

SI5/M2II - IoT Security Lab

Smart Card Project

This project has been developed by: Ahmed Khalil Ben Sassi/ Yassin Anakar / Jamal El Alouani

Introduction:

This project focuses on the development of a secure communication system between terminal (CLI) coded in python and a smart card-Applet application that has been programmed in Java. The primary objective is to enable the generation of an RSA private and public key pair within the smart card, enhancing the security of data transmission. The terminal application will interact with the smart card, sending data for encryption and receiving the encrypted version along with the public key for verification purposes, of course all these scenarios won't be accessible unless the Pin code is entered the default Pin code is "Khalil" and we kept it at 6 characters length since Khalil have 6 characters, you can modify it eventually.

Summary:

- **Project Scope focusing on:**

Creating a full functional easy to use application that is divided into two parts:

- Developing a Smart Card Applet with java.**
- Programming a user-friendly interface to communicate with the card**

Smart Card Features:

Basic functionalities

- **PIN Protection:** Integrating a PIN code for user authentication and card protection.
- **Key Pair Generation:** Enabling the smart card to generate an RSA 512-bit key pair securely.
- **Data Signing:** Implementing the smart card's capability to sign data.
- **Public Key Transmission:** Facilitating the transmission of the smart card's public key to the terminal application on demand.

Payment Functionalities

- **Account credit:** Check how much credit you have on your card threw data received from the card itself.
- **History:** Check how many payments have been made with this card
- **Payment:** Static payment of 10 euros
- **Recharge:** Static card recharge of 10 euros

Terminal-Side Interaction:

- Our CLI that is lunched in terminal is very user friendly we handled many possible exceptions in it like when the card is not connected it show a message when the smart card reader is not connected there's another warning message, there's also a help functionality that show all the possible commands to enter from the basic functionalities to the payment methods.

The terminal provides functionalities that are communicating directly with the card such as

Default mode:

- **Login:** Authenticate user access, ensuring secure entry into the system.
- **Change Pin:** Allows users to update their personal identification number for enhanced security.
- **Logout:** Securely exits the user from the system, terminating the session.
- **Keygen:** Generates a unique cryptographic key pair for secure communications.
- **Public Key:** Retrieves and displays the public part of the cryptographic key pair.
- **Sign:** Creates a digital signature on data to verify its origin and integrity.
- **Verify:** Confirms the authenticity and integrity of signed data using the corresponding public key.

Payment mode:

- **Info:** Displays current account information and settings.
- **History:** Provides a log of past transactions for review and record-keeping.

- **Pay:** Enables users to make payments or transfer funds.
- **Charge:** Allows users to add funds to their account for future transactions.

There's also way to navigate easily in the modes and the CLI's commands such as "back", "help", "exit".

Architecture

| Instruction (INS) | Functionality Description | Return Values |
|-------------------|---|---|
| 0x00 | Balance Inquiry: Authenticates PIN and returns the current balance. | Balance byte: 90 00 on success, 69 82 if PIN not verified. |
| 0xC1 | Make Payment: Deducts 10 euros from balance after PIN verification. | Updated balance: 90 00 if successful, 69 82 if PIN not verified, 69 85 if insufficient balance. |
| 0xD1 | Charge Card: Adds 10 euros to balance, requiring PIN verification. | Updated balance: 90 00 if successful, 69 82 if PIN not verified. |
| 0xA0 | PIN Verification: Checks PIN against stored value. | 90 00 if correct, 69 82 if incorrect. |
| 0xA1 | Disconnect PIN: Resets PIN session for security. | 90 00 on success, 69 82 if already disconnected. |
| 0xA2 | Change PIN: Allows PIN modification, followed by session reset. | 90 00 on success, 69 83 if PIN changed, 69 82 if incorrect PIN. |
| 0xB0 | Key Pair Generation: Generates RSA 512-bit key pair. | 90 00 on success, 69 82 if PIN not verified. |
| 0xB1 | Send Public Key: Sends RSA public key to terminal. | Public key data: 90 00 on success, 69 82 if PIN not verified. |

| | | |
|----------------|---|--|
| 0xB2 | Sign Message: Signs a message using private key. | Signature data: 90 00 on success, 69 82 if PIN not verified, 69 86 if message too long. |
| 0xB3 | Send Digital Signature: Transmits the digital signature. | Signature data: 90 00 on success, 69 82 if PIN not verified. |
| Default | Exception Handling: Manages unsupported INS codes. | 6D 00 for invalid INS. |

