UNIVERSITY OF CAGLIARI

FACULTY OF ENGINEERING AND ARCHITECTURE

MASTER DEGREE IS COMPUTER ENGINEERING, CYBERSECURITY AND ARTIFICIAL INTELLIGENCE

Network Security Project

Author
Nicola Deidda
70/90/00358

Author
Antonio Campus
70/90/00344

June 24, 2024

Abstract

The main goal of this project is to develop an infrastructure capable of emulating how NFC-based payments work. The core of this implementation will be the two mobile applications: one will emulate the Customer, the other the Merchant. Then, different servers will be deployed to simulate how the banks' network works.

Contents

Chapter 1

Introduction

The proliferation of online commerce, combined with the pervasive presence of smartphones, has started a new era within the domain of payment solutions.
This evolution has opened up multiple possibilities for businesses looking to expand their reach and tap into new markets. As such, it has become increasingly important for companies to stay apprised of the latest developments in the field of payment processing to remain competitive.

The outbreak of the pandemic has been accompanied by a state of emergency across the globe. The risks associated with the spread of the virus through physical contact have necessitated the development of contactless solutions to mitigate transmission.
As such, recent years have witnessed a surge in the development of contactless solutions in response to these emerging concerns.

Given the increasing interest, Near Field Communication (NFC) technology has become a cornerstone in the realm of contactless payments, transforming the way individuals conduct financial transactions.
Operating on the principle of short-range wireless communication, NFC enables secure and swift data exchange between compatible devices, laying the foundation for seamless and convenient contactless payment experiences.

1.1 Contribution

This report aims to describe the design of a NFC-based contactless solution. Given the current state of the art, the proposed system will demonstrate how an NFC-based payment works, from the configuration of a credit card to its usage.

1.2 Project Outline

The present report is organized into four chapters:

1. The first describes the theoretical foundations and the current state-of-the-art above which the project will be developed.
2. The second describes the implementation of our solutions, in particular focusing on the NFC communication between the devices.
3. Then, the third provides a description of the proof-of-concept and guidelines about how to deploy it.
4. The fifth, the last, addresses some open issues related to our implementation.

1.3 Acknowledgement

This report focuses on implementing a proof-of-concept on how a NFC-based payment solution works.
Given the academic nature of this project, some implementation aspects, particularly concerning the current standards, will be simplified. When it is done, it will be emphasised.

Chapter 2

Analysis

To accurately comprehend the initial basis for the development of our solution, it is essential to examine two critical aspects:

1. The reference scenario, which delineates the conditions under which the system will operate.
2. The current state of the art, to understand the foundation and starting point of our solution.

2.1 Reference scenario

This subsection aims to describe the scenario where the proposed implementation should operate.
In addition, keeping into account the academic nature of this project, this paragraph contains constraints on the working environment.

To pay for a good or a service, the Customer must pair its device with the Point Of Service (POS) of the Vendor. The connectivity of both the Client's device and the Vendor's Point Of Service is considered as guaranteed.

Then, the project supposes that the Client's device runs the latest version of Android as the operating system. This choice will enforce compatibility with the proposed implementation.

Furthermore, the unique payment network to be implemented will be a simulated one. This deliberate choice is made with the awareness of the

need to offer the most universally applicable solution, independent of specific commercial implementations.

2.2 State of the art

The current state of the art in the fields of NFC payments is represented by different Mobile Wallets, which allow users to securely store multiple payment methods (e.g.: credit cards) in their smartphones.

These Wallets enforce security combining secure execution and secure access procedures. In the following subsections, these concepts will be described in depth.

Secure Execution

Secure execution refers to the process of running software applications in a manner that ensures the confidentiality, integrity, and availability of data and resources. It can be enforced in different ways:

- Using Secure Element (SE): It is a tamper-resistant, dedicated hardware component commonly situated in mobile devices. Its purpose is to securely retain sensitive data and execute cryptographic operations within an isolated environment, distinct from the device's primary processor and operating system.
- Using Host Card Emulation (HCE): This technology enables mobile devices to simulate contactless smart cards without relying on specialized hardware SE. The subsequent section will provide a detailed analysis of this solution.
- Using Trusted Execution Environment (TEE): a secure area within a processor that ensures the confidentiality and integrity of sensitive data and processes. On TEE, trusted applications can run securely, performing sensitive operations independently from the main operating system.

Given the definition above, Table 2.1 lists the advantages and disadvantages of each possible choice.

Additionally, alternative security mechanisms can be implemented to uphold payment security. Specifically, when handling sensitive data, such as credit card numbers or CVVs, the utilization of Tokenization serves to

Technology	Pros	Cons
Secure Element	- Hardware-based security - High level of tamper resistance - Isolated from main processor	- Costly to implement - Limited flexibility for updates - Dependency on hardware integration
Trusted Execution Environment	- Isolation from main processor - More flexible than Secure Element - Better integration with OS	- Complexity in implementation - Potential performance overhead
Host Card Emulation	- Software-based - Flexibility for updates - Lower implementation cost - No dependency on specific hardware	- Reliance on network - Potential performance issues

Table 2.1: Pros and Cons of Secure Element, Trusted Execution Environment (TEE), and Host Card Emulation (HCE)

anonymize the data by substituting them with surrogates.

In pursuit of broad applicability and to circumvent reliance on specialized hardware that may be scarce or arduous to replicate, our approach will be grounded in Host Card Emulation (HCE). Consequently, the subsequent subsection provides an extensive analysis of this technology.

2.2.1 Host Card Emulation (HCE)

Smart cards, also referred to as chip cards, may come in the form of contact cards, contactless cards, or a combination of both.

Contact smart cards are equipped with SIM-like contact chips: the chip uses a character or block-level protocol based on ISO 7816 [1], with individual applications defining command and response protocols based on APDU (Application Protocol Data Unit) messages, or application programming data units, which will be described in the next section. This standard is the ISO 781 is ISO 14443 [2], also known as ISODEP.

Contactless smart card emulation is enabled by the card emulation feature of NFC, also known as CE mode.

Before the release of Android KitKat [3], NFC card emulation was based on a hardware secure element connected to the device's NFC controller. The secure element could be part of the device's SIM card or other embedded component. Regardless, it provides a secure environment for running and interacting with applets using a contactless smart card reader, such as

a payment terminal (e.g. POS).

