Scientific Journal of Informatics
Vol. 5, No. 2, Nov 2018

p-ISSN 2407-7658 http://journal.unnes.ac.id/nju/index.php/sji e-ISSN 2460-0040

Java Card Approach to Emulate The Indonesian National Electronic ID Smart Cards

Dwidharma Priyasta[1], Wahyu Cesar[2], Yanti Susanti[3], Juliati Junde[4]

[1,3,4]Pusat Teknologi Elektronika, BPPT, Indonesia
[2]Pusat Teknologi Industri Hankam, BPPT, Indonesia
Email: [1]dwidharma.priyasta@bppt.go.id, [2]wahyu.cesar@bppt.go.id, [3]yanti.susanti@bppt.go.id, [4]juliati.junde@bppt.go.id

Abstract

This paper presents a successful effort in emulating the Indonesian national electronic ID smart cards using Java Card. The aim is to provide the opportunity for other smart card products to contribute in the national electronic ID program. The life cycle status, the file system concept, commands and security procedures implemented in the Indonesian national electronic ID were reproduced and emulated in a Java Card applet. In the first stage, the emulator applet was tested using some test scenarios developed during the implementation phase. Later on, the emulator applet was tested in the real system under supervisioned by Direktorat Jenderal Kependudukan dan Pencatatan Sipil in order to measure its reliability as well as to determine that its behaviour is identical with the Indonesian national electronic ID. The results confirmed that this approach was fruitful.

Keywords: The Indonesian National Electronic ID, Java Card, Emulation

1. INTRODUCTION

The Indonesian national electronic ID uses contactless smart cards to store personal data. To date, it is only supported by two command-based type of smart cards for application creation with native operating system. These smart cards follow the life cycle status, the file system concept, commands and security procedures as defined in [1, 2, 3]. So far, the availability of chips for the Indonesian national electronic ID program is well maintained.

It is factual that smart cards technology is growing fast. The semiconductor side has provided faster processor speed, larger memory size, various types of memory and unique hardware security approaches. While the operating system side has enabled multiapplication and the latest cryptography algorithms. Therefore, considering about technology migration is a must for the Indonesian national electronic ID to overcome future needs after its first release in 2011.

Compatibility to the existing system is the key word for any technology migration. In this case, any new smart cards technology inserted should be able to run on the exis-ting system. Therefore, a program-based type of smart card such as Java Card is on the list, since they can be programmed to emulate a command-based type of smart card such as the Indonesian national electronic ID smart

cards, as long as the required features are supported. Java Card is also well-known as the latest technology provider which might be suitable for any migration purposes. An example approach to emulate a command-based type of smart card using Java Card had been done by J. P. Casanovas and G. V. Dame [4].

2. METHODS

2.1. Analysis dan Design

Architectural overview of a Java Card that contains the emulator applet is shown in Figure 1. The following steps had been conducted in the development of the applet.

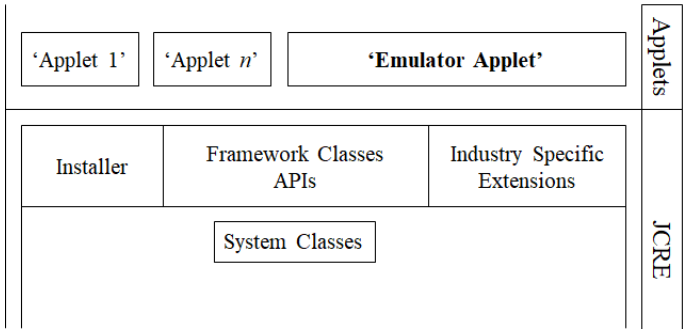

Figure 1. Java card with the emulator applet

The first step was to identify the life cycle status implemented in the Indonesian national electronic ID smart cards. These life cycle status are defined in [1], namely the Initialization state and the Operational state. These life cycle status can be applied to each application inside a smart card separately. The availability of commands and security procedures depends on the life cycle status.

In the Initialization state, applications and data are created and can be initialized. The following commands with their hexadecimal codes inside a bracket are available in the Initialization state:

1. GET CHALLENGE ('84')
2. EXTERNAL AUTHENTICATE ('82')
3. CREATE FILE ('E0')
4. SELECT FILE ('A4')
5. UPDATE BINARY ('D6')
6. READ BINARY ('B0')
7. ACTIVATE FILE ('44')
8. CHANGE REFERENCE DATA ('24')

Authentication procedure is required before executing some commands provided in the Initialization state. Also, a message authentication code (MAC) has to be applied to ensure data integrity during communication.