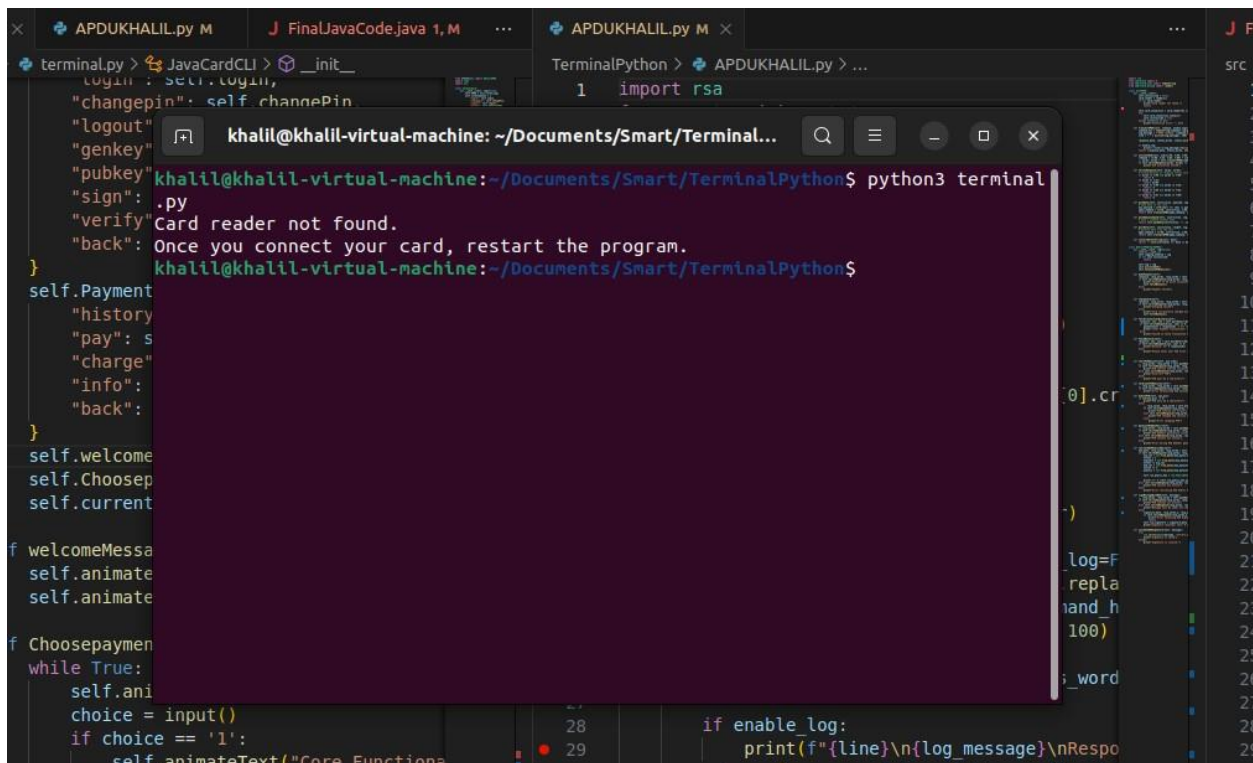
Libraries:

- In our Smart Card Project, we utilized the smartcard library in Python for interfacing with the card reader and managing APDU communications, and the rsa library for cryptographic operations. On the Java side, the javacard.framework and javacard.security packages were essential for smart card applet development, including PIN management and RSA encryption.

The terminal side application:

1. Exception Handling:

If the card reader is not connected to the host machine a message asking to insert the card will appear.