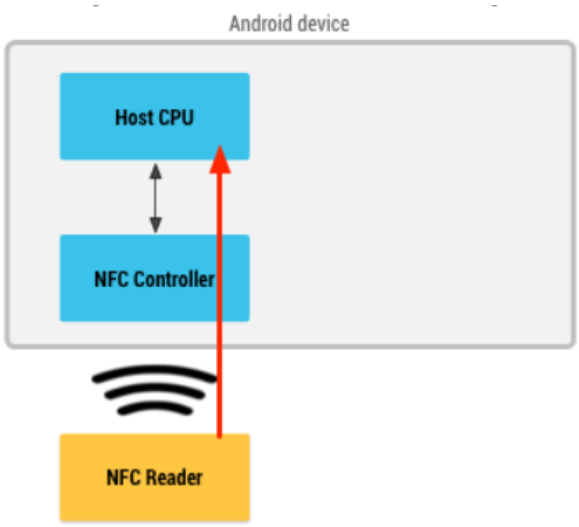


Figure 2.1: Host Card Emulator

Android KitKat introduced a new feature called Host Card Emulation (HCE) [4] which allows Android applications running on the host processor to directly control interactions in Card emulator Card Emulation mode: it communicates directly with smart card readers without using hardware secure element components.
This feature provides compatibility with the Hardware Secure Element, enabling one application to utilize "Host" card emulation (HCE) while another utilizes "Offhost" card emulation (Hardware Secure Element).

Host Card Emulation (HCE) technology relies on a cloud-based Secure Element, where data is stored and managed in the cloud rather than on the smartphone. In this scenario, smartphones are used solely to emulate the smart card and to store the necessary tokens, which is why they are also referred to as "cardholders."

2.2.2 Tokenization

Tokenization denotes a process aimed at replacing sensitive data with non-sensitive data, referred to as a token, that cannot be exploited to retrieve or infer the original data.

The EMV [5] has defined the technical specification [6] for implementing the tokenization process.

The mapping between original data and tokens is kept in secure storage, under the responsibility of the Tokenization System.
This entity must be logically isolated and segmented from data processing systems and applications that will process or store sensitive data replaced by tokens.
Only the Tokenization System must tokenize data to create tokens or detokenize them to redeem sensitive data.

Figure 2.2 schematizes how the tokenization flow works.

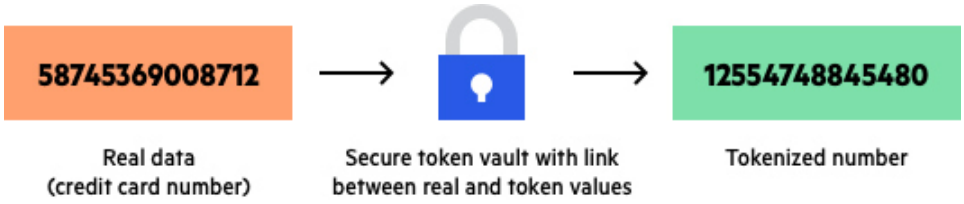


Figure 2.2: Schema of the Tokenization flow

The fundamental concept of tokenization involves substituting the Primary Account Number of a credit card with its tokenized counterpart.
Within the payment tokenization ecosystem, two primary implementations exist, contingent upon the identity of the token requestor.
If the merchant serves as the token requestor, as illustrated in Figure 2.3, the cardholder still transmits the Primary Account Number to the merchant, who subsequently solicits the token from the token provider.

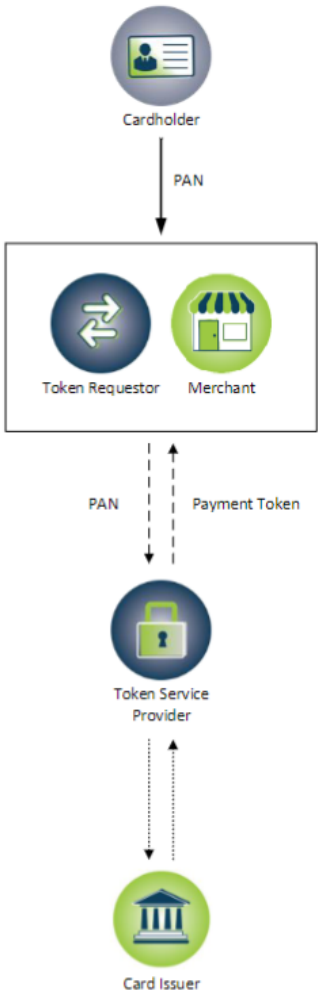


Figure 2.3: Merchant Token Requestor, *EMVCo-Payment-Tokenisation-Specification-Technical-Framework*

The second type, which is the focus of this report, pertains to tokenization where the token is stored in the cardholder's device and transmitted to the Merchant during the transaction. An overview of this process is presented in Figure 2.4.

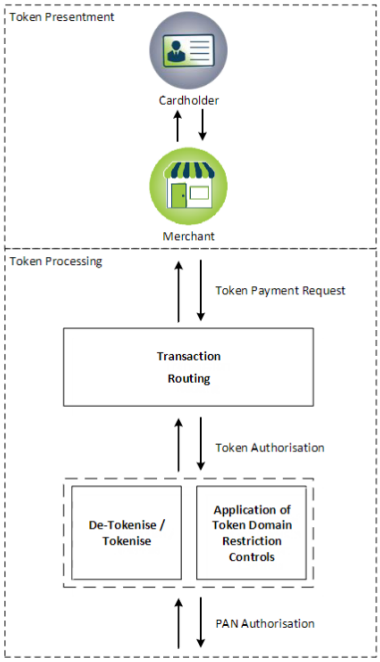


Figure 2.4: Cardholder Token Requestor, *EMVCo-Payment-Tokenisation-Specification-Technical-Framework*

2.2.3 Interface with point-of-sale terminals

The objective of this section is not to provide an exhaustive description of the Point Of Service ecosystem and its protocols, but rather to offer a broad overview of the communication between the Host Card Emulation (HCE) and the Point Of Service reader.

Every implementation of HCE must be able to interact with the common payment system like Mastercard and Visa.
The international standard EMV has written many complex protocols on the Contactless transaction.
The contactless standard consists of three general books, Book-A [7], Book-B [8], Book-D [9]. These books describe the general mechanisms of contactless EMV payment methods.

In addition, EMV defines a set of specifications, called Kernels. Each payment scheme (e.g., Visa, MasterCard, American Express) has its specific

kernel.

Figure 2.5 shows the general architecture of a point-of-sale system.
As we can see the POS consists of two components, the Terminal and the Reader.
Of particular interest within the scope of this document is the reader, who engages with Host Card Emulation via Near Field Communication (NFC).

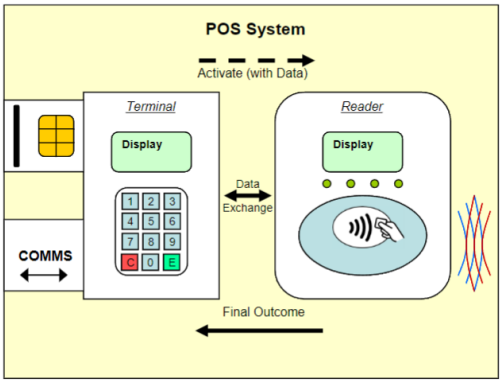


Figure 2.5: POS System Descriptive Model, *EMV,Book A: Architecture and General Requirements*

According to[7], the reader is responsible for:

- communication with contactless cards
- application selection and kernel activation
- running the contactless kernel applications
- CVM selection
- creation of a Final Outcome and parameter set
- management of a "no card response" timeout
- management of the contactless field

Figure 2.8 shows the reader in more detail.
As mentioned before, the HCE and the reader communicate using ISO 14443 [2] and ISO-7816-4 [1]. The PDU on this standard are called Application Protocol Data Unit, APDU.
There are two types of APDU [10]:

- C-APDU: command APDU, send by the reader.

