiers as well as indicator used for the select of cipher keys and signature keys.
- *Http connection parameters*. Also parameters such as retry counter, timeout values are included in this short message.

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

The TLS *client hello* is demonstrated by figure 6.2. And in this demonstration case *TLS_PSK_WITH_NULL_SHA256* is chosen. At last, the master key, shown in figure 6.3 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: TLS client hello sent by smart card applet

```

verifyClientKeyExchange: PSK Identitiy -> passed
verifyChangeCipherSpec: Value 0x01 -> passed
verifyFinished: Verify-data -> passed
Master Secret:
struct {
    master_secret = 590407CDA02F0A60F8F56D4D59A0E386C6A25BCD59530BCB6E86237AA4988F1BC025FE8E076DBBA890768E651879A01;
};
Key Block:
struct {
    client_write_MAC_secret = C4601C7A81F5480E62B4C391EA19BB1BD1346D7B86C109BC1A37CBFB33156175;
    server_write_MAC_secret = 6C6163E7A3A548DE516148D789D70BB5F936A883DF35CDBF9FF699B506265E07;
    client_write_key = ;
    server_write_key = ;
};

```

Figure 6.3: Successful generation of master key between remote server and *CommunicationStack* applet

6. IMPLEMENTATION TEST

■ claMasked	A0	byte
■ ins	25	byte
■ cmd[0x240]	4304	byte[]
■ [0x000-0x007]	A0 25 00 00 02 02 06 01	byte
■ [0x008-0x00F]	01 02 00 05 B0 00 23 01	byte
■ [0x010-0x017]	2F 34 31 30 37 30 31 30	byte

Figure 6.4: *cmd* Buffer in *CommunicationStack* applet.

Secure Messaging Exchanging

Based on in previous paragraph 6.1.1 created secure channel and Http Connection, in this test case, smart card applet and remote server perform secure message exchange. As illustrated in figure 6.4, my applet is able to receive Http Message from remote server and retrieve the encapsulated command data. Furthermore as shown in figure 6.5, at the end of this testing scenario, remote server receives a Http message from *CommunicationStack* and the content of this message is apparently secured, whose clear test is presented in figure 6.5.

```
HTTP Request:  
[POST /CEH/cehocp HTTP/1.1  
Host: 80.66.10.200:8080  
X-Admin-Protocol: globalplatform-remote-admin/1.0  
X-Admin-From: 007  
Content-Type: application/vnd.globalplatform.card-content-mgt-response;version=1.0  
Transfer-Encoding: chunked  
X-Admin-Script-Status: ok  
  
D  
4  
0  
]  
D
```

Figure 6.5: Http Response Message received by remote server

```
=====Response from target applet=====  
44 0D 0A 01 12 34 00 01 02 03 04 05 06 07 08 09 0D 0A 30 0D 0A 0D 0A  
01 12 34 00 01 02 03 04 05 06 07 08 09  
=====END=====
```