Meanwhile, the Operational state provides the following commands:
1. GET CHALLENGE (‘84’)
2. MUTUAL AUTHENTICATE (‘82’)
3. SELECT FILE (‘A4’)
4. UPDATE BINARY (‘D6’)
5. READ BINARY (‘B0’)

The readiness of commands in the Operational state depends on access requirements applied to each application. For example, the GET CHALLENGE command cannot be executed in an application which has no access requirements. But, for those with access requirements this command will be ready, and after successful authentication all communications from and to the smart card have to be encrypted through a Secure Messaging which is defined in [2].

The next step was to select appropriate Java Card. The following features are absolute musts to emulate the Indonesian national electronic ID:
1. UID length of 7-bytes
2. SHA-1 cryptographic hash function
3. single DES crypto algorithm
4. 3DES crypto algorithm with 2 and 3 keys

These features become mandatory when implementing functions for deriving crypto-graphic keys, calculating a message authentication code (MAC), and composing a Secure Messaging.

The final step was to decide which Java Card version and what programming tools to be used. Here, we chose the followings:
1. Eclipse Indigo SR2 Integrated Development Environment for Java Developers with Java Card 2 plug-in for Java applet programming
2. PC/SC contactless reader with USB 2.0 CCID host interface
3. Java Card version 2.2 with 64 kilobytes EEPROM that supports Global Platform version 2.1.1

2.2. Implementation

The Indonesian national electronic ID smart cards apply file system concept defined in [1], as shown in Figure 2.

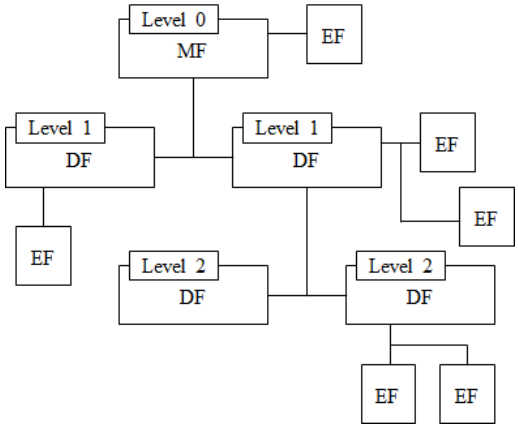

Figure 2. File system concept in ISO/IEC 7816-4

To handle the file system concept, three persistent variables were used to store the property of each category of files, namely the master file (MF), the dedicated files (DFs), and the elementary files (EFs). The MF is a mandatory root file according to [1]. The DFs are commonly associated with applications. While the EFs are the place for data storage which can be accessed by external entities. In Java Card version 2.2, the corresponding variables declaration is as shown in Figure 3.

```
Object[ ] hdr_mf;
Object[ ] hdr_df;
Object[ ] hdr_ef;
```

Figure 3. Persistent variables declaration

A. The Life Cycle Status (LCS)

The life cycle status apply to the MF and DFs. Information of the life cycle status is stored in the last byte of *hdr_mf* and *hdr_df*. According to [1], the indicated value for the Initialization state is '03', while for the Operational state is '05'.

B. File Selection and Access Control

The MF is automatically selected at the first time of communication. According to [1], the value '3F00' is reserved for referencing the MF. This file identifier is stored in the 2nd byte and the 3rd byte of *hdr_mf*.

DFs header (*hdr_df*) and EFs header (*hdr_ef*) have more information than MF. This information is related to the control parameter data objects (CP DOs) described in [1]. DFs header contains the file descriptor byte, the file identifier (FID), DF name, and the access control methods defined in [2] (which can be Logical Data Structure or Basic Access Control). While EFs header contains the number of data bytes in a file, the file descriptor byte, the file identifier (FID), and security attributes.

For successfully selecting a certain file, one has to consider the access control methods and security attributes attached to the file. This property is given at the file creation phase in the Initialization state.