- R-ADPU: response APDU, send by the HCE.

The command ADPU [10] comprises two main fields, a Header and a Trailer.

CLA	INS	P1	P2	Lc	Data	Le
Header				Trailer		

Field	Description
CLA	Class byte 0x00
INS	Instruction byte 0xA4:Select Command 0xB2:Read Record Command
P1	Parameter 1 byte The value and meaning depends on the instruction code (INS).
P2	Parameter 2 byte The value and meaning depends on the instruction code (INS).
Lc	Number of data bytes send to the card. With the Smart Card Shell the value of Lc will be calculated automatically. You don't have to specify this parameter.
Data	Data byte
Le	Number of data bytes expected in the response. If Le is 0x00, at maximum 256 bytes are expected.

Figure 2.6: Command ADPU, *https://www.openscdp.org/scripts/tutorial/emv/reademv.html*

As shown in Figure 2.8, the first command sent by the reader is *Select PPSE*: this command lets the reader know what payment systems are available on the HCE or in the card.
The header of this command is made up of the following combination of bytes:

- CLA field = 0x00
- INS field = 0xA4, select command, Table-38 [1]
- PI field = 0x04, select by name, Table-39 [1]
- P2 field = 0x00, first or only occurrence, Table-40 [1]

On the other hand, the trailer starts with the size of the data sent, the actual data, and the length of the data expected in the response.
The header's P1 field indicates that the selection must start with the name, so the data field must contain the byte of the string indicating the application that the POS wants to select. In case of proximity, the string must be *2PAY.SYS.DDF01* [11]. So the trailer will be:

- Lc field, len of *2PAY.SYS.DDF01* byte string
- Data field, bytes of *2PAY.SYS.DDF01*
- Le field,0x00, maximum byte expected

Figure 2.7 shows an example of a payload.

```
00a404000e325041592e5359532e444446303100
```

Figure 2.7: select PPSE bytes

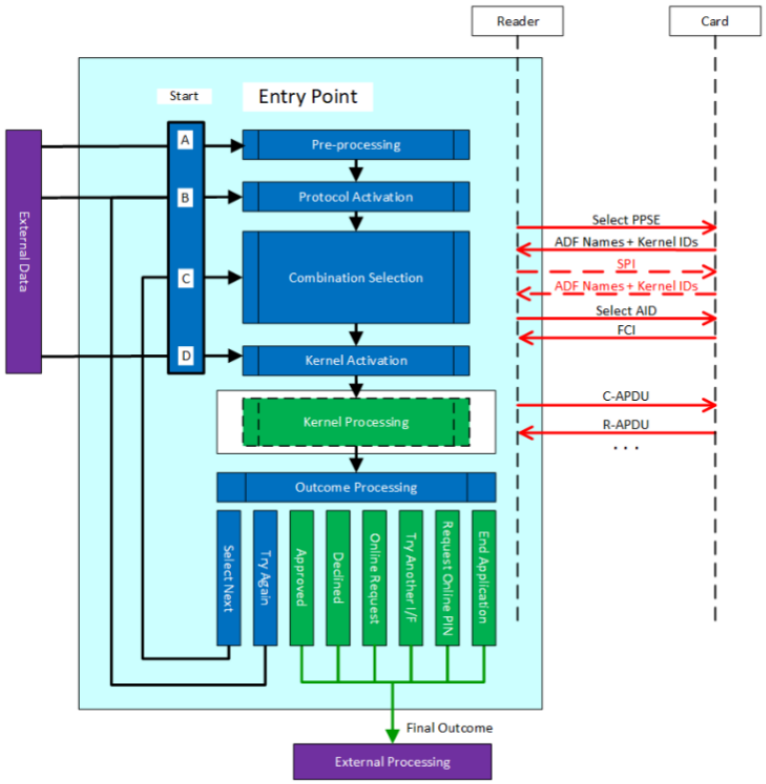
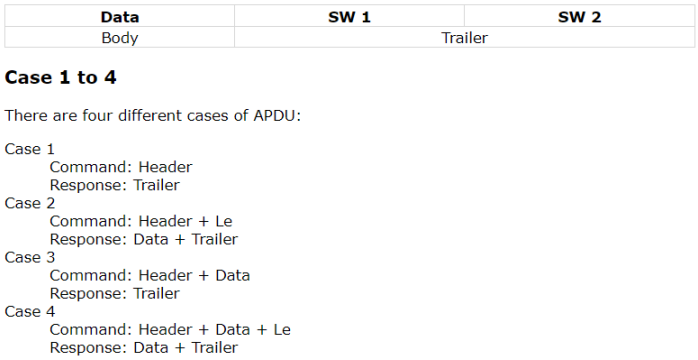


Figure 2.8: Entry Point High-Level Architecture, *EMV,Book B: Book A: Entry Point*

After the HCE receives the *SELECT PPSE* command, it must communicate to the reader the payment methods supported, by sending back the R-ADPU.

Figure 2.9 shows the R-ADPU: it consists of two parts, the actual data and the status byte.



Data	SW 1	SW 2
Body	Trailer	

Case 1 to 4

There are four different cases of APDU:

Case 1
Command: Header
Response: Trailer
Case 2
Command: Header + Le
Response: Data + Trailer
Case 3
Command: Header + Data
Response: Trailer
Case 4
Command: Header + Data + Le
Response: Data + Trailer

Figure 2.9: Response ADPU, *https://www.openscdp.org/scripts/tutorial/emv/reademv.html*

Figure 2.10 shows an example of a response payload.

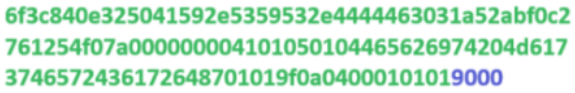


Figure 2.10: Response ADPU

By utilizing the EMV decoder [12], it is possible to systematically parse each field within the response payload.

As previously indicated, each application (payment system) is distinguished by its unique Application Application Identifier (AID): the response comprises structures known as File Control Information (FCI), which encapsulate the payment systems along with their corresponding Application Application Identifier.

Figure 2.12 illustrates the parsed payload: it can be observed that, in this instance, there is only one accepted payment system, identified by the *A0000000041010* Application Application Identifier, associated with *Mastercard International* [11].

