University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Acceptance of Payment Cards on Android OS Devices

Plzeň 2022 Bc. Stanislav Král

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Stanislav KRÁL**
Osobní číslo:	**A20N0091P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Akceptace platebních karet na zařízeních s OS Android**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte informace o používaných čipech na platebních kartách a jejich zabezpečení s ohledem na bezkontaktní transakce. Prostudujte standardy PCI CPOC pro akceptaci karetních transakcí a PCI SPOC pro možnosti zadání PINů na běžných chytrých zařízeních obchodníků.
2. Proveďte analýzu možností akceptace karetních transakcí na vybraných zařízeních na platformě Android. Uveďte rozdíly pro transakce prováděné fyzickými platebními kartami a virtuální kartami včetně Google Pay a Apple Pay.
3. Na základě předchozích bodů implementujte pro platformu Android prototyp aplikace pro čtení údajů z platební karty s využitím dostupných SDK asociace VISA.
4. Ověřte funkcionalitu aplikace na simulátoru karetních transakcí, výsledky vyhodnoťte a navrhněte vhodné budoucí úpravy či rozšíření.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Martin Míček**
Smart software s.r.o.
Konzultant diplomové práce: **Ing. Ladislav Pešička**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **10. září 2021**
Termín odevzdání diplomové práce: **19. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2021

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Plzeň, 19th May 2022

Bc. Stanislav Král

Acknowledgement

I would like to thank Ing. Martin Míček and Ing. Ladislav Pešička for their valuable advice and guidance throughout this project. Additionally, my thanks also go to my family and friends who supported me during my studies.

Abstract

The main goal of this master's thesis is to find out whether non-specialized devices, such as mobile phones, could be used to accept payment cards. Therefore, it covers the SPoC and CPoC standards and the way payment transactions utilizing the EMV technology are processed. Additionally, the differences between the Google Pay and Apple Pay mobile payment applications are described there. The commercial application Dotypay is extended in such a way that it can be used on mobile phones and Nexgo devices running the Android OS to accept cards. Finally, using the industry-standard UL Brand Test Tool product, it is verified that the extended application is able to correctly process payment transactions.

Abstrakt

Hlavním cílem této diplomové práce je ověřit, zdali mohou být k akceptaci platebních karet používána běžně dostupná zařízení, jako např. mobilní telefony. Z tohoto důvodu se práce nejdříve zaměřuje na to, jakým způsobem funguje placení kartami, které využívají technologii EMV, a poté popisuje standardy SPoC a CPoC. Dále jsou zde také popsány rozdíly mezi mobilními aplikacemi Google Pay a Apple Pay umožňující použití mobilních telefonů namísto karet. V neposlední řadě je rozšířena komerční aplikace Dotypay tak, že může být provozována k akceptaci karet na mobilních telefonech a zařízeních Nexgo, které používají platformu Android. Pomocí specializovaného testovacího nástroje UL Brand Test Tool, který se v odvětví platebních karet běžně používá, je pak ověřeno, že aplikace správně zpracovává platební transakce.

Contents

1 Introduction

Today, the option to pay for goods or services using payment cards is almost taken for granted by customers. Payment cards make the checkout process more convenient and quicker for both the customer and the merchant. In some situations, the inability to pay a merchant with a card may eventually discourage customers from visiting such a business, making them look for other options in the area. To summarize this, payment cards play an important role in today's economic system and society, and merchants that do not allow their customers to pay with cards are at a disadvantage over those who accept them.

However, some merchants may consider the initial cost of getting a card terminal rather high and since for every transaction performed the merchant has to pay a small fee, they eventually keep on accepting cash only.

Nowadays, a lot of people already own a mobile device. Enabling it to be used instead of card terminals to accept payment cards could be a solution to the problem as this would completely remove the initial cost related to buying a card terminal and may even lead to smaller fees for transaction processing. This could result in the increase of the number of merchants that allow payment cards to be used in their stores. Additionally, a mobile device could be also used for other purposes required to run a store, e.g., to operate a point of sale cash register application.

This work aims to introduce the reader to the complex system that stands behind the world of cashless card payments we use every day by covering the most important standards and principles of this industry. From the description of a payment card to the explanation of card transaction processing, the fundamentals of this topic are covered in the second chapter, while the most widely used standard to facilitate payments is introduced in the third chapter. In the fourth chapter, the reader is familiarized with technologies used to allow Android and iOS devices to be tapped onto card readers to authorize card transactions. Chapters 5 and 6 summarize two standards that can make acceptance of cards available to more merchants by allowing commercial mobile devices to be used as card terminals. Finally, the seventh chapter describes a designed architecture of a card terminal application and its implementation in a commercial application – Dotypay, that is also described there. Last but not least, the results of testing the updated commercial application on selected devices using a specialized tool for testing card terminals are presented in the final chapter.

2 Payment Cards and Their Protocols

This chapter covers the fundamentals of the card industry and is mostly based on the book Acquiring Card Payments by Ilya Dubinsky [1].

2.1 Payment Card Description

The most important standards related to the description of payment cards include standards ISO/IEC 7813, 7816 and 14443.

Dimensions, shape and design of cards are defined in the ISO 7813 standard, which also covers the location of magnetic stripes on the card as well as the structure of the data stored on it. The ISO 7816 standard covers the definition of a smart card (card with an integrated circuit chip embedded into it) by specifying its physical characteristics such as dimensions of contacts and their placement, data transfer protocols, or definition of commands used to manage applications in the multi-application environment.

The definition of properties of contactless cards and their communication protocols is present in the ISO 14443 standard. The most widely used implementations of this standard include technologies such as PayPass (now Mastercard Contactless), Visa payWave (now Visa Contactless) or ExpressPay.

Figures 2.1 and 2.2 contain illustrations of a payment card's front and back sides. The following elements are annotated in these figures:

1. **PAN** (Primary Account Number) – further described in 2.1.1.
2. Card's date of expiration – the last day the card can be used.
3. Cardholder's name – the name of the person to whom the card belongs.
4. **IC** (Integrated Circuit) – integrated circuit used in contact and contactless transactions to exchange card data with a card terminal.
5. Card issuer logo/branding.
6. Card scheme logo/branding (card scheme entity further described in 2.5).

7. Visual indicator that the card supports contactless transactions.
8. Cardholder's signature.
9. Last four digits of card's PAN identifier.
10. **CVV** (Card Verification Value) code – further described in 2.2.
11. Magnetic stripe – contains card data, further described in 2.1.3.

