HSM Project
JavaCard Applet & SCP

Documentation

Deniz Agaoglu
Jerguš Lysý

1.5. 2017

Table of Contents

1 Introduction

For the purposes of the HSM project the JavaCard applet has to be implemented as the working key provisioner for the application. This document shall specify this part of the project in more detail as well as the implemented applet.

1.1 Design

The aim of the applet is to provide a cryptographic key that is used by the application. The key transmitting is made in a rather secure way, which consists of the following main features:

- Secure channel opened on JavaCard using three cryptographic keys.
- Key protected by a PIN (required from the application) and PUK (set during applet installation).
- Protected from running any applet functions by using automata-based programming approach.

All of these three concepts are introduced in the applet and are discussed in the following sections. The implementation of these features is described in the next chapter. The last two chapters presents the usage of security domain and secure channel of Global Platform.

1.2 Security

For the security of the overall project the secure channel must be initialized between the application and the applet. These are the goals of using such a secure channel:

- Mutual authentication of the JavaCard a the application.
- Integrity and confidentiality of APDUs.

All of the goals are reached by using the Security Domain that has been already installed inside the JavaCard. Using this it is only necessary to provide mandatory keys and open a secure channel between the card and the application. Keys can be sent into the card using GP Pro, while the secure channel can be open using the GP API. The channel is then open on the application side using functions from GP Pro. The communication is then sent through this channel to the Security Domain and then forwarded to the applet itself where it is unpacked.

1.3 Applet

The applet provides a key for an application. For this reason it stores the key and provides functions for setting/getting this key out of the card. This key has to be protected from malicious users and the applet has to protected from running unprivileged functions. PIN, PUK and and automata-based programming are presented in the applet to reach the desired protection.

- PIN is used every time the user requires to send the key from card to the application. This PIN firstly set by the user using the application. Then it can be changed only when verified with this PIN again.
- PUK is used when the user fails to verify by PIN. This can happen in a case that the user ran out of the available PIN tries. PUK is set during the FACTORY state of the applet and is not supposed to be changed. If the user verifies with the PUK code then the applet is changed to NORMAL state.
- Automata-based programming provides a way to call functions inside the applet in a secure way. There are several states inside the applet. Each state specifies what functions can be called. Example of such an approach is to have a factory state. The applet is in this state right after the applet has been installed. With this state set inside the card the user can send change, generate or send a key. But no other functionality is provided. For the PIN verification the other state has to be set. That is reached by calling a certain function. Using this approach the calling of functions is watched and thus is protected.

2 Applet implementation

Before describing the implementation details it is good to note that the entire project, as well as the applet was created in Java. The applet is built with JavaCard Kit 2.2.2 and Global Platform 2.1.1 library.

Before installing the applet the PUK has to be prepared. This PUK will be then set as a parameter of install command and then set to the PUK structure inside constructor of an applet. PUK is not changeable from the applet and thus has to be saved.

2.1 Instructions

The applet implements several instructions that can be run. To be able to run then it is also required to set the APDU class field to the value 0xB0, which specifies this applet.

2.1.1 INS_SENDKEY (0x50)

Instruction calls the function *sendKey* and is the response APDU will contain the key bytes. This function can be run only if the PIN was previous verified or the applet is in FACTORY state. If the applet is in the FACTORY state then the function does not change it. If the card is in AUTHORIZED state then the state is changed to NORMAL. The instruction does not accept any data or parameters.

2.1.2 INS_CHANGEKEY (0x51)

Instruction calls the function *changeKey* and does not returns any data. This function can be run only in FACTORY state and is performed when the key needs to be changed. The new key is either accepted in the APDU from the application – the parameter P1 is set to 0 – or is generated a new one using on-card RNG – the parameter P1 is set to 1.

2.1.3 INS_SETPIN (0x52)

Instruction calls the function *setPIN*. It requires data to represent the PIN that is to be set and can be run only when the applet is in the state SETUP or AUTHORIZED. If the state is SETUP, then there was no previous PIN set and thus needs to be set. If the state was AUTHORIZED then PIN was already set and has to be verified to change. The function changed the state of the applet to NORMAL.