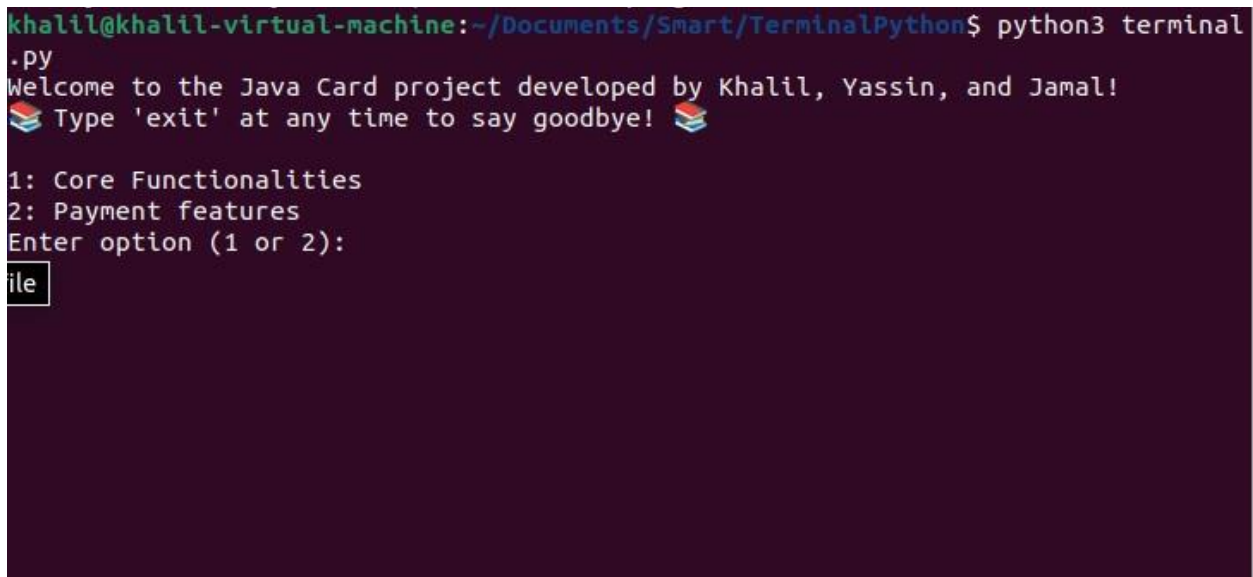
```
terminal.py > JavaCardCLI > _init_
login : self.login,
logout : self.logout,
"changePin" : self.changePin,
"genkey" :
"pubkey" :
"sign" :
.verify :
"back" : Once you connect your card, restart the program.
}
self.Payment
    "history"
    "pay" : s
    "charge"
    "info" :
    "back" :
}
self.welcome
self.Choosep
self.current

f welcomeMessa
self.animate
self.animate

f Choosepaymen
while True:
    self.ani
    choice = input()
    if choice == '1':
        self.animateText("Core Functiona
        28
        29
        if enable_log:
            print(f"{line}\n{log_message}\nRespo
            28
            29
```

2. Smart Card Connected

After inserting the card and running the terminal side application, there will be a welcoming message with some information and choices to pick which mode of functionalities will be used.