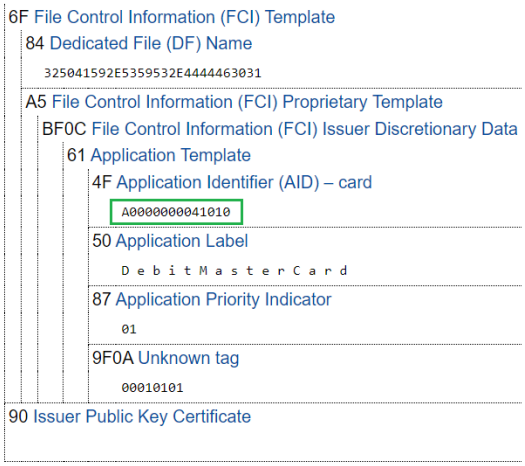


Figure 2.11: Response ADPU

Upon reading the available payment systems, the Point Of Service device will proceed to select one of them by transmitting the *SELECT AID* command. The header of this command remains consistent.
Within the data section, the payload includes the size of the AID string and the AID itself. Below is an illustrative example of this payload format.

00a4040007a000000004101000

Figure 2.12: Select AID

Following the handshake, the reader and the Host Card Emulation (HCE) initiate communication utilizing the selected payment scheme. This communication relies on the proprietary protocol outlined in the corresponding kernel books.

2.2.4 The HCE based payment system from 10'000 feet

In the preceding sections, the primary components of an HCE-based payment system have been meticulously elucidated.

In this section, we will delve into the interplay among these components

during a hypothetical transaction, with the cardholder serving as the token requestor.

In a tokenization scenario, two main phases exist, both extensively detailed in the *Payment Tokenisation – A Guide to Use Cases* [13].

- **Issuance phase**, when the cardholder receives the token from the Token Service Provider (TSP).
- **Transaction Flow**, when the token is sent to the merchant and forwarded to the exciting ecosystem payment, to complete the payment.

Issuance phase

Figure 2.13 shows the issuance phase, consisting of nine steps.
The first two are related to the setup of the cardholder, while the other are described below.

3. The cardholder initiates a Token Request to the Token Service Provider by using the PAN of his/her credit card.
4. The Identity and Verification subphases start
5. The Card Issuer responds to the Token Service Provider with its ID&V result
6. The Identity and Verification subphase end
7. The cardholder application receives the token.

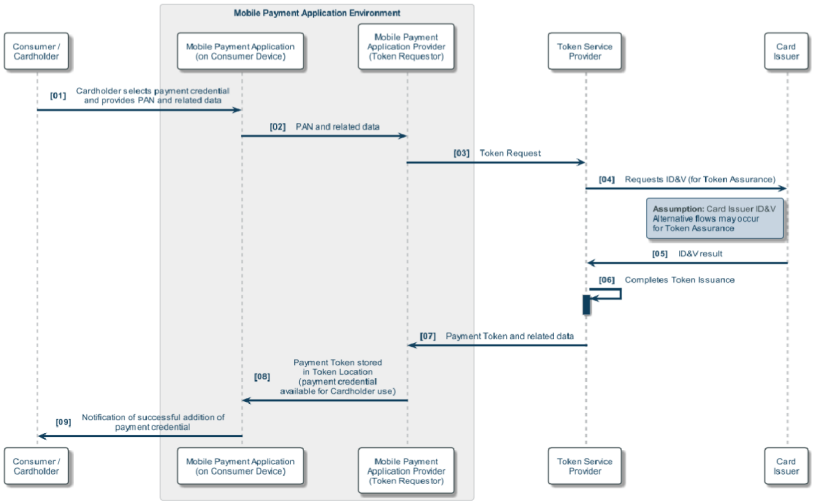


Figure 2.13: Proximity at Point of Sale – Example Issuance Flow, *EMV,Payment Tokenisation – A Guide to Use Cases*

Transaction Flow

Once the cardholder has obtained the token, it can use it to start the transaction phase.
As in the previous phase, only the main step is listed, to see them in detail refer to [13].

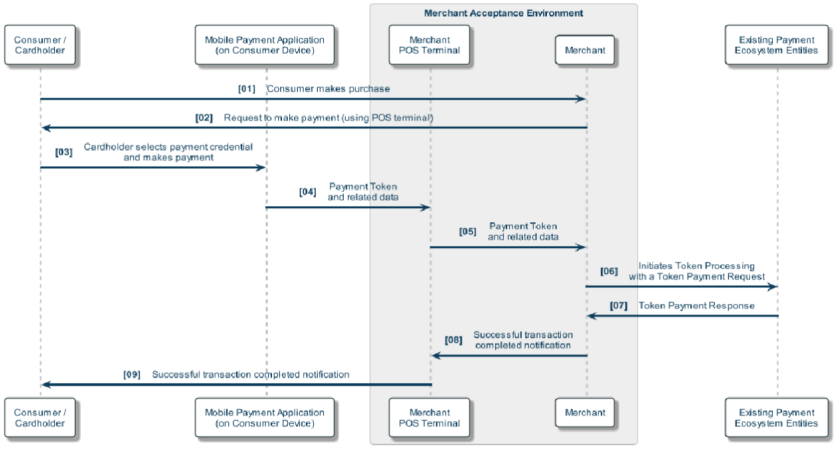


Figure 2.14: Proximity at Point of Sale – Example Transaction Flow, *EMV,Payment Tokenisation – A Guide to Use Cases*

4. The reader of the POS reads the token sent by the cardholder.
5. The POS terminal provides the Payment Token and related data to the Merchant
6. The token is sent over the networks of payment
7. The merchant receive the payment result
8. Te result is sent to the POS
9. Finally the cardholder is notified.

Chapter 3

Design

Following the comprehensive analysis presented in the preceding chapter, it is now feasible to delineate the structure of our implementation with greater precision.

3.1 Overview

Figure 3.1 furnishes an implementation overview.

There are two primary phases: the issuance phase, during which the Host Card Emulation (HCE) acquires the token, and the transaction flow phase, wherein the HCE utilizes the token to execute a transaction. These phases occur at distinct points in time.

For the sake of clarity, each link between different components is identified with a number, whose meaning is explained below:

1. When the user adds a new credit card, its data are sent through the network to the Application Server, i.e., the server of our wallet.
2. The Application Server forwards the credit card data to the Token Service Provider, which performs the tokenization. As previously described, the TSP is the only entity able to link the tokenized data to its real equivalent.
3. After the tokenization is performed, the TSP returns the token to the Application Server.

4. For the sake of efficiency, the Application Server stores the token, linked with a unique ID to the user. Then, the token is forwarded to the user's device.

5. Now, the Client wants to perform a payment. In this case, the POS of the Merchant will request the data to the device.

6. The user's device reply sending the token it received when the card was added.

7. The POS forwards the token, together with information about the payment (e.g. the amount to pay) to the Merchant's bank.

8. The bank's server forwards the user's token to the TSP.

9. The TSP replies with the de-tokenized data.

10. The Merchant's bank can now ask, through its Payments Network, to operate: at this moment, the users are charged the amount to pay and the funds are transferred.