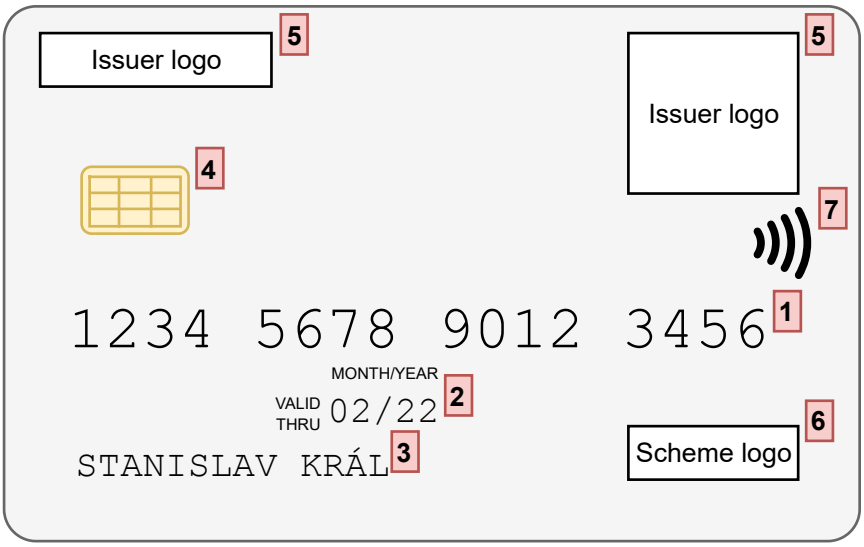


Figure 2.1: Illustration of the front side of a payment card with annotations.

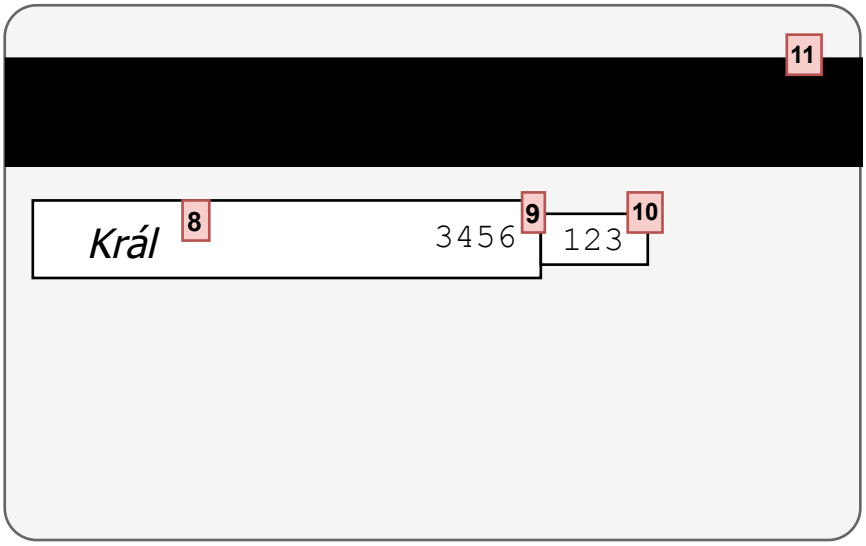


Figure 2.2: Illustration of the back side of a payment card with annotations.

2.1.1 PAN Identifier

The main identifier of payment cards is the numeric PAN identifier, which consists of between 8 to 19 decimal digits. The first 6 to 8 digits represent the **IIN** identifier (Issuer Identification Number), which identifies the card's issuer. The IIN identifier is followed by the identifier of the cardholder which consists of up to 12 digits. The last digit of the PAN identifier is used as a check digit to ensure the integrity and validity of the identifier as a whole.

The IIN identifier, also known as **BIN** (Bank Identification Number), relates to BIN tables that are used to determine additional information about the card, such as its type, issuer or PAN identifier length. The BIN identifier may help merchants to decide whether the brand of the card presented by a customer is supported.

2.1.2 Types of Payment Cards

The most simple way of categorizing payment cards is to divide them into groups based on whether they are issued to an individual or a company, while the former group is more common.

However, more often are cards categorized based on the pattern of reconciliation with the cardholder. Debit cards are such cards, that are tied to a banking account with a limited balance, and each time a successful payment transaction is finished, the transaction amount is subtracted from the account balance. Credit cards work almost identically to debit cards, except the issuer pays for purchases made with credit cards and their cardholders become indebted to him. The third most common group of payment cards are prepaid cards that are not necessarily tied to a bank account but rather to a prepaid balance managed by the issuer.

In the early times of payment cards, all cards were essentially charge cards and worked similarly to credit cards. The issuer paid for all purchases made by the cardholder, who had to settle the outstanding debt in full by the end of the month, but as opposed to credit cards, the loan of money was interest-free in this case.

2.1.3 The Magnetic Stripe

The magnetic stripe located on the backside of payment cards is used as storage for data containing information about the card and its owner. The stripe usually contains 3 tracks, but only the first 2 tracks, available only for reading, are commonly used today.

Track 1

The content of the first track can contain up to 79 alphanumeric characters and holds the PAN identifier, cardholder's name, card expiration date, service code and discretionary data.

Track 2

As opposed to Track 1 the content of Track 2 may consist of numeric characters only and is limited to 40 characters. It contains the PAN identifier, card expiration date, service code and discretionary data.

Track 3

The third track was designed in a way that its content may be dynamically changed to enable additional features and functions, such as setting a daily limit to the spending of available funds.

However, Track 3 is not widely used today as integrated circuit chips offer better support for the implementation of dynamic features.

Service Code

All supported card operations are specified by the service code that resides on the magnetic stripe. It consists of three decimal numbers that have the following meaning:

- **the first number** – restricts the card's usage abroad,
- **the second number** – defines whether online authorizations are required,
- **the third number** – defines conditions that require authorization using a **PIN** (Personal Identification Number) and whether the card usage is limited only to cash withdrawal at an ATM.

For example, the service code 226 indicates that the card can be used abroad, but its IC must be used during transaction processing. Additionally, it specifies that transactions performed with this card must be authorized online and preferably requiring a PIN from the cardholder.

2.2 CVV Code

To reduce the ability of attackers to create and use counterfeit cards, today's payment cards have **CVV** (Card Verification Values) codes assigned to them.