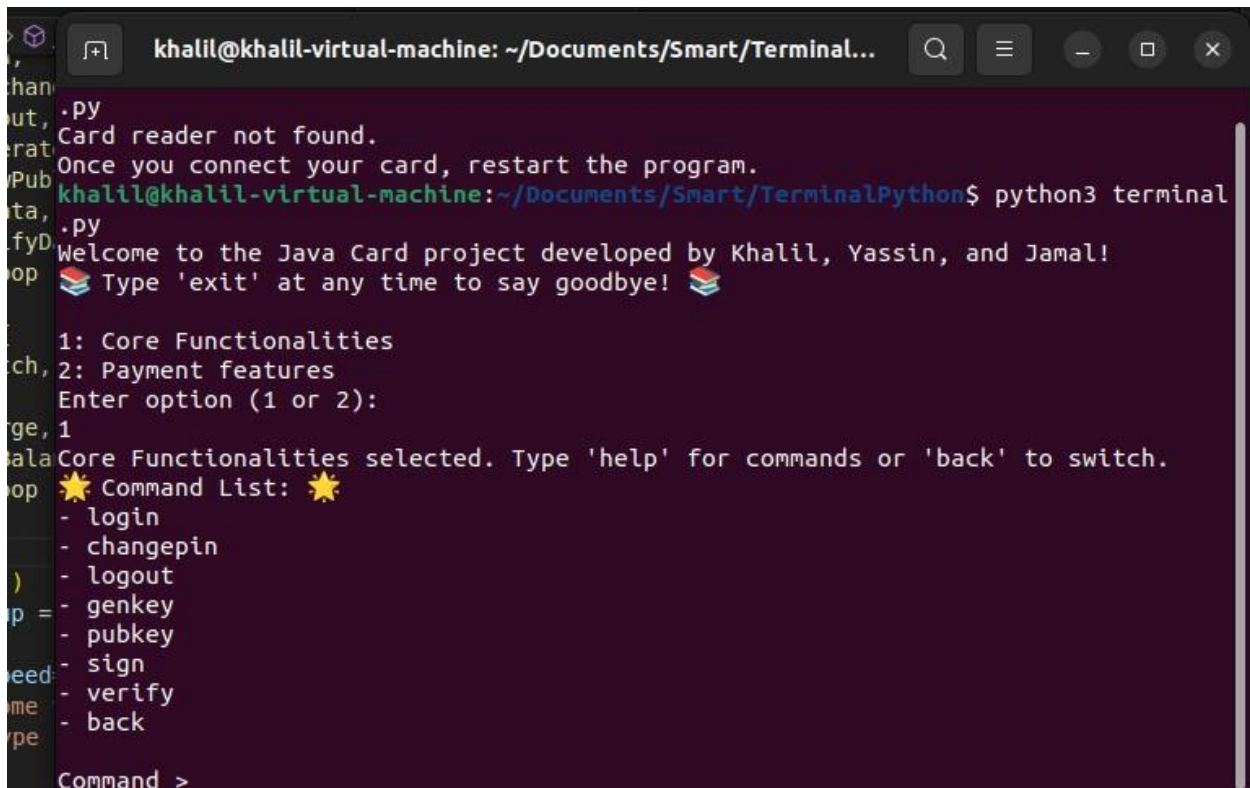
```
khalil@khalil-virtual-machine:~/Documents/Smart/TerminalPython$ python3 terminal
.py
Welcome to the Java Card project developed by Khalil, Yassin, and Jamal!
📖 Type 'exit' at any time to say goodbye! 📖

1: Core Functionalities
2: Payment features
Enter option (1 or 2):
1
```

3. Functionalities :

Once the user selects "**1: Core Functionalities**" the application displays a list of commands related to the core features of the Java Card project.

The list includes commands for user authentication and card management, such as 'login,' 'changepin,' 'logout,' along with commands to handle the cryptographic functions like 'genkey,' 'pubkey,' 'sign,' 'verify,' and an option to go 'back' to the previous menu:



```
khalil@khalil-virtual-machine: ~/Documents/Smart/TerminalPython$ python3 terminal.py
Card reader not found.
Once you connect your card, restart the program.
khalil@khalil-virtual-machine: ~/Documents/Smart/TerminalPython$ python3 terminal.py
Welcome to the Java Card project developed by Khalil, Yassin, and Jamal!
📖 Type 'exit' at any time to say goodbye! 📖

1: Core Functionalities
2: Payment features
Enter option (1 or 2):
1
Core Functionalities selected. Type 'help' for commands or 'back' to switch.
🌟 Command List: 🌟
- login
- changepin
- logout
- genkey
- pubkey
- sign
- verify
- back

Command >
```

4. Initiating Login :

The default pin is Khalil, so just login with Khalil > login khalil


```
khalil@khalil-virtual-machine: ~/Documents/Smart/TerminalPython$ python3 terminal
.py
Welcome to the Java Card project developed by Khalil, Yassin, and Jamal!
📦 Type 'exit' at any time to say goodbye! 📦

1: Core Functionalities
2: Payment features
Enter option (1 or 2):
1
Core Functionalities selected. Type 'help' for commands or 'back' to switch.
🌟 Command List: 🌟
- login
- changepin
- logout
- genkey
- pubkey
- sign
- verify
- back

Command > login khalil
PIN session started successfully

Command > 
```