11. The Merchant's bank receives the result of the transaction and forwards it to the Merchant.

12. The POS receives the result of the operation and can print the receipt.

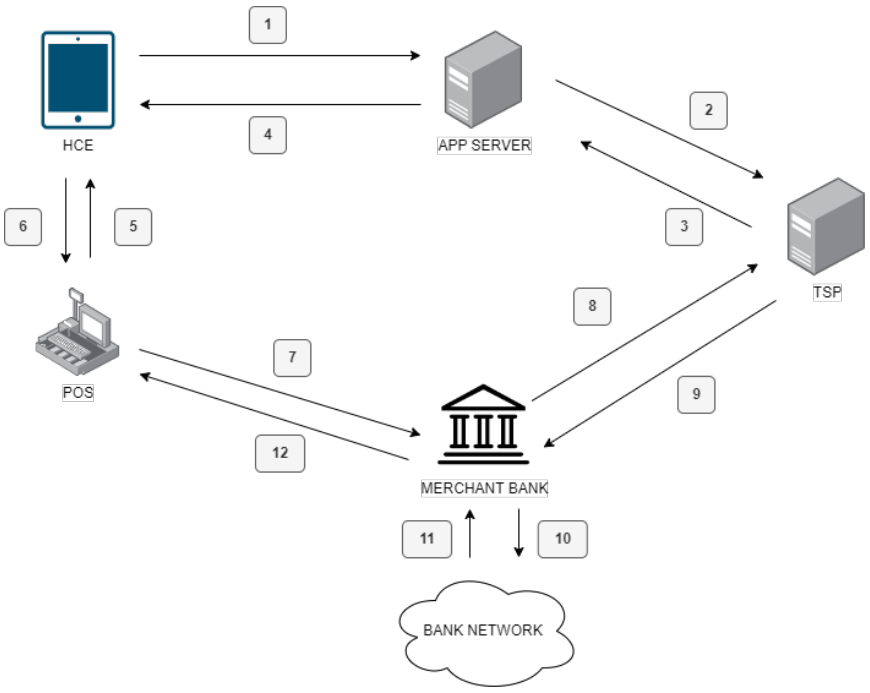


Figure 3.1: Overview of the implementation

Given this step-by-step description, it is now mandatory to focus on the single components.

The forthcoming contents will be organized into two principal sections: the implementation of the NFC terminals (comprising the user's wallet and the merchant's Point of Sale) and the implementation of the "backend" payment network.

3.2 NFC terminals

To simulate a payment environment, two terminals will be developed: the first terminal will implement the Host Card Emulation (HCE), while the second terminal will emulate the Point Of Service (POS).

HCE

The Host Card Emulation (HCE) will be implemented utilizing an Android device.
In the issuance phase, the application will interact with the application server to acquire the token via the HTTP protocol.
Conversely, during the transaction phase, the application will communicate with the Point of Sale (POS) using a customized version of the ISO/IEC 7816-4 protocol [1], featuring a bespoke Application Identifier (AID).

POS

The Point of Sale (POS) will be emulated utilizing a second Android device.
The reader functionalities will be implemented using the ISO-DEP standard [2]. The POS will utilize the HTTP protocol to communicate with the backend server.

Protocol

The main goal of the project is to implement tokenization, so for simplicity, it will be implemented a custom and a simplify version of the ISO/IEC 7816-4[1]. The *select PPSE* command will not be supported and the POS will start reading by sending the *select AID* command directly. After the HCE will ack the response,the POS will read the actual token, by sending a *READ RECORD* command. See the Figure-3.2

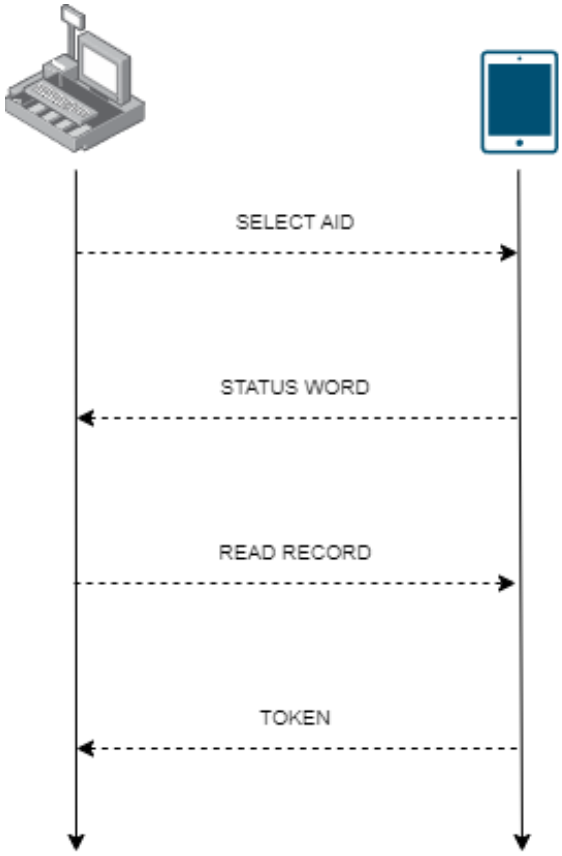


Figure 3.2: Custom POS-HCE protocol

3.3 Payment network

As previously mentioned, this project focuses on the implementation of the HCE payments using tokenization. However, to properly emulate a payment scenario, it is necessary to introduce two additional entities, as described in Figure 3.1:

1. A server able to interface the NFC terminals with the TSP.

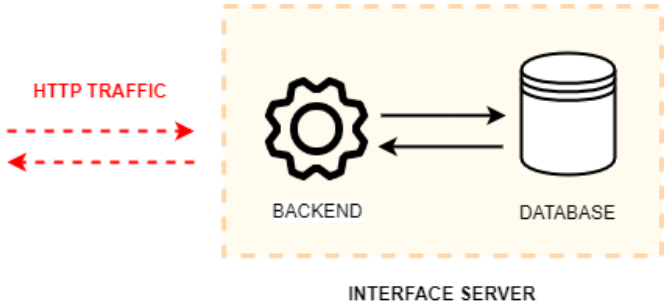


Figure 3.3: Schema of the Interface servers

2. The TSP server.

These entities expose a set of REST APIs to allow terminals and TSP interface. In the following sections, these APIs will be described, as the structures of the requests and the handshakes.

3.3.1 Interface server

The Interface Server is the relay between the NFC terminals and the TSP: it receives data related to a credit card, either its sensitive data or its token, and queries the TSP for its counterpart.
For the sake of clarity, it is necessary to differentiate the Interface server in two cases:

1. The application server, that interfaces the Client's device with the TSP server, during the issuance phase

2. The Banks server, to interface the Merchant's device with the TSP, during the transaction phase.

Besides the different logical nature of these servers, their interfaces share the same endpoints. Then, the inner implementation of both these servers is schematized in Figure 3.3.

Application/Bank Server

Table 3.1 summarizes the methods the Application/Bank Server exposes to add a new credit card to its database or to retrieve the credit card's data from a given token.