These codes are also known as **CSC** (Card Security Code), **CID** (Card Identification Number), **CVC** (Card Verification Code) or **CVN** (Card Verification Number), but this text will refer to them as CVV codes.

2.2.1 Computation Algorithm

The computation of a CVV code includes assembling an input string containing the PAN identifier of the card, card's date of expiration, and card's service code. This string is then encrypted multiple times using the 3-DES encryption algorithm with the **CVK**[1] (Card Verification Key) used as the encryption key.

Multiple variants of this algorithm are used to secure payment card transactions but often differ only in the way that the input string is created.

2.2.2 CVV1

The CVV1 code is a part of the discretionary data stored on Track 1 and Track 2 and its main purpose is to prevent the creation of counterfeit cards when an attacker gets hold of the card's PAN identifier and its expiry date. The attacker would have to have the CVK in possession in order to recreate the CVV1. This code also provides protection against manipulation of the card's service code, which could allow the attacker to bypass the limitations of the card's usage. The protection is guaranteed by the fact that the service code is present in the input string that is eventually transformed into a CVV1 code.

2.2.3 CVV2

In card-not-present scenarios[2] the CVV2 code is used to verify that the originator of the transaction is the owner of the card. The CVV2 is usually located on the backside of the card and consists of 3 to 4 decimal digits.

The algorithm used to compute the CVV2 code is identical to the algorithm used to compute the CVV1 code. However, a security code con-

[1] A key kept secret by the issuer of the card.

[2] Such scenarios, where the card is not read by the terminal, but its PAN and date of expiration are used to perform payment transactions in an online environment.

sisting of three zeroes is used in the computation instead of the card's true security code.

2.2.4 iCVV

Due to backward compatibility with payment networks that use legacy protocols utilizing data only from the magnetic stripe, cards nowadays have the iCVV code stored in the integrated circuit chip to be able to distinguish transactions made using the magnetic stripe from transactions made using the integrated circuit chip. Without the iCVV code, the attacker may be able to use magnetic stripe data skimmed[3] from a card to create a magnetic stripe-only card and use it in networks with legacy protocols.

The iCVV code computation follows the same algorithm as the previously mentioned codes but uses the number 999 as the security code.

2.2.5 dCVV

Similar to the previously mentioned CVV code computation algorithms, the dCVV computation algorithm also uses the card's PAN identifier, its expiry date and its service code to create the input string that is later encrypted using the 3-DES cipher. However, in the case of dCVV code, the input string also contains a randomly generated number and a transaction counter. This ensures that for each transaction a new CVV code is generated, reducing the risk of fraud.

The dCVV code is usually used in contactless transactions.

2.3 Card Presentation Methods

During the processing of a payment transaction, the card might be presented using several methods.

2.3.1 Use of Magnetic Stripe

One of the oldest card presentation methods is definitely the one using a card's magnetic stripe when the card is swiped through the magnetic stripe card reader, during which one of its tracks is read to gather cardholder data. Because it contains mostly static data, this method is vulnerable to various attacks, such as skimming the card to perform fraudulent transactions or

[3]To skim a card means capturing the data stored on its magnetic stripe.

changing the card's service code [2]. For security reasons, the use of this card presentation method is now being abandoned worldwide.

2.3.2 Chip Transaction

Card presentation methods which include the insertion of a payment card into a card terminal are considered secure by today's standards since they use the integrated circuit chip present on the card and utilize the EMV technology[4] for establishing the exchange of data between the card and the terminal.

2.3.3 Contactless Transactions

Similarly, as with chip transactions, where the card is inserted into the card reader, the contactless card presentation method utilizes the integrated circuit chip and the EMV technology. The main difference from chip transactions is that the card is not inserted into the card reader, but rather attached to the contactless card reader of the terminal. During the attachment to the reader, the integrated circuit chip is powered up using the electromagnetic field of the reader and establishes an EMV dialog with the terminal.

One of the things the method must account for is the fact that the card is attached to the reader only for a limited period of time, and before the terminal receives a message from the back-end processing system indicating whether the transaction has been successfully authorized, the card will probably not be attached to the reader anymore.

2.3.4 Card Tokenization

A special token that can be used instead of the payment card is presented to the card terminal using a mobile device such as a mobile phone or a smartwatch and the payment network translates it to the card's original identifier before the transaction is authorized.

Card tokenization on mobile phones introduces new forms of cardholder verification methods that can utilize their authentication capabilities such as facial recognition or authentication using fingerprints.

[4] A technology following a set of standards used to process card transactions, described in detail in the chapter 3.

2.4 Cardholder Verification Methods

When a payment card is used to perform a payment transaction, a verification might be required to ensure whether the card has not been stolen and the person who presented it is indeed the legitimate cardholder. In other words, **CVM** (Cardholder Verification Method) is a procedure that allows for verification that the person at the point of sale is the person to whom the card was issued.

CVM methods differ based on whether the transaction is made in a card-present or card-not-present environment. The former type covers all scenarios in which a card has been presented by its cardholder and read using a payment terminal, as opposed to the latter which covers scenarios in which the card's PAN identifier, expiry date and other required attributes have been manually (e.g., using an online form) supplied to the merchant without presenting the physical card.

Card-present CVM methods include Offline PIN, Online PIN, Signature, Consumer Device Cardholder Verification Method, Failed CVM and No CVM.

Card-not-present CVM methods include Static password, One-time password, Address Verification Service and Mobile authentication.

The CVM limit is the transaction amount above which all card transactions must include some sort of CVM and it is often specific to the country the transaction is being performed in. During contactless transactions this most often includes the Online PIN CVM (described in 2.4.2). Table 2.1 shows CVM limits in selected countries.

Country	**CVM Limit**
Australia	A$200,00
Croatia	350,00 HRK
Czech Republic	500,00 CZK
Germany	€ 50,00
Poland	100,00 PLN
United Kingdom	£100
USA	$100

Table 2.1: CVM limits in selected countries as of 2021 [3].

During the COVID-19 pandemic, some countries have decided to increase the CVM limit in order to promote contactless transactions and reduce the amount of cardholder's physical interaction with the card terminal [4].

2.4.1 Offline PIN

In order to verify the identity of the cardholder, a secret PIN, known only to the cardholder and the card's issuer, has to be entered using a pin-pad of a terminal. By communicating with the card, the terminal verifies whether the entered PIN is valid.