Figure 6.6: Http Content Translation

6.2 Android Application Testing

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

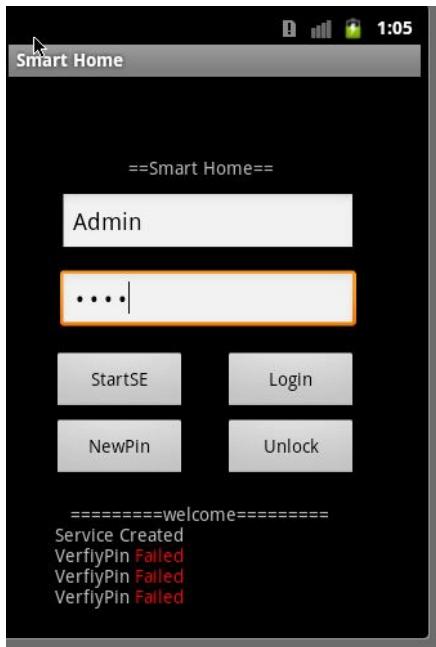


Figure 6.7: Wrong PIN input

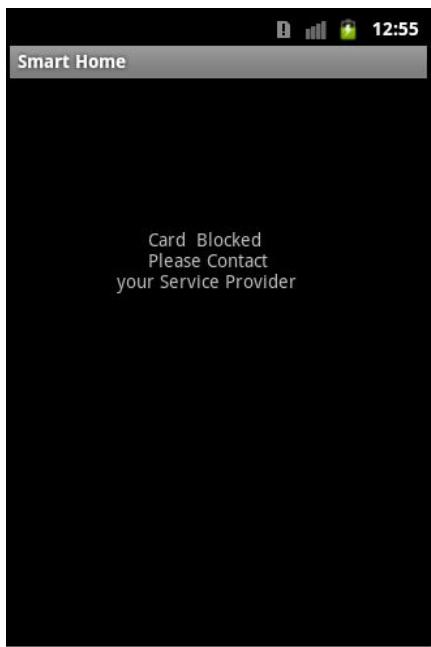


Figure 6.8: PIN is blocked

6.2.1 PIN Verification

As already introduced, in order to use functionalities provided by Smart Home, phone holder must authenticate himself by inputting PIN for Android application. But after three times wrong PIN input, as shown in figure 6.7, PIN will be locked. And only the service provider can unblock this 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 shown in following figures.

6.3 Conclusion

In conclusion, as the result tests show, my proposal is practical and secure. My demonstration system is able to create secure connection between communication partners, to perform successful secure messaging. Moreover, all exchanged messages are well-formed and carefully encrypted. My applications take care of the integrity, confidentiality of user credentials and at the same time provides traceability records.