Path	Method	Description
/addcard	POST	Adds a card to the server's database and returns the token to perform the payments
/getcard	POST	Returns the data about the card the sent token refers to

Table 3.1: Application Server methods

Field	Description
pan	Primary Account Number - the credit card number
expire	The expiration date
cvv	The secret, 3 digits, number

Table 3.2: Data required to add a new credit card

When the Application server receives the data of a new credit card, it forwards them to the TSP server for the tokenization. The token is returned to the Client's device when the operation is completed.
The process is described in Figure 3.5, while Table 3.2 summarizes the data the Application Server expects to add a new credit card.

Then, when the Merchant wants to perform a transaction, its POS send the received token to the Bank Server, that forward it to the TSP. The Token Service Provider's server retrieves the credit card's data from the given token and returns them to the Merchant's device. This flow is described in Figure 3.4. In a real scenario, the POS will not receive all the card data from its bank. In this case, it has been done only for practical reasons, to show the de-tokenization process.

3.3.2 Token Service Provider server

Table 3.3 summarizes the methods exposed by the TSP server.

When the Application Server want to get the token of a newly added credit card, it forwards the data to the Token Service Provider's server. These data are the same as described in Table 3.2.
Then, the TSP server performs the tokenization, stores the token and the credit card data in its database and retrieves the token to the Application Server.

Otherwise, when the Application Server wants to perform a transaction with the Bank Network, it uses the path `/get` to retrieve the data corresponding to the provided token.

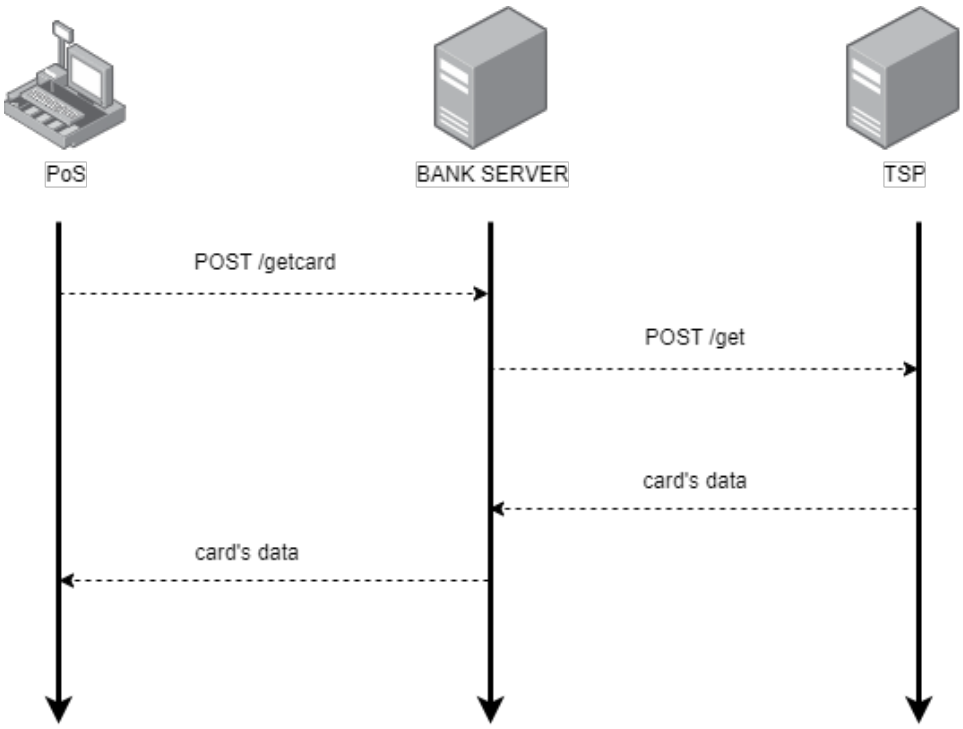


Figure 3.4: Process to retrieve the credit card's data from the token

Path	Method	Description
/tokenize	POST	Receives the card's data, performs the tokenization and returns the token.
/get	GET	Returns the credit card's data referred to the received token

Table 3.3: TSP Server methods

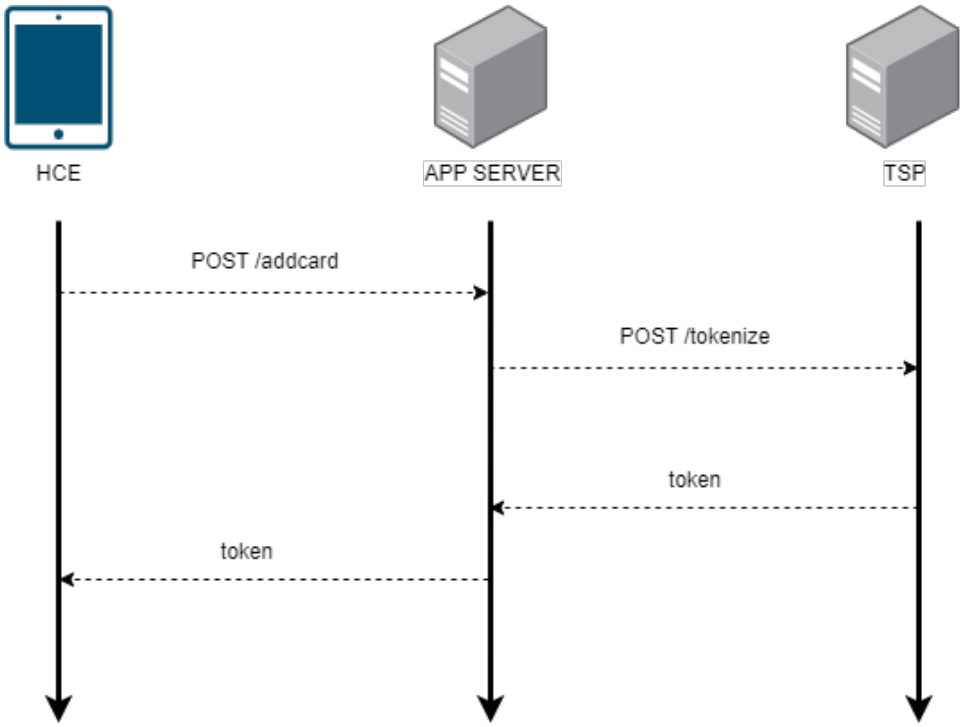


Figure 3.5: Request and response flow between HCE, App Server and TSP

Chapter 4

Proof of Concept

This section will describe how to deploy and test the developed proof of concept.

4.1 Software and Hardware setup

The subsequent tests are performed using:

1. Two Android devices with NFC capability
2. Android Studio
3. A laptop with a Python3 environment
4. Wi-Fi connectivity

4.2 Services deployment

The services required are the TSP server and an Application Server, which in this case can act both as a Bank server and Wallet server. This has been done to simplify the testing environment.