2.1.4 INS_VERIFYPIN (0x53)

Instruction calls the function *verifyPIN*. This instruction can be run only when the applet is in state NORMAL. It requires data to be sent in the APDU that represents the PIN being verified. The function can then change the state to either AUTHORIZED or FAILED. The former of the two is set when verification passed successfully, while the later one is set when PIN tries are exhausted. After successful verification the PIN counter is reseted.

2.1.5 INS_VERIFYPUK (0x54)

Instruction calls the function *verifyPUK*. This instruction can be run only when the applet is in the state FAILED or NORMAL. In the case of former one it means that the verification of PIN has failed and user has to authorize with PUK. If the state of applet is NORMAL then user has to set PUK and if the verification is successful then the state is changed to FACTORY, otherwise the card is locked.

2.1.6 INS_RUN (0x55)

Instruction calls the function *run*. This function is called whenever the card has been configured and is about to be used. This function changes the state from FACTORY to SETUP and prepares the card to receive and set the first PIN.

2.1.7 INS_SETPUK (0x56)

Instruction calls the function *setpuk*. This function can only be used in FACTORY state and should be used only once to set the PUK code before the card it given to user to be normally used. This function does not change state.

2.2 States

There are several states of the applet. These states are implemented as constant values which are assigned to a state variable during the applet operation. The following states are available:

- **FACTORY** – represents the state of the applet after the installation. In this state the user can change or generate a new key as well as send it out. It is also convenient to have an option to get into this state once user needs to change the key – after successful PUK verification, applet state is changed to FACTORY (this holds only when the previous state was NORMAL).
- **SETUP** – once the function *run* is called and successfully ended, the state changes to SETUP. This is the state when PIN needs to be set for the first time before applet can operate normally. This state can be reached only from the FACTORY state.
- **NORMAL** – after the PIN was set the state is changed from SETUP to NORMAL. This state represents a normal operating mode of the applet and during this state functions such as verifyPIN and verifyPUK can be run.
- **AUTHORIZED** – once the application successfully verifies with the PIN the applet changes to the this state and the function *sendKey* can be called to transmit the key to the application. After this operation the state changes back to NORMAL.
- **FAILED** – if the verification of PIN fails due to exhausted PIN tries the state FAILED is set. During this state the user needs to verify with PUK. If this verification is successful then the state is changed to AUTHORIZED and counter of the PIN and PUK are reseted.
- **LOCKED** – after user fails with PUK verification the state is changed to LOCKED. This

state is not possible to change and is tested in the *select* function of applet – applet will not be selected. The automaton with described states is displayed in the following figure.

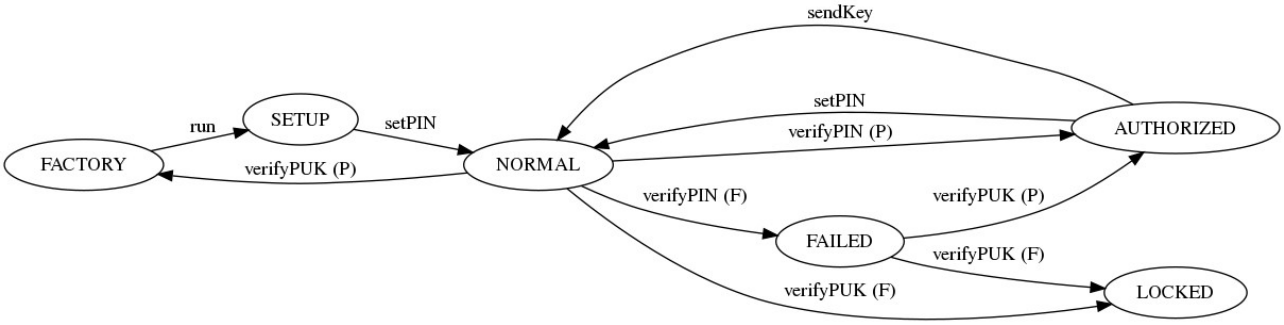

Figure 1: Automaton with states of the applet. Notice it does not display loops – only functions changing state.