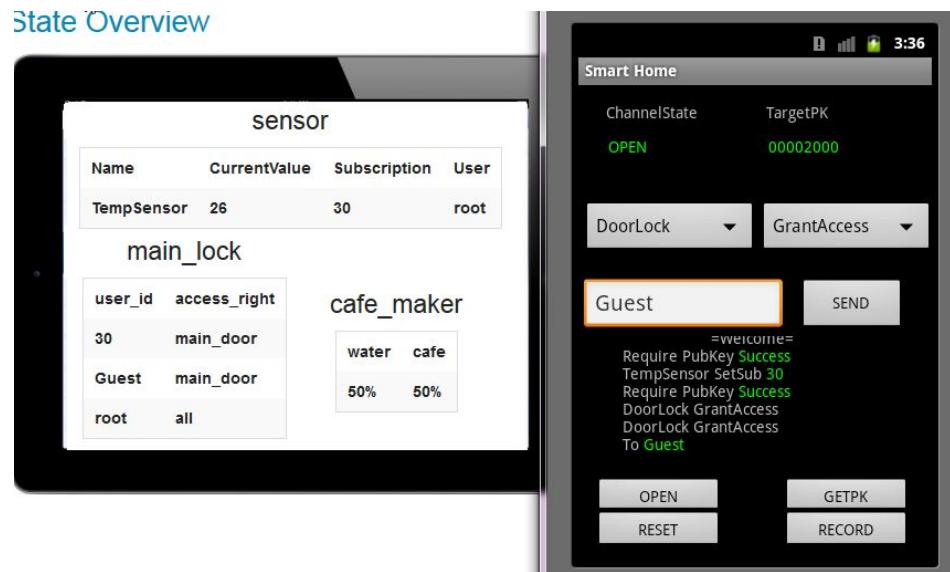


Figure 6.9: Using Android application remotely managing home devices

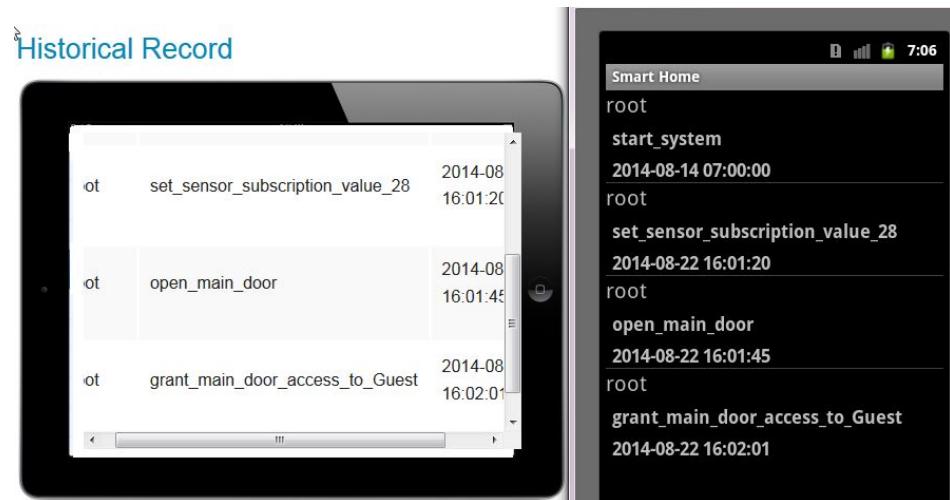


Figure 6.10: Historical Record Overview

7 System Security Analysis

In this chapter, I will perform a secure analysis based on the demonstration system described early for the purpose of confirming the feasibility of my proposal. 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 proof that there exists at least one secure countermeasure in my system.

7.1 Attack Tree Overview

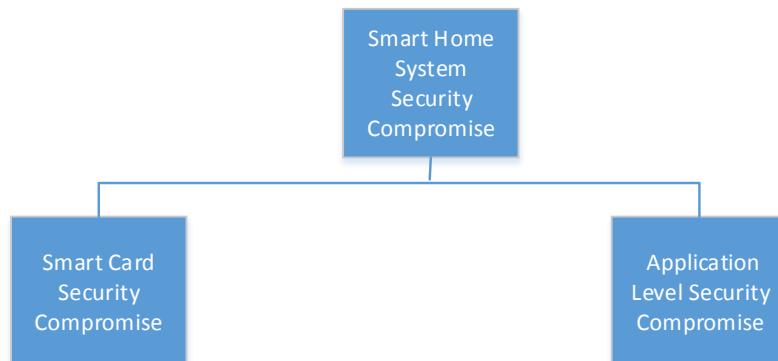


Figure 7.1: Attack tree overview

As shown in figure 7.1 the attack tree is categorized into two main types. The first type group describes attacks that the third malicious party could apply to the smart card employed in my scenario. And the first ones contain higher application level related threats, namely the security threats to 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 is going to be described.

7.2.1 Sub Attack Tree and its Content

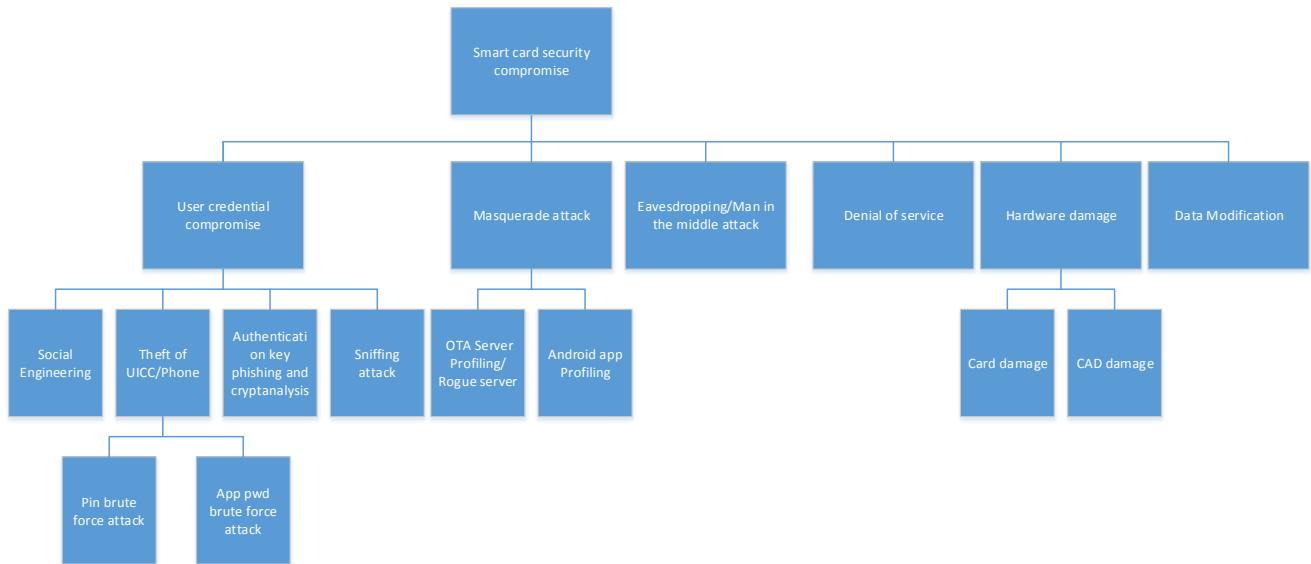


Figure 7.2: Smartcard Security Compromise

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

Secure 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 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 users 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 attacker.
 - Key phishing and cryptanalysis/Sniffing. As introduced in 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 smart card key management mechanism and 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*. If an Android Application claims himself as the smart card Application, it must provide the according Application signature, as described in essential secure