To deploy the TSP server, it is necessary to move inside the TSP folder, then run:
`python3 main.py`
The server will start running on port 5000 on the local host. Figure 4.1.B shows the output of the console.

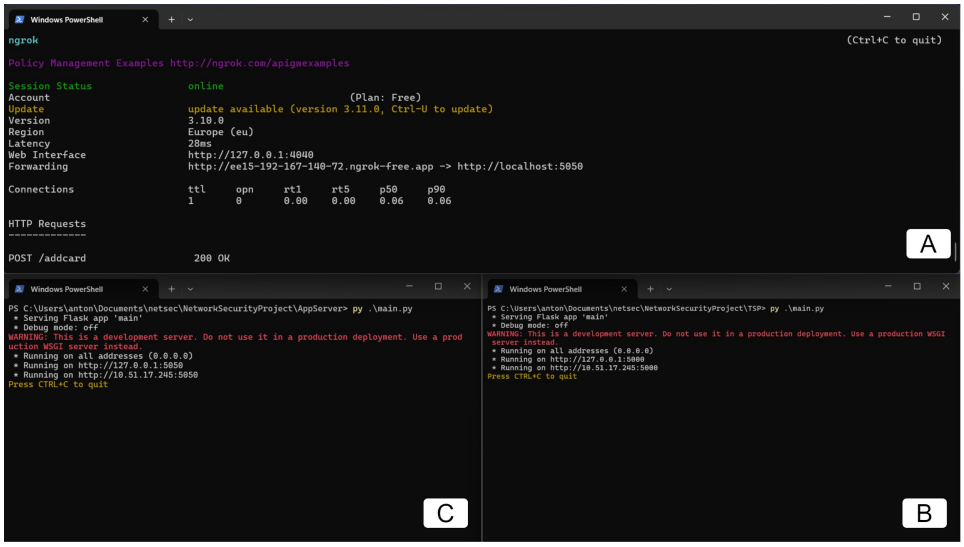


Figure 4.1: Testing environment

Then, the Application Server can be deployed. After entering the AppServer folder, run the command:
`python3 main.py`
The server will start running on port 5050 on the local host. Figure 4.1.C shows the output of the console.

Both services are based on the Flask framework and use a sqlite3 database, meaning that the following Python modules must be installed in the host:

- Flask
- sqlite3
- requests

For the sake of simplicity, the project includes a text file `requirements.txt`. Running the command:
`python3 -r requirements.txt`
all the dependencies will be installed on the host machine. A virtual environment is recommended but not mandatory.

4.3 Making the Application Server's port reachable

To allow the Android devices to connect to the server's port, it is necessary either to stay on the same LAN or to expose the port.
Aiming to avoid connectivity issues, in this deployment the second solution has been chosen.

To expose the Application Server's port, the service chosen is `ngrok`.
The server is now reachable from devices in other networks. Figure 4.1.A shows the console's output.

4.4 Applications deployment

In order to deploy the application, the utilization of Android Studio is necessary. The minimum required SDK version is 23, while the target version is 34, as depicted in Figure 4.2.

Two devices are requisite for this purpose: the initial device will function as the host card emulation (HCE) application, whereas the second device will operate the point of sale (POS) application.

Figure 4.2: SDK versions

4.5 Adding a new credit card

With all the services and applications deployed, it is now feasible to execute the HCE application and add a new credit card.

Upon initial launch, the application is preloaded with a placeholder credit card: it is possible to add a new card by selecting the designated button, located in the bottom right corner.
This action will prompt a form to appear, allowing the input of the relevant credit card details.
Submitting this information, the credit card will be successfully added to

the wallet.
All the procedure is described in Figure 4.3.

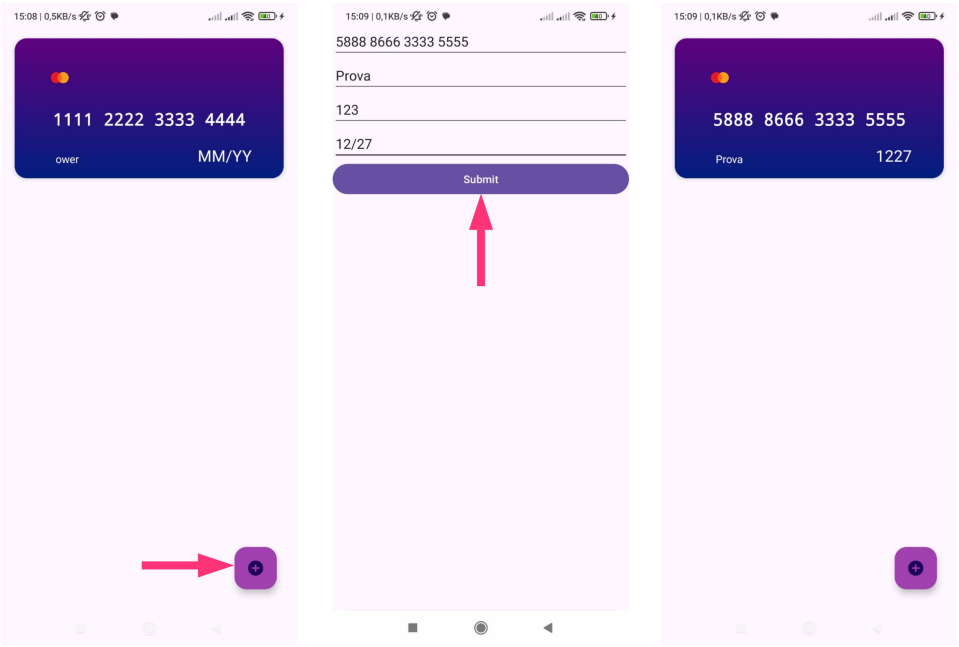


Figure 4.3: Procedure to add a new credit card

4.6 Testing the transactions

When a new credit card is configured, it is possible to emulate a transaction between the HCE application and the POS.

As it is possible to see in Figure 4.4, the HCE device initializes a transaction with the POS. It is possible to notice that the transaction has been performed correctly, due to the POS showing the received data. This procedure emulates a payment.
It is worth underlining that in a real-world scenario, the POS will not receive the sensitive data. These data have been shown only for academical purposes.

Finally, Figure 4.5 shows the consoles of the services, with the logs of the

transactions performed.