As the name of the method implies, the entered PIN is verified without the presence of the card's issuer and does not require a connection to the internet. Local communication between the card's IC and the terminal is established during the Offline PIN CVM.

After a PIN is entered using the terminal's PIN pad, it is transferred to the card for verification either encrypted or in plain text. Before verifying the PIN, the card's IC also checks whether the limit of failed PIN attempts hasn't been reached. To prevent the submission of an incorrect PIN (e.g., when a cardholder has been prompted for it but realized he does not remember it) a cardholder can abort the verification method and the counter of invalid PIN entry attempts on the card is not incremented.

2.4.2 Online PIN

Similar to the Offline PIN method, the Online PIN method depends on a secret PIN to verify the identity of the cardholder. However, instead of relying on the card's IC, this method requires that the PIN, alongside additional transaction information, is securely sent to the card's issuer for verification.

The entered PIN must be packaged into an encrypted PIN block using the 3-DES cipher before it is sent to the issuer to ensure it is not eavesdropped on by an attacker.

Since this method does not rely on EMV technology it can be used with magnetic stripe cards too.

2.4.3 Signature

It is a verification method that is based on a comparison of the cardholder's signature on the card with a signature obtained from the customer at the point of sale.

The merchant asks the customer to sign for the payment (typically using a touchscreen device or a paper receipt) and then compares it with the signature on the card itself. Additionally, the merchant is responsible for keeping a physical or digital copy of the signature together with the receipt to be able to use it in case of a dispute.

This method is not considered secure, since the signature is already present on the back of the card and is very easy to forge. Because of that, it is most often used in combination with the Offline PIN CVM method.

2.4.4 Failed CVM

Used in exceptional scenarios, this CVM method indicates that the card forces a CVM failure in the terminal, which may result in the abortion of the transaction or an online connection to the card's issuer for further details.

2.4.5 No CVM

This method immediately accepts the transaction and is used in places where PIN or other CVM methods may lead to unreasonable congestion or traffic build-up and where the amounts of individual transactions are relatively small. At the cost of lower security, this method offers faster customer checkouts and is usually used in unattended terminals (e.g., in public transit or vending machines).

2.4.6 Consumer Device Cardholder Verification Method

With the growing popularity of mobile phones being used instead of cards to pay for goods in stores, it was required to introduce a new CVM method convenient for that presentation method. **CDCVM** (Consumer Device Cardholder Verification Method) allows customers using their own devices, such as mobile phones, to authenticate themselves via biometrics or passwords for secure cardholder verification. This means that customers can authenticate themselves on their mobile devices instead of card terminals, effectively reducing the possible attack surface that might be used, for example, to capture the cardholder's PIN.

2.4.7 Static Password

When finalizing an online payment the customer has to enter a password that he has previously defined on the issuer's bank website to prove his identity.

2.4.8 One-time Password

Similar to the Static password CVM, the One-time password CVM requires the customer to enter a password when finalizing an online payment. However, a new password is generated by the issuer for each transaction and is delivered to the customer via an independently verified channel, such as via an SMS sent to the cardholder's mobile phone.

This method offers better security than the Static password CVM method, as passwords are generated for each individual transaction and a potential attacker is unable to perform multiple fraudulent transactions when the cardholder's password has been leaked.

2.4.9 Mobile Authentication

To provide greater security than other card-not-present CVM methods, some issuers allow for a CVM using a custom mobile application developed by the issuer for cardholder verification. Instead of relying on the protocol behind SMS messages, this method may deliver a one-time password to the mobile application on the cardholder's mobile phone using the internet network.

However, more often than delivering one-time passwords, the issuer bank delivers a payment request to the application installed on the cardholder's mobile phone. For the CVM to be successfully completed the payment request has to be manually confirmed by the cardholder in the application. The request can only be confirmed after a successful cardholder authentication using a previously defined PIN, password, or biometrics such as fingerprint or facial recognition.

2.5 Entities Participating in Transaction Processing

Before describing details of transaction processing, it is required to introduce its main actors.

A card scheme, also known as a card network, stands between issuers and acquirers and transfers card transaction information based on a set of security procedures and rules. Today, Visa, Mastercard and UnionPay are considered the three of the largest card schemes [5].

An issuer (sometimes referred to as issuer-bank) is responsible for relationships with cardholders, as it issues them cards according to card scheme brand guidelines. It keeps track of the cardholder's account balance, provides credit or prevents card fraud. Additionally, based on requests from card

schemes, it checks whether the cardholder's account has sufficient funds to authorize a card payment transaction.

An acquirer is an entity that processes card transactions originating from merchants, allowing them to accept supported payment cards. To process card transactions, it has to forward the transaction to the correct card scheme. Eventually, after a card transaction has been cleared and settled, the acquirer is responsible for depositing the transaction amount (minus applicable fees) to the merchant's account.

2.6 Transaction Processing

After a card has been successfully presented to the card terminal and the cardholder's identity was verified, the transaction data leaves the terminal and is transferred to an acquirer where it is routed based on the BIN identifier of the card to the correct card scheme [6].

The card scheme then processes the transaction data and requests the card's issuer for authorization of the payment transaction, i.e., to check whether the cardholder's account holds sufficient funds to pay for the transaction. If the authorization was successful, the cardholder can see the payment transaction in their bank account with funds equal to the amount of the transaction unavailable for spending for the next few days or until it is claimed by the merchant. The result of the authorization is then sent back to the acquirer, who notifies the merchant. If the payment has been authorized, then it is usually considered that the cardholder has paid for the goods and may leave the point of sale.

However, up until this point, the funds have not been transferred to the merchant's bank account. The merchant has to assemble a list of authorized transactions (also called a clearing file) and send it to the acquirer, who forwards it to the card scheme for clearing[5]. This usually happens once a day, but card schemes like Mastercard allow for the submission of clearing files up to 5 times a day. The card scheme then communicates with the issuer and a fund transfer to make a settlement between involved parties is initiated. Both issuer and card scheme do perform analysis of all submitted transactions and determine their validity to prevent fraudulent transactions. It may take several business days before the transaction payment is received by the merchant. A sequence diagram of the aforementioned process can be found in Figure 2.3.

[5]The process of settling financial transactions between two banks.