C. Data Storage

Memory is dynamically allocated for any persistent data at the file creation phase in the Initialization state. Variable declaration for this purpose is as shown in Figure 4.

```
Object[ ] card_data;
```

Figure 4. Variable declaration for data storage

Persistent variable *card_data* has one-to-one relationship with *hdr_ef* through their array index. For example, *card_data* with array index *i* that has been instantiated as shown in Figure 5, has corresponding property stored in *hdr_ef[i]*.

```
card_data[i] = new byte[file_size];
```

Figure 5. A persistent variable instantiation

D. Class Byte (CLA)

The class of command was implemented according to [1]. Coding for the inter-industry class is '00', secure messaging is '0C', and the proprietary class is '80'.

E. Instruction Byte (INS)

All commands mentioned in section (2) were implemented as shown in Figure 6.

```
switch (buf[ISO7816.OFFSET_INS]) {
   case GET_CHALLENGE:
      Get_Challenge(apdu);
         break;
   case EXTERNAL_AUTHENTICATE:
      External_Authenticate(apdu);
         break;
   case CREATE_FILE:
      Create_File(apdu);
         break;
   case SELECT_FILE:
      Select_File(apdu);
         break;
   case UPDATE_BINARY:
      Update_Binary(apdu);
         break;
   case READ_BINARY:
      Read_Binary(apdu);
         break;
   case ACTIVATE_FILE:
      Activate_File(apdu);
         break;
   case CHANGE_REF_DATA:
      Change_Ref_Data(apdu);
         break;
}
```

Figure 6. Commands implementation

F. Status Bytes (SW1-SW2)

The processing states were implemented according to [1]. Coding for normal processing and warning processing is '9000' and '6300' respectively. While coding for checking errors are '6700', '6982', '6A80', '6A82', '6A8A', '6B00', '6C00', '6D00', '6E00' and '6F00'.

G. Security Procedures

When a DF is set to Basic Access Control, then access to its EFs might be in Secure Messaging after authentication. Therefore, the following items have to be prepared:

1. Key derivation method
2. Cryptography functions and padding
3. Function to compute a mac
4. Function to handle a secure messaging

Key derivation method generates the encryption key, the MAC key and the session keys. This method is based on the derivation of 3DES keys from a seed value K_{seed} described in [2]. The hash function SHA-1 was used during session keys generation.

DES/3DES cryptography algorithms were already given by the selected Java Card. Hence, preparing a single function for encryption and decryption was easy. Padding method 2 with 8 bytes zero Initialization Vector as defined in [3] was used in all processes that involved cryptography.

A function for calculating a MAC had to be created from scratch. Procedure to compute a MAC is based on [3], as shown in Figure 7. Send Sequence Counter (SSC) is a 64 bit unsigned integer which value is to increase every time before a MAC is calculated. The MAC length is 8 bytes.

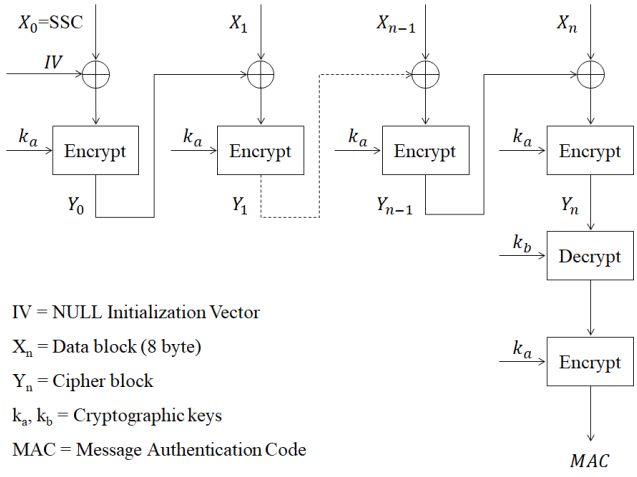

Figure 7. Computation of a MAC

A function for decomposing a Secure Messaging command APDU and for composing a Secure Messaging response APDU had to be created from scratch as well. This function is slightly complicated. Given an arbitrary data as input, this function was designed to decide whether to decompose or to compose a Secure Messaging.

The decomposing and composing processes require the encryption/decryption func-tion as well as the function that calculates a MAC. These two functions have been mentioned previously. Details of these two processes can be found in [2]. Focus should be put on the computation of a Secure Messaging command APDU and the computation of a Secure Messaging response APDU.

2.3. Testing

It is necessary to test all commands that had been implemented in the emulator applet. Therefore, some test scenarios were developed during the implementation phase. These test scenarios were designed to determine the behaviour of the emulator applet in accordance with the Indonesian national electronic ID. An example of test scenario is shown in Table 1.