5. **Attempted Unauthorized Login** we attempted to login with a false password (unauthorized login) to demonstrate the authentication mechanism.

```
khalil@khalil-virtual-machine: ~/Documents/Smart/Terminal...
1: Core Functionalities
2: Payment features
Enter option (1 or 2):
1
Core Functionalities selected. Type 'help' for commands or 'back' to switch.
★ Command List: ★
- login
- changepin
- logout
- genkey
- pubkey
- sign
- verify
- back

Command > login khalil
PIN session started successfully

Command > logout

Command > login khalol
Incorrect PIN

Command > 
```

6. Changing PIN :

Now we tried to change the PIN from “khalil” to “yassin” and we see that when we change the PIN the session is terminated automatically:

```
login: self.login,
"changepin": self.chan
"logout": self.logout,
"genkey": self.generat
"pubkey": self.showPub
"sign": self.signData,
"verify": self.verifyD
"back": self.exitLoop
- login
- changepin
- logout
- genkey
- pubkey
- sign
- verify
- back

lf.PaymentMethods = {
"history": self.fetch,
"pay": self.pay,
"charge": self.charge,
"info": self.fetchBala
"back": self.exitLoop
Command > login khalil
PIN session started successfully
Command > logout
Command > login khalol
Incorrect PIN
Command > login khalil
PIN session started successfully
Command > changepin yassin
PIN changed and session terminated
Command >

lf.welcomeMessage()
lf.ChoosepaymentFonc()
lf.currentCommandGroup =
lf.welcomeMessage(self, speed
lf.animateText("Welcome
lf.animateText("🌟 Type
oosepaymentFonc(self):
```

7. Key Pair Generation:

we initiated the task sequence with a secure PIN session to authenticate users. Upon successful login, we generated an RSA key pair essential for the encryption of communications. We then retrieved and displayed the public key, enabling others to encrypt messages or verify signatures. Subsequently, we created and verified a digital signature, demonstrating the smart card's capabilities in signing and ensuring data integrity.

```
Command > login yassin
PIN session started successfully

Command > changepin khalil
PIN changed and session terminated

Command > genkey
PIN session not active

Command > login khalil
PIN session started successfully

Command > genkey
RSA KeyPair generation successful

Command > pubkey
-----BEGIN RSA PUBLIC KEY-----
MEgCQQDAyy7rLHKy5/iMnmRDCXMWt7a/cz5G3gQ1sEro9B+bsdz+N8m0Il9j7VXx
HlvOWeSdX1Zxt7MgxJtHcxCb+WCNAgMBAAE=
-----END RSA PUBLIC KEY-----

Command > sign khalil
Signature received:
-írÐiSöp          Å,°
-éû'û~iNe1Ëþ(·W²"·í

Command > verify khalil
Signature is valid

Command >
```

8. Payment functionality

We Interacts with the payment features of our Java Card application. We successfully log in and navigate to the payment section, starting with a balance of 50 euros. Our team performs a transaction, deducting 10 euros, and later recharges the same amount.

Further payments are processed, and we conclude by reviewing the transaction history, which confirms three completed payments.

```
Command > login khalil
PIN session started successfully

Command > back

1: Core Functionalities
2: Payment features
Enter option (1 or 2):
2
Payment Features selected. Type 'help' for commands or 'back' to switch.
★ Command List: ★
- history
- pay
- charge
- info
- back

Command > info
Balance: 50

Command > pay
Payment of 10 euros successful
Balance: 40

Command > charge
Card successfully charged with 10 euros
Balance: 50

Command > pay
Payment of 10 euros successful
Balance: 40

Command > pay
Payment of 10 euros successful
Balance: 30

Command > history
Total Payment Transactions: 3

Command > 
```

In our Java Card project, we have implemented two methods for compiling and managing the smart card application lifecycle: using a shell script and a Makefile.

While both methods assure the same output, the Makefile approach is often considered more professional due to its standardization and efficiency in handling complex build environments. Both methods coexist in our project, giving us the flexibility to choose the most suitable tool for different development scenarios.