2.3 Summary

The following table summarize all the instructions and their relation to the applet.

*INS_*	*Code*	*P1*	*P2*	*INPUT*	*DATA IN*	*OUTPUT*	*DATA OUT*	*From state*	*To state*
SENDKEY	0x50	-	-	N	-	Y	key	AUTH,FACT	NORM,FACT
CHANGEKEY	0x51	0/1	-	Y	-/key	N	-	FACT	FACT
SETPIN	0x52	-	-	Y	PIN	N	-	SETUP,AUTH	NORM
VERIFYPIN	0x53	-	-	Y	PIN	N	-	NORM	AUTH
VERIFYPUK	0x54	-	-	Y	PUK	N	-	NORM,FAIL	AUTH,FACT
RUN	0x55	-	-	N	-	N	-	FACT	SETUP
SETPUK	0x56	-	-	Y	PUK	N	-	FACT	FACT

3 Secure channel

Secure channel functions and protocols of Global Platform has been used to create a secure environment between JavaCard and the PasswordSafe application. As in describe in the specifications of GP, secure channel of our project has been divided into the following phases : i) Secure Channel Initialization, ii) Secure Channel Operation and iii) Secure Channel Termination.

i) We begin at this phase when the JavaCard is connected. Information about channel is gathered after plugging the JavaCard and then a secure communication is needed. Channel and secure channel are initialized in APDU of the card and in the application.

ii) Bidirectional secure communication between the application and the JavaCard is established considering authenticity, confidentiality and integrity. A secure conservation is achieved by 1) Key Establishment Protocol, 2) Key Derivation Function, 3) Authenticated Encryption and 4) Authenticated Decryption. To generate shared session key in each conversation, Key Establishment Protocol is used in APDU (thus, trusted server is the JavaCard) and shared with the application via Key Transportation Protocol. Since we need different keys for ENC/MAC algorithms, we needed to use a cryptographic hash function. To achieve this, Key Derivation Function is used to produce AES types of keys (since it is longer, symmetric and provides confidentiality) from the session key. To protect the confidentiality and integrity, block cipher and MAC should be encapsulated to be sent to the application as decrypted secret key. Lastly, Authenticated Decryption by the application is applied to recover plain text message from authenticated cipher text coming from the JavaCard if MAC is verified.

Since these 4 parts of secure communication are what we expect, we used '02' SCP version which provides AES based cryptography for ENC/DEC/MAC communication. Thus, OpenSecureChannel (session keys, host challenge, SCP version, security level) function has been called by null session key, null host challenge, '02' SCP version and SECURED security level.

iii) Communication terminates when security level is set to NO_SECURITY. This happens after the JavaCard plugged out, unselected, powered off or blocked (if 5 PIN and then 5 PUK trails are failure, then the user becomes unauthenticated), channel is closed, the application is closed or communication end. Information about these states are reached by checking the status of the channel.

The following two figures and the table are taken from GPC_Specification-2.2.1.pdf. They show the mechanism of phase ii) Secure Channel Operations.