Table 1. Read binary in the operational state

Life Cycle Status	Tester		Emulator applet
Operational	SELECT FILE [DF--]	⟶	
		⟵	SW1-SW2: '9000'
	GET CHALLENGE	⟶	
		⟵	RND.ICC || SW1-SW2: '9000'
	MUTUAL AUTHENTICATE	⟶	
		⟵	enc(RND.ICC || RND.IFD || K.ICC) || MAC || SW1-SW2: '9000'
	SM SELECT FILE [--EF]	⟶	
		⟵	SW1-SW2: '9000'
	SM READ BINARY [--EF]	⟶	
		⟵	enc(DATA) || MAC || SW1-SW2: '9000'

The test scenario shown in Table 1 corresponds to the read binary operation in the Operational state. Here, the read binary operation is conducted through a Secure Messaging after successful authentication. The emulator applet must complete all the required steps without exemptions. By using this test scenario, one can explore and check the implemented behaviour in the emulator applet, and if

necessary fixes that behaviour in accordance with the Indonesian national electronic ID.

Furthermore, the emulator applet was tested in the real system under supervisioned by Direktorat Jenderal Kependudukan dan Pencatatan Sipil. This test used real data and included application creation, data storage, and data retrieval. This test was important in order to measure the emulator applet's reliability as well as to determine that its behaviour is identical with the Indonesian national electronic ID. The results con-firmed that this approach had been successful. Figure 8 represents the results.

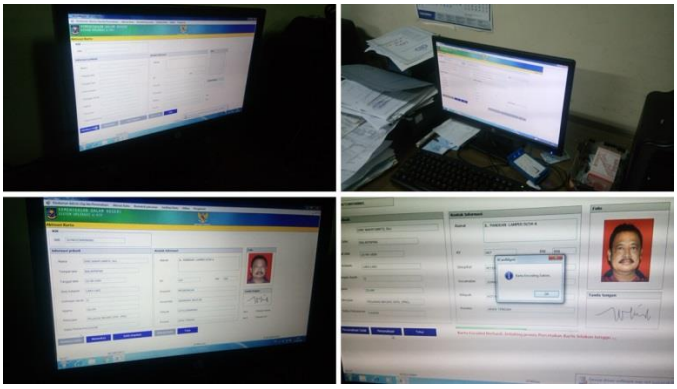

Figure 8. Test result using the real system

Finally, we evaluated the execution speed of the emulator applet (that is on a Java Card) and compared its performance with the Indonesian national electronic ID. For this purpose, we used 112 bytes of sample binary data, stored the data into each cards, and conducted the read binary operation to both of them in rotation. We performed 20 attempts and counted the elapsed time per operation. We obtained the following result as shown in Figure 9.

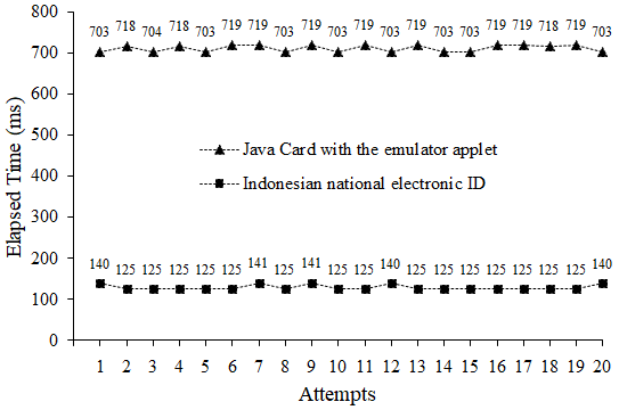

Figure 9. Emulator applet evaluation result

3. RESULTS AND DISCUSSION

The emulator applet had demonstrated its compatibility with the Indonesian national electronic ID. This gives an opportunity for many Java Card products to contribute in the national electronic ID program. Moreover, the emulator applet can attract interest of other Java Card applets to join in just one card. This condition will enable multiple applications to be ready inside the Indonesian national electronic ID, such as applica-tion for banking purposes, transportation, health services, and many more. We believe that this will be available in the near future.

Some limitations in the Indonesian national electronic ID can be overcome as well, such as the size of data exchanged. Current system only support up to 112 bytes of data that can be exchanged in one transaction. Java Card does not have this limitation. Any size of data will be accepted during transactions. Java Card is only limited by the structure of an APDU that only provides one byte of Lc and one byte of Le.