element control protocols, which are also provided by the *GloabalPlatform*. Since any fraud Application can not pass the exam of providing the correct Application signature, users of smart card will not worry about Android Application masquerade attack.

- *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 victims.
- *against DoS attack.*.. The remote administrator server could be a victim of denial of service attack, when the adversary floods message 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 protecting mechanisms against physical attacks, such as the already introduced power difference analysis and so on, therefore 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 using checksum algorithm, which means any information modified by a third party is going to be ignored.

7.3 Application Level Security Compromise

In this section, the second category of threads is discussed and accordingly 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 figured as 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

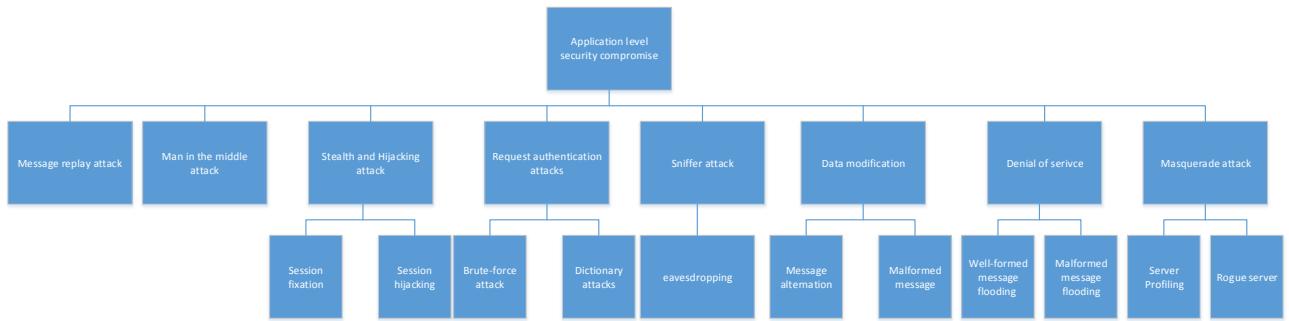


Figure 7.3: Application Level Security Compromise

- a) Session fixation
- 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) Malformed 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 capture a illegal message exchanged for instance between two housing devices in my scenario, he can never able to reply this message, because each request and response message is protected by its session IDs, secure channel IDs 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, the users don't need to concern attacks in this category.
- *Request authentication attack.* No matter either the 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 introduced in OPC UA standards, any by third part 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 to the other partners before he can perform any other actions. Moreover it also has to be able to perform dual authentication with the remote administrator server. Even when the fraud application receive a message, it doesn't possess the required private keys to decrypt the message.

7.4 conclusion

In conclusion, my proposal system is strong against the known attacks by combination of the magic bullet - smart card and the newly released OPC UA specifications.

8 Summary and Future Work

The last chapter summarizes your thesis and draws conclusions.

8.1 Summary

This chapter summarizes the content of your thesis.

8.2 Future Work

Unfinished solutions and open questions are discussed in this chapter.

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
- [Avi09] AVIK CHAUDHURI: *Language-Based Security on Android*. 2009
- [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
- [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: *GlobalPlatform 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 Internet 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 CHOURHARY, 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 ROH-JANS, 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
- [SID13] SID: *MQTT vs OPC-UA : Das HTTP der Industrie 4.0?* <http://dennisseidel.de/the-http-for-industrie-4-0-mqtt-vs-opc-ua-german>. Version: 2013
- [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;Februray 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
- [Yaj11] YAJIN ZHOU, XUXIAN JIANG: *Dissecting Android Malware: Characterization and Evolution*. September, 2011

Bibliography

- [Zhi00] ZHIQUN CHEN: *Java Card Technology for Smart Card.* June 2000