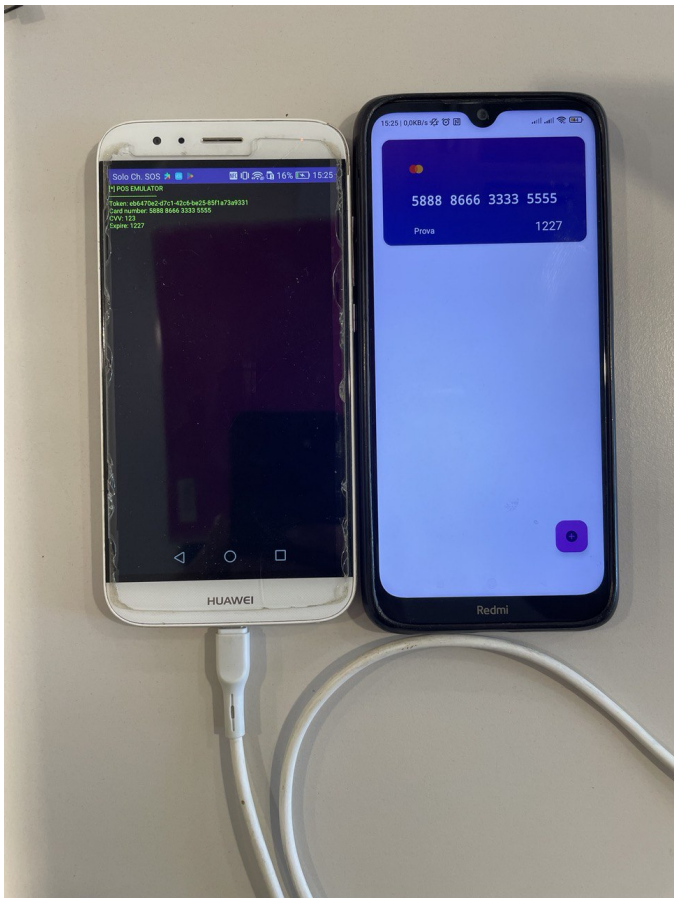


Figure 4.4: Result of the transaction between the POS and the HCE application

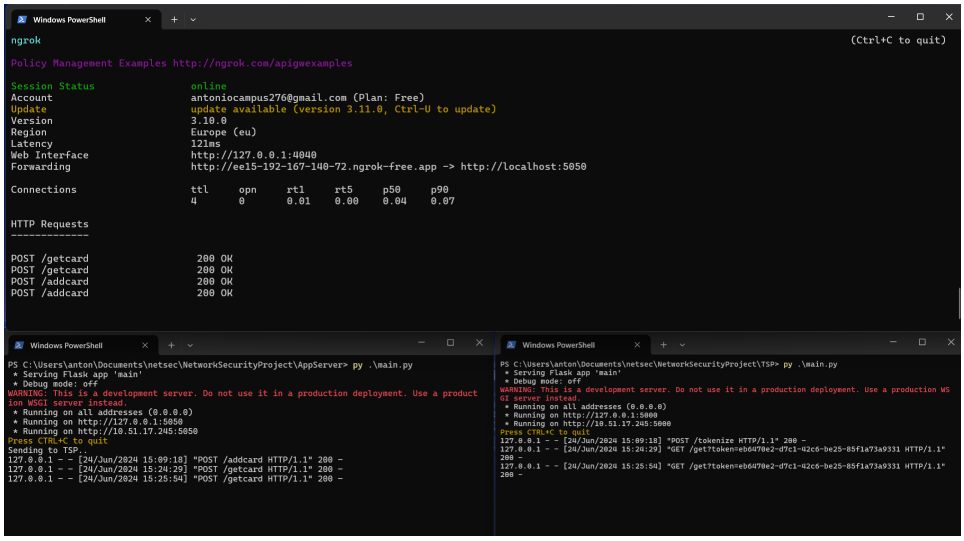


Figure 4.5: Dashboard with the logs of the transactions

Chapter 5

Conclusion and Open Issues

This project presented a proof-of-concept about a NFC-based payment infrastructure.
Despite the desire to stick as closely as possible to the conditions of a real scenario, as previously mentioned, some simplifications have been considered mandatory.

For the sake of completeness, these simplifications will be discussed, together with security issues recognized in our design.
First thing first, it must be considered the lack of knowledge about proprietary protocols of Credit Card Networks. Those protocols, when available, are subject to fees, making their implementation unfeasible in our proof-of-concept.
Then, the real implementation of a tokenization infrastructure is known from a high-level point of view: the details are not usually provided and change between different providers.
Last, but not least, the final part of the transaction has been simplified: our solution does not involve any, emulated, money transfer, it is limited to forwarding of the de-tokenized data to the Merchant's POS. This has been done due to the lack of knowledge and excessive specificity of each real Bank's service implementation.

In conclusion, it is worth highlighting some security concerns that have been noticed when implementing this project.
The lack of encryption in the token transmission to the user's device could lead to a Man In The Middle, where a malicious actor could be able to steal the token and use it for its purposes. Instead, the predictability of the token

is not a matter of concern, being it generated randomly. A future improvement could be, sure, to implement encrypted data transmission.

Acronyms

AID Application Application Identifier. 14

APDU Application Protocol Data Unit. 6, 11, 12

CE Card Emulation. 7

EMV Europay, Mastercard and Visa. 10

HCE Host Card Emulation. 1, 5–7, 10, 11, 21, 23

NFC Near Field Communication. 2, 3, 5, 11, 23, 24, 34

PAN Primary Account Number. 8

POS Point Of Service. 4, 7, 10, 11, 15, 20, 21, 25, 31, 34

SE Secure Element. 5–7

TEE Trusted Execution Environment. 5, 6

TSP Token Service Provider. 1, 16, 23–25, 28

List of Figures

List of Tables

Bibliography

[1] "iso-7816." https://www.iso.org/obp/ui/#iso:std:iso-iec:7816:-8:ed-5:v1:en. Visited in 21/05/2024.

[2] "iso-14443." https://www.iso.org/standard/73599.html. Visited in 21/05/2024.

[3] "Android KitKat." https://www.android.com/intl/en_en/versions/kit-kat-4-4/. Visited in 06/06/2024.

[4] "Host card emulator." https://developer.android.com/develop/connectivity/nfc/hce. Visited in 21/05/2024.

[5] "EMV." https://www.emvco.com/. Visited in 21/05/2024.

[6] "EMV." https://www.emvco.com/emv-technologies/payment-tokenisation/. Visited in 21/05/2024.

[7] "Book A: Architecture and General Requirements." https://www.emvco.com/specifications/book-a-architecture-and-general-requirements/. Visited in 21/05/2024.

[8] "Book A: Entry Point." https://www.emvco.com/specifications/book-b-entry-point/. Visited in 21/05/2024.

[9] "Contactless Communication Protocol." https://www.emvco.com/specifications/book-d-contactless-communication-protocol/. Visited in 21/05/2024.

[10] "ADPUs." https://www.openscdp.org/scripts/tutorial/emv/rademv.html. Visited in 21/05/2024.

[11] "aid-list." https://www.eftlab.com/knowledge-base/complete-list-of-application-identifiers-aid. Visited in 21/05/2024.

[12] "EMV Ddecoder." https://emvlab.org/tlvutils/. Visited in 21/05/2024.

[13] "Payment Tokenisation – A Guide to Use Cases." https://www.emvco.com/specifications/emv-payment-tokenisation-a-guide-to-use-cases-2/. Visited in 21/05/2024.