The emulator applet also opens up opportunities for the development of new features for the Indonesian national electronic ID. As a Java Card is programmable, then we can insert any additional functions, features, and applications by writing a suitable program. For instance, we can add a feature to go back to the Initialization state from the Operational state by using the new DEACTIVATE FILE command that currently does not exists, add a feature to delete a certain DF or EF by using the new DELETE FILE command that we consider necessary to have, and many more. This is another benefit that we will receive onwards upon this successful result.

Furthermore, this fruitful result can be extended to other program-based type of smart cards, such as BasicCard, or even to NFC smartphones that support card emulation. It will be interesting to hear other works in emulating the Indonesian national electronic ID that end up in success as well. The works on NFC smartphones in particular would possibly give a new definition related to the form factor of the next generation elec-tronic ID in Indonesia.

For supporting other works on this approach, we summarized some important components that need to be considered in order to create similar applet. We put them in a diagram as shown in Figure 10. The diagram describes what are the critical components (shown in circles), where to find the information about these components (shown in itemized), and what will be the output of these components. For example, to master the life cycle status, we have to follow [1] and so forth. By knowing this in advance, one can prepare accordingly.

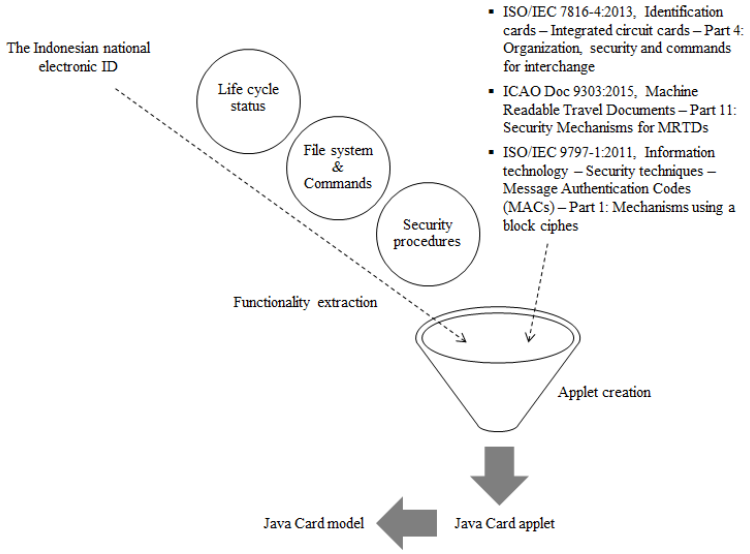

Figure 10. A diagram to describe the critical components and the output

One important aspect that needs to proceed is to reevaluate the execution speed of a Java Card that contains the emulator applet in comparison with the performance of the Indonesian national electronic ID. Various Java Card products must be tried out to sweep up the remainding probability space. Conceptually, Java Card inherits the Java platform components. And usually an interpreted program execution where codes have to be interpreted by a virtual machine is slower than a direct native program execution on the CPU. Therefore, this kind of performance must be taken into consideration, if in the future the Indonesian national electronic ID will also be used in a multiapplication environment that requires transaction speed.

4. CONCLUSION

A Java Card applet that emulates the behaviour of the Indonesian national electronic ID was successfully implemented. The emulator applet has been tested in the real system under supervisioned by Direktorat Jenderal Kependudukan dan Pencatatan Sipil, and now is ready to contribute in the national electronic ID program. We believe that Java Card is a highly feasible solution for the Indonesian national electronic ID to overcome future needs.

5. REFERENCES

[1] ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). (2013). ISO/IEC 7816-4:2013. *Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*. Switzerland: ISO/IEC.

[2] ICAO (International Civil Aviation Organization). (2006). ICAO Doc 9303 Part 1 Vol 2. *Machine Readable Travel Documents – Part 1: Machine Readable Passports – Volume 2: Specifications for Electronically Enabled Passports with Biometric Identification Capability*. Canada: ICAO.
[3] ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). (2011). ISO/IEC 9797-1:2011. *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*. Switzerland: ISO/IEC.
[4] J. P. Casanovas and G. V. Damme. (2011). *DESfire Emulation Using Java Card*. In Trustworthy Embedded Devices. Leuven, Belgium: Conference Publishing Services IEEE.