1-Using a shell script :

We developed a **shell script** to streamline the setup and execution of our Java Card project. The script begins by starting the PC/SC Smart Card Daemon, ensuring communication with the smart card reader.

We then compile our Java Card application with specific source and target compatibility, followed by using the Java Card Converter tool to convert the compiled Java class files into a format suitable for loading into Java Cards(.cap)

To maintain a clean state on the smart card, we execute a **GPShell** script to delete any existing CAP files. Afterward, we upload the newly generated CAP file onto the smart card. The final step in our process involves running a Python script that initiates the terminal application, allowing us to interact with the Java Card directly.



```
1 #!/bin/bash
2 sudo service pcscd start
3
4 javac -source 1.2 -target 1.1 -g -cp jc211_kit/bin/apl.jar -d src/class src/FinalJavaCode.java
5
6 java -classpath ./jc211_kit/bin/converter.jar:. com.sun.javacard.converter.Converter -verbose -exportpath ./jc211_kit/api_export_files:source -classdir ./src/class -applet 0xa0:0x0:0x0:0x02:0x3:0x1:0xc:0x6:0x1:0x2 FinalJavaCode source 0xa0:0x0:0x0:0x02:0x3:0x1:0xc:0x6:0x1 1.0
7
8 gpshell ./gpshellscript/deletefromcard
9 gpshell ./gpshellscript/uploadtocard
10
11 python3 TerminalPython/terminal.py
```



```

Unwrapped response <-- 9000
Command --> 80E80005EF00400052001800000000209005A00890025000000000309008100520007000000000409008A0089002800000
0000509008700520015000000000060900CE0052001500000000070900E500890048000000000809012F0089005400000000090901850089
001A000000000A0901A1008E000B00000000070101AE009100C10000000000270050005000520054005600560058005C005E00620052005
2FFF006400670052006BFFF006F00720075007A007E00560083FFF005200520087005600520052008900520089005200890089001
3081100241018006B4B443014006B4B4440244033410056811000568
Wrapped command --> 84E80005F876A9C29DAE7A387391FFCCD91ACD352867ACFA2CC9BBA44FF5F3FED475CD2AF1424121B51D2999194
6F12ACB5B87C0BA7B85877BDAF9B83F563220ED9E493196DF283D309AA733181C8B9467FD1AABAAB9DA5F9BDABC7574D99CE13E6FD65FDE
08D74B160DB2A9AC81BD04540AA7FB4A3971C7421911E51095A8146FB6DB4403EA3546481A007EADC00534DB93FEEDA4781539AEA00D46B
9834E57CD68897E918D335FF0D9D54578E4EEC8E3CE4A5ECD25E28BD6824B521519A13A72EE87B7BDD840E015EA4DB5E0EF8328EA860D68
54EA7D009B1DED936FCEE76EBE47B673AE803F521CF52B48768AC92A50DF9A8EA40D269FB26DAF53
Response <-- 9000
Unwrapped response <-- 9000
Command --> 80E8800628103004B44403441007326810F005681020076810031006B44B440120076800AB1004B431066800A100
Wrapped command --> 84E8800638E631920EB790E074D9588168ED6E36D1B27AE903206DF2D7F67330E1B512A1EFB051238A9874736CF
E03A2AF0E1612E672134B0F7109365F00
Response <-- 009000
Unwrapped response <-- 009000
Command --> 80E60C00290A0A0000006203010C06010BA00000006203010C06010208A00000006203010C060102010002C9000000
Wrapped command --> 84E60C003845A27B90A8BD9519FDB05243ACDAF323CB0F0E8BE6DC3A4AA5B71332CEE40217756AA301E5D7462
249D9105CF386C666A9D591E23FCC9700
Response <-- 009000
Unwrapped response <-- 009000
command time: 3272 ms
card_disconnect
command time: 56 ms
release_context
command time: 2 ms
Welcome to the Java Card project developed by Khalil, Yassin, and Jamal!
👋 Type 'exit' at any time to say goodbye! 🌈

1: Core Functionalities
2: Payment features
Enter option (1 or 2):

```

2-Using a Makefile :