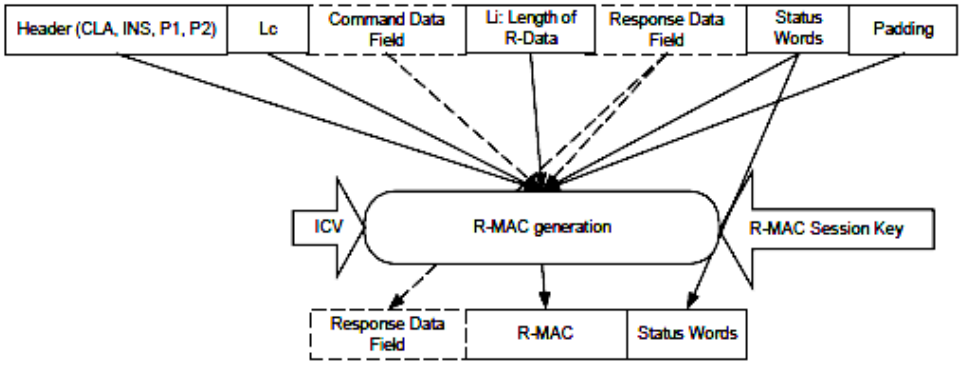

Figure E-5: R-MAC Generation

Counter value	Key
1	The MAC key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
2	The ENC key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
3	A subsequent MAC key, if any
4	A subsequent ENC key, if any
5	The data encryption key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
6	A subsequent data encryption key, if any

Table F-12: Counter Value for Session Key Calculation

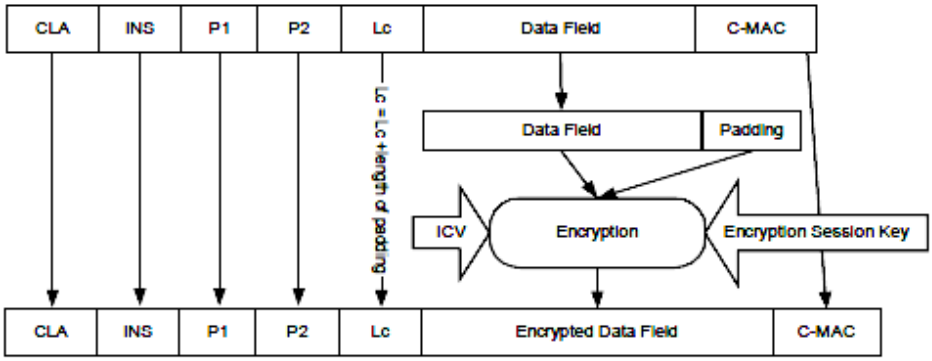

Figure E-6: APDU Command Data Field Encryption

3.1 Implementation

For opening a secure channel the GP API was used. Inside the applet two main function were called:

- *getSecureChannel* – using this function I can obtain a handle to secure channel of SD.
- *processSecurity* – this function will forward the message to the SD a returns a response. This works in a transparent way and applet does not need to know what security level has been used.

Another function that is needed in the applet is *getSecurityLevel*. This function is needed to invoke since it obtains information about what security is going to be used. It's needed to be able to call functions such as *wrap* and *unwrap* properly. It also provides information whether the authentication has occurred.

After processing the SC commands for establishing secure messaging[1] the applet calls *unwrap* function to process secured apdu, which basically verifies MAC and/or decrypt data if there was some. After calling this function, the instruction with apdu and its data can be processed further. Howeever the function *unwrap* is not used in the applet, since there is no support for R_MAC and R_ENCRYPT in the SCP inside javacard. This means that the responses are not secured, but sensitive data, such as the key that is sent to the application is encrypted using the GP function *encryptData*, which uses KEK stored inside SD.

The secure channel opening inside the application is done using the function *openSecureChannel*. This function is implemented inside GP Pro in GlobalPlatform.java file. GP Pro was thus used for our application, while it has been slightly modified. The main modification was to add extended modes for security levels. These are used only inside GP Pro and are not set while opening the secure channel – they are not sent while opening the secure channel session. The implemented modes are extDEC and extRMAC. The later one is already implemented by GP Pro[2], while extDEC adds a functionality to decrypt incoming apdu data using KEK. This mode is needed whenever the key is received from the applet in an encrypted form. For the simplicy it avoids padding, which still should be satisfied for data such as cryptographic keys.

1 Explicit method of invoking secure messaging was used.

2 It is a mode RMAC, which is also set in P1 of apdu sent within opening the secure channel. The extended mode extRMAC makes the code for RMAC run even if it has not been set previous. But this requires to implment RMAC inside the applet.