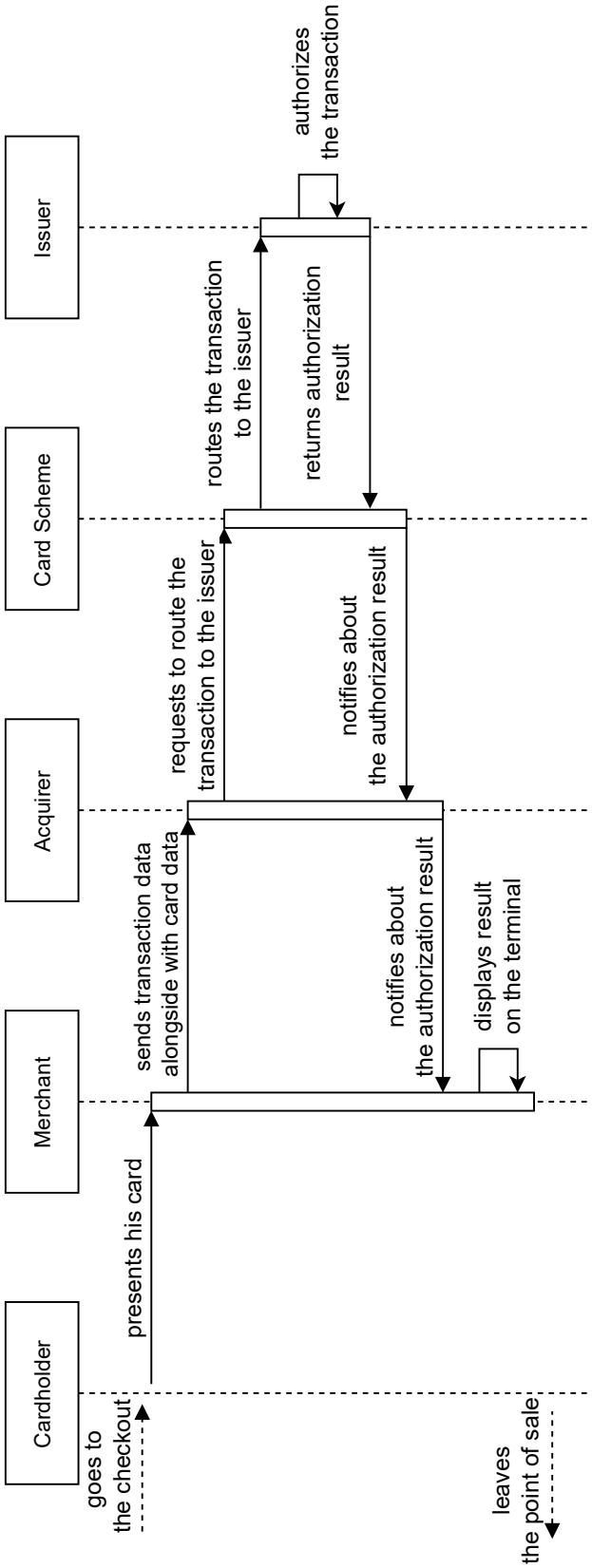


Figure 2.3: A sequence diagram visualizing the authorization of a transaction.

Merchants are often connected to Payment Service Providers (PSPs) instead of acquirers because merchants wish to accept as many card brands as possible and support various payment methods, but due to technical difficulties, it is often unfeasible for merchants to achieve this on their own.

2.7 Transaction Types

2.7.1 Purchase

Purchase is the most frequently used type of card transaction. It is initiated by a merchant and finalized by a customer so that he can pay for goods using his payment card. Once a transaction is authorized, the customer may leave the **POS** (Point-of-Sale) with the goods they have just bought.

2.7.2 Pre-authorization and Completion

In cases where services are delivered at a later date or when the final amount to be authorized is unknown at the time of authorization pre-authorization type transactions are used. It is similar to a classic purchase transaction, but instead of being sent by the merchant to the acquirer for immediate clearing, it is kept until the merchant is able to determine the final amount to be paid by the customer. Pre-authorization is typically used in places where merchants or service providers take an advance, such as in car rental services, gas pumps or hotels.

Completion does not require the card to be present at the terminal and takes place after the final amount to be paid by the customer is determined. The customer may only be charged the amount originally authorized at a maximum during that transaction.

2.7.3 Refund

A customer can have his funds returned to his account after a purchase using a refund transaction. Refunds usually happen when a customer returns bought products, services are canceled or price adjustments are made.

To limit fraud and money-laundering risks, card schemes usually set a limit to the maximum amount of refund transactions, often the full or partial amount of the original transaction.

2.7.4 Reversal

In cases where a transaction was accidentally processed more than once, a merchant requested the wrong amount when submitting a transaction, or a product purchased is out of stock, it can be canceled using a reversal transaction. All purchase, pre-authorization and refund transactions can be canceled within a certain time window (typically within 24 hours of a transaction's authorization [7]) set by card schemes and acquirers, and, when canceled, a transaction does not appear in the cardholder's bank account as opposed to a refund transaction made after a purchase transaction.

2.7.5 Mail Order Telephone Order

MOTO (Mail Order Telephone Order) is a card-not-present environment transaction type, during which card details (PAN identifier, expiry date and CVV code) are given over the phone or through the mail. The cardholder's card data is then used to perform a purchase transaction.

Today, this transaction type is being replaced by online payment gateways that allow filling in the card details into an online form.

2.7.6 Balance Inquiry

This transaction type is used to request the available balance on the cardholder's account. During balance inquiry transactions no funds are moved and are typically performed to prevent transaction declines due to insufficient funds.

2.8 Terminal Capabilities

Since card terminals differ in capabilities and supported features, the card has to discover the terminal's capabilities and features at the start of each transaction to be able to define the flow of the transaction to be made. The card can discover the capabilities of a terminal by making a request to the application hosted in the terminal. The response to the request contains a list of supported features based on which the card proceeds to take further actions.

2.8.1 PIN Entry

PIN Entry capability indicates that the terminal supports the secure entrance of a PIN. Since a terminal may have an external **PED** (PIN entry

device) connected to it, the PIN Entry capability may change in time as it can become unavailable in cases where the PED is disconnected.

2.8.2 Key Entry

Whether the terminal supports the ability of entering a PAN identifier and expiry date manually is indicated by the presence of the Key Entry capability. Since both PAN identifier and expiry date consist only of numeric characters, one may expect that Key Entry capability may indicate the presence of the PIN Entry capability. However, because PIN entry requires strict security requirements, this is not the case.

2.8.3 Chip Reader

To indicate that a terminal supports data interchange with ICCs, the Chip Reader capability is used. Additionally, it also indicates the support of EMV transactions.

2.8.4 Contactless Reader

The ability to exchange card data with the terminal using contactless data transfer technologies is denoted by the Contactless Reader capability.

2.8.5 Magnetic Stripe

Whether the terminal can read card data from a magnetic stripe is indicated by the Magnetic Stripe capability.

2.8.6 Contactless Magnetic Stripe

Contactless Magnetic Stripe capability is the ability to support contactless transactions using legacy communication protocols that do not utilize EMV technology.

2.8.7 Card Capture

Support for physical retention of a card that was inserted into a terminal is indicated by the Card Capture capability. Most commonly, this capability is used in ATMs, where, in case of a detected fraud, the card may not be returned to the terminal's user.

2.8.8 Card Data Output

The ability of the terminal to write data to the card's IC chip is indicated by the presence of the Card Data Output capability. This feature could be leveraged to update the card's internal counters such as the transaction counter or to initialize an EMV process.

2.8.9 Terminal Output

Support for displaying data to the customer either by displaying it on the terminal's display or by printing it is denoted by the Terminal Output capability.

3 EMV Transactions

In the time when magnetic stripe cards were widely used the size and cost of card skimming devices used to make copies of cards grew smaller, a need for more secure payment cards arose. It was required to create a new way that terminals and cards communicate with each other that allowed for safer offline transactions as well as options to host multiple card scheme applications in cards.

In the 1990s began development of a new standard related to card payments in which Europay, Mastercard and Visa companies participated. The standard was based on a standard for smart payment cards and payment terminals and aimed to provide a solution for previously mentioned problems by utilizing ICCs. Usage of ICCs allowed for maintaining various internal counters based on which offline transactions can be authorized or for a more sophisticated dialog between the card and a terminal. Eventually, this new standard was named EMV after the companies that participated in the initial development.

To further combat fraud the set of EMV standards also supports remote execution of code. This allows issuers to blacklist stolen cards and remotely block them for all transactions (including offline transactions) when they are presented at an online terminal.

Because a dialog between the ICC and a terminal has to be established to perform card transactions, EMV cards are considered much harder to be faked as opposed to magnetic stripe cards, which are only read from but do not actively engage in the exchange of data.

As of Q2 2021, 88.55% of card transactions utilize the EMV technology [8].

This chapter covers the most important EMV standards that were used to implement the payment card system of today's world. EMV Book 1 covers both the hardware and the software architecture of payment cards from a high-level overview [9]. The second EMV book focuses on security features of integrated circuit cards and terminals [10], while the third book focuses on card application specification, including data elements and commands that a terminal and a card exchange between each other [11]. Additional requirements that support the acceptance of integrated circuit cards are defined in the fourth book [12]. Last but not least, a design for contactless payments is defined in the EMV Book A [13].

3.1 Hardware Architecture

EMV cards are essentially ICCs, which makes them smart cards compatible with the ISO 7816 standard. These cards are powered externally, typically by a payment terminal when inserted into or tapped onto.

As defined in the ISO 7816 standard, the chip contains 8 external contacts numbered from C1 to C8, but only 5 of them are actively used today. Contacts C4 and C8 are reserved for future use, and the C6 contact is marked as deprecated [14].

The functioning of ICCs is also supported by ROM, EEPROM and RAM memories that are all present on them.

A contactless card is sometimes referred to as a **PICC** (Proximity Integrated Circuit Card) and is powered wirelessly by electricity inducted in the card's antenna by the electromagnetic field of the terminal's contactless card reader. All devices capable of **NFC** (Near Field Communication) technology can be used as payment cards because contactless payment cards communicate with terminals at **RFID** (Radio Frequency Identification) frequency, which NFC technology supports.

3.1.1 Answer to Reset

After the integrated circuit has been powered up by the terminal a signal on the card's `RST` contact is sent, causing any previously persisted state to be erased [9]. When it's reset, it transmits the `ATR` (Answer-to-Reset) message to the terminal to indicate that the card is ready to exchange commands. The `ATR` message also contains a description of supported communication methods and protocols the terminal may utilize.

3.2 Software Architecture

A smart card hosts an operating system, that resides in its ROM memory. Its file system is hierarchically organized and consists of three file types: master files ("MF"), dedicated files ("DF"), and elementary files ("EF") [15, 16].

The root of the file system is a master file, where headers of dedicated and elementary files are stored (see fig. 3.1 for an example filesystem hierarchy).

Directories in the file system are represented by dedicated files whereas data files are represented by elementary files.

In EMV cards dedicated files in the root directory of the file system represent card scheme applications. Application dedicated files are identified using a **FID** (Fixed File Identifier) and an **AID** (Application Identifier)

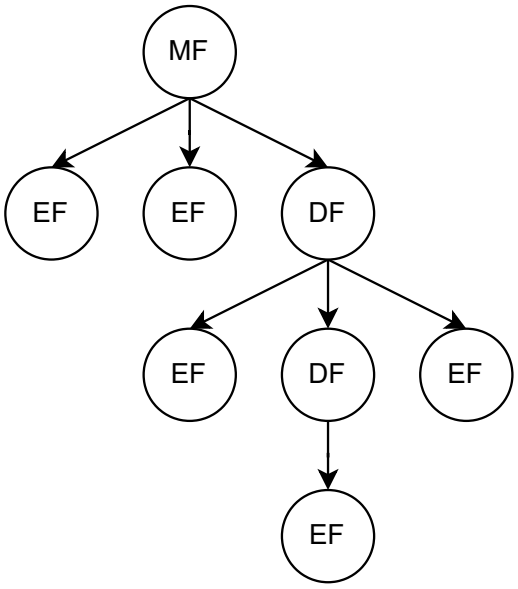


Figure 3.1: A diagram visualizing the ICC filesystem file hierarchy.

that consists of 2 parts (see table 3.1 for example AIDs). The first part is a **RID** (Registered Application Provider Identifier) and is assigned to the application by an overlooking authority, while the second part is a **PAIE** (Proprietary Application Identifier Extension), also known as PIX, which is used by application providers to uniquely identify their applications. Obviously, one card may contain multiple applications.

AID	**Vendor**	**Product**
A000000003101001	Visa International	Visa Credit
A000000003101002	Visa International	Visa Debit
A0000000041010	Mastercard International	MasterCard Credit/Debit (Global)
A0000005241010	RuPay	RuPay

Table 3.1: Example of application identifiers with their vendor and product names [17].

Elementary files are referenced using either an FID or a **SFI** (Short File Identifier) and can be categorized based on their usage. *Internal* files contain data that are specific to the application and *working* files contain data to be accessed by the terminal during communication with it.

The communication unit used during the communication between a smart card and a smart card reader is called an **APDU** (Application Protocol Data Unit). APDUs can be categorized based on whether they originate from a

card reader or a card. The former is called a *command* and the latter is called a *response*. While a command APDU must contain a 4-byte header and data of length up to 65 535 bytes, a response APDU contains two mandatory status bytes and up to 65 535 bytes of data.

3.2.1 Command APDU Structure

Bytes of the command APDU have the following meaning:

- **CLA** – indicates the class of the messsage,
- **INS** – specifies the instruction to be executed,
- **P1** – the first parameter of the instruction to be executed,
- **P2** – the second parameter of the instruction to be executed,
- $\mathbf{L_C}$ – number of bytes the command's data consist of,
- **DATA** – commands's data,
- $\mathbf{L_E}$ – expected number of data bytes in the response APDU.

Inter-industry commands have their CLA byte's most significant nibble set to `0`, while commands proprietary to the EMV standard have their most significant nibble set to `8`. Any other commands are proprietary to card schemes or issuers and aren't covered by the EMV standard. The least significant bits of the CLA byte indicate what communication protocols are used.

3.2.2 Response APDU Structure

Bytes of the response APDU have the following meaning:

- **DATA** – command's data,
- **SW1** – the first status byte of command's processing,
- **SW2** – the second status byte of command's processing.

When a command has been successfully processed the first status byte (SW1) is set to `90`. A warning is indicated by the first status byte being set to `62` or `63`. The presence of any error that has occurred during the processing of a command is indicated by the value `69` or `6A` in the SW1 byte. Other SW1 values are not described by the EMV standard. The value of SW2 provides additional information related to the command's processing.

3.3 EMV Transaction Flow

Throughout a card transaction, the card and the terminal exchange data using APDUs and the transaction flow consists of several steps.

High-level overview of the currently processed transaction's state can be observed by inspecting the value of the **TSI** (Transaction Status Information) register that contains two bytes, and its first 6 bits indicate which stages of the transaction have been finished:

- **bit 8** – offline data authentication stage finished,
- **bit 7** – cardholder verification stage finished,
- **bit 6** – card risk management stage finished,
- **bit 5** – issuer authentication stage finished,
- **bit 4** – terminal risk management stage finished,
- **bit 3** – issuer script processing stage finished.

The **TVR** (Terminal Verification Result) register consists of 5 bytes set by a terminal reading a card and contains additional information about the processed transaction, such as whether the PIN try limit has been exceeded or whether an online PIN has been entered.

3.3.1 BER-TLV Encoding

The smallest data unit of an ICC card scheme application is a data element, that is identified by a name, a description, format and coding. Individual data elements are coded in the **BER-TLV** (Basic-Encoding-Rules – Tag-Length-Value) encoding.

The ASN.1 (Abstract Syntax Notation One) standard defines that the data elements of the TLV encoding are encoded by the tag identifier of the element, element's value length and finally the element's value itself.

The tag identifier occupies at least 1 byte and is not constrained to any length. The first two bits of the tag identifier represent the tag's class (see table 3.2).
The tag class is followed by the `P/C` bit indicating whether the tag is primitive or consists of additional TLV sub-elements. The 5 other bits of the first byte are used to hold the tag number. In cases, where the tag number cannot fit into 5 bits, these 5 bits are set to `1` and other tag bytes follow. The first bit

Value	Class
0	Tag native for ASN.1
1	Tag valid for a specific application
2	Tag dependent on a specific context
3	Tag defined in private specifications

Table 3.2: Tag classes as defined in the ASN.1 standard.

in the following bytes always indicates whether another tag byte follows the current tag byte and the other 7 bits contain the tag number.

In cases where the element's value length is smaller than 128, the length is encoded in a byte that has its first bit set to 0 and the remaining bits set to the length itself. To cover cases in which the length is equal to or larger than 128, additional length bytes have to be used. The first bit of the first byte is set to 1 and the other 7 bits store the number of bytes the length consists of. After that, bytes containing the length follow.

Length data is followed by the element's data. The last data byte might be followed by an end-of-content marker, but that is not the case for EMV transactions as end-of-content markers are not used there.

All TLV data elements used in the EMV transaction processing are listed in the EMV Book 3, Annex A [11].

See tables 3.3 and 3.4 for selected TLV tags used in the EMV standard.

Tag	Name	Description	Length
5A	Application Primary Account Number (PAN)	Contains the identifier of the account linked to the application	up to 10 bytes
9F08	Application Version Number	Version of the payment application	2
9F17	PIN Try Counter	The number of remaining PIN attempts	1

Table 3.3: Example TLV data elements sourcing from a card.

3.3.2 Data Object List (DOL)

To reduce the amount of processing required to be done by the ICC in cases when multiple data elements are requested from the terminal, these data elements are not TLV encoded but rather sent to the card as a single field where all requested data elements are concatenated.

Tag	Name	Description	Length
9F01	Acquirer Identifier	Uniquely identifies an acquirer	6
9F16	Merchant Identifier	Uniquely identifies a merchant when combined with the *Acquirer Identifier* data element	15
9F02	Amount, Authorised (Numeric)	The amount that has been authorized during the transaction.	6

Table 3.4: Example TLV data elements sourcing from a terminal.

The ICC defines a DOL that specifies the content and the format of the requested data. A DOL consists of concatenated entries, where each of them contains a tag identifying the requested data element and the number of bytes the requested data is expected to be consisted of. The tag identifier may consist of up to two bytes and the length must occupy exactly one byte.

Following DOLs are used in the current version of the EMV specification:

- **Processing Options Data Object List (PDOL)** – data residing on the terminal requested by the ICC during the processing of the `GET PROCESSING OPTIONS` command, such as the value of the tag `9F66` (*Terminal Transaction Qualifiers*),
- **Card Risk Management Data Object Lists (CDOL1 and CDOL2)** – used during the processing of the `GENERATE AC` command to compose the input vector for the application cryptogram (described in 3.3.9),
- **Transaction Certificate Data Object List (TDOL)** – provides a list of data elements used when generating the hash value of transaction data,
- **Dynamic Data Authentication Data Object List (DDOL)** – used during the offline card authentication where dynamically generated data from the terminal are used (at minimum, contains the tag `9F37` – *Unpredictable Number*).

The EMV standard also specifies rules that the terminal must follow in specific conditions such as when the length specified in a DOL entry is less than the length of the actual data object in the terminal or when an entry in the DOL refers to an unknown data object.

3.3.3 Application Selection

Since the card may host more than just one payment application, it is required that a mechanism that determines which application should be selected is in place. There are two ways the terminal can enumerate applications present on the ICC.

The first and the most straightforward way to read the list of available applications is to parse a specialized dedicated file `1PAY.SYS.DDF01`, which contains a list of applications hosted on the card. The file can be read using a combination of `SELECT` and `READ RECORD` commands, however, the EMV standard does not require that this file must be present on the ICC.

In cases, where the `1PAY.SYS.DDF01` file is not present on the ICC, the terminal has to query the ICC multiple times using the `SELECT` command for the application the terminal knows and supports. This process of looking up available applications by using a list of AIDs of applications supported by the terminal is called *direct application selection* and can be quite time-consuming if the terminal contains a lot of entries. To optimize this process the terminal can also utilize a partial name matching (querying using AID's prefix) for looking up supported applications.

The list of applications that both the terminal and the card supports is called a *candidate list*. When the candidate list is assembled at the start of an EMV transaction and the list contains more than one application, the terminal may choose a payment application automatically based on internal priorities and rules or prompt the cardholder to select an application. Once it is determined which application will be used, the terminal confirms the selection by sending the `SELECT` command with the selected application's AID to the ICC. The terminal then responds with **FCI** (File Control Information) data elements of the selected application in the command response and the initial processing of a transaction may begin.

The FCI template may contain data elements such as *Language Preference* (preferred language to be used in the terminal), *Application Label* or a PDOL (described in 3.3.2).

3.3.4 Initiate Application Processing

After the application has been selected by the terminal it then proceeds to perform the *Initiate Application Processing* function, which informs the ICC about a new ongoing transaction and provides it with all information related to it. Additionally, the terminal receives an **Application Interchange Profile** (AIP) from the ICC and the list of ICC files to be used in the transaction processing.

Before the function is performed, the TVR and TSI bits must be set to zero. Then the terminal issues the `GET PROCESSING OPTIONS` command providing all data required by the ICC previously specified in the PDOL data element in parameters. If the application's FCI template did not contain the PDOL data element, `8300` data field is sent as a parameter instead and when successfully processed, the ICC then responds with an *Application Interchange Profile* and an *Application File Locator*.

Application Interchange Profile

To provide the terminal with the specification of features supported by the selected application, the AIP consists of two bytes where individual bits indicate whether a particular feature is supported. Support of the following features is currently described there: SDA, DDA, CDA, cardholder verification, terminal risk management and issuer authentication.

Currently, only the meaning of the first byte is described in the standard and the second byte is marked as reserved for future use.

Application File Locator

To describe the data elements to be read by the terminal from the card the **AFL** (Application File Locator) is returned as a response to the `GET PROCESSING` command. It contains a list of entries that each consist of 4 bytes that have the following meaning (when read from left to right):

- **Byte 1** – SFI of the application's **AEF**[1] (Application Elementary File) to be read,
- **Byte 2** – the first AEF record to be read during transaction processing,
- **Byte 3** – the last AEF record to be read during transaction processing,
- **Byte 4** – the last AEF record to be read when composing the input vector for offline data authentication.

Entries defined in the AFL are used right after they are parsed in order to read data from the ICC necessary for further processing of the transaction.

[1] Elementary file specific to an application.

3.3.5 Offline Card Authentication

Today's terminals support offline chip card authentication using asymmetric cryptography and hash functions. A chain of trust between participating entities must be well established and terminals have to be preloaded with public keys of card schemes and card issuers.

Offline card authentication is most often used in cases where the terminal has temporarily lost access to the internet and can't perform online authentication.

During offline card authentication the terminal and the card agree on one of the following authentication types that they both support [18]:

- **Static data authentication (SDA)** – The issuer of the card signs static authentication data using its private key and stores it on the card. During offline card authentication, the terminal decrypts the signed value using the issuer's public key and verifies whether it was signed by the issuer. It must be noted that this kind of authentication can be vulnerable to replay attacks because the attacker could eavesdrop on the signed static authentication value and broadcast it to the terminal during an attack.

- **Dynamic data authentication (DDA)** – In contrast to SDA, this method does not depend on static data value generated by the issuer at the time the card was created, but rather for each authentication request Dynamic Data Authentication DOL and an *Unpredictable Number* (tag `9F37`) are used to assemble an input vector that is later signed by ICC's private key. The terminal then verifies the cryptogram using ICC's public key. This method offers greater security than the SDA method since it is not vulnerable to replay attacks.

- **Combined data authentication (CDA)** – Instead of performing multiple data exchanges between the card and the terminal that might become a performance bottleneck during card authentication, the CDA method utilizes data already provided to the card by the terminal during the *Generate Application Cryptogram* stage to assemble the input vector. Otherwise, the flow of CDA is similar to the flow of the DDA method.

Before any of the aforementioned authentication methods take place both ICC's and the issuer's public key have to be recovered.

Public Key Recovery

During offline authorization, the terminal has to recover the card's public key and verify its validity [10]. It's done via a retrieval of a key certificate from the card that contains a part of the card's public key that is encrypted by the issuer's private key to which the terminal has a matching public key (publicly available). Additionally, the key certificate also contains the expiry date, the owning entity (PAN or BIN) and some additional data. The rest of the card's public key (*key remainder*) is received by the terminal in an unencrypted form. If the terminal successfully decrypts the key certificate and verifies its validity it extracts the part of the public key and its hash value. After that, it combines the key remainder with the now decrypted part of the public key, computes its hash value using the specified hash function and checks whether it matches the hash value present in the key certificate. If it matches, then the card's public key has been successfully extracted.

3.3.6 Cardholder Verification

To help the terminal decide which cardholder verification method should be used for a particular transaction, the card application contains a *Cardholder Verification Method List* (CVM List) in the data element with the `9F42` tag identifier [11].