We created a Makefile used in our Java Card project to automate compilation, conversion, deletion of old applets, uploading of new applets, and running the terminal interface. This Makefile simplifies the process into single-command operations.

```

Makefile
~/Documents/3met

1 all: compile convert delete_applet upload_applet run_terminal
2
3 compile:
4     @sudo service pcscd start
5     @echo "Compiling Java source..."
6     @javac -source 1.2 -target 1.1 -g -cp ./jc211_kit/bin/apl.jar -d ./src/class ./src/FinalJavaCode.java
7     @echo "Compilation complete or with warnings..."
8
9 convert:
10    @echo "Converting to JavaCard format..."
11    @java -classpath ./jc211_kit/bin/converter.jar: con.sun.javacard.converter.Converter -verbose -exportpath ./jc211_kit/apl_export_files:source -
classdir ./src/class -applet 0xa0:0x0:0x0:0x0:0x02:0x3:0x1:0xc:0x6:0x1:0x2 FinalJavaCode source 0xa0:0x0:0x0:0x0:0x02:0x3:0x1:0xc:0x6:0x1 1.0
12
13 delete_applet:
14    @echo "Deleting applet from card..."
15    @gpshell ./gpshellscript/deletefromcard
16
17 upload_applet:
18    @echo "Uploading applet to card..."
19    @gpshell ./gpshellscript/uploaddtocard
20
21 run_terminal:
22    @echo "Running Terminal Python script..."
23    @python3 ./TerminalPython/terminal.py
24
25 .PHONY: compile convert delete_applet upload_applet run_terminal all
26

```

```
khalil@khalil-virtual-machine: ~/Documents/Smart
khalil@khalil-virtual-machine:~/Documents/Smart$ make
[sudo] password for khalil:
Compiling Java source...
warning: [options] bootstrap class path not set in conjunction with -source 1.2
warning: [options] source value 1.2 is obsolete and will be removed in a future release
warning: [options] target value 1.1 is obsolete and will be removed in a future release
warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
4 warnings
Compilation complete or with warnings...
Converting to JavaCard format...

Java Card 2.1.1 Class File Converter (version 1.1)
Copyright (c) 2000 Sun Microsystems, Inc. All rights reserved.

parsing /home/khalil/Documents/Smart/src/class/source/FinalJavaCode.class
converting source.FinalJavaCode
parsing /home/khalil/Documents/Smart/jc211_kit/api_export_files/javacard/framework/javacard/framework.exp
parsing /home/khalil/Documents/Smart/jc211_kit/api_export_files/javacard/security/javacard/security.exp
writing /home/khalil/Documents/Smart/src/class/source/javacard/source.exp
6800AB1004B431066800A100
Wrapped command --> 84E8800638EAF59E8E4A6AB7E4D990E92F5491047F59CC7A7446139E1C1F
A53EC39704EA8CABAF9B0EDF64E690493AA15519D3B34BAF8A2F83DE486EA300
Response <-- 009000
Unwrapped response <-- 009000
Command --> 80E60C00290A0A00000006203010C06010BA000000006203010C0601020BA0000000062
03010C060102010002C9000000
Wrapped command --> 84E60C00387F2A5F1DA3ABCD57765CFBC86494519BEFFF549B560B4BC8DA
6A716EF0CED18EF5749AF40077BA4D6154127BCB008439114C158BF700828600
Response <-- 009000
Unwrapped response <-- 009000
command time: 3266 ms
card_disconnect
command time: 55 ms
release_context
command time: 1 ms
Running Terminal Python script...
Welcome to the Java Card project developed by Khalil, Yassin, and Jamal!
📖 Type 'exit' at any time to say goodbye! 📖

1: Core Functionalities
2: Payment features
Enter option (1 or 2):
```

Conclusion:

In conclusion, the Smart Card Project fulfils the specified requirements and demonstrates a well-designed and tested system for secure communication. The combination of PIN protection, RSA key pair generation, data signing, and public key transmission ensures a reliable and secure communication framework between the terminal and the smart card, it was such a pleasure for it and quite the delight to understand a component that we are using in our daily lives and programming us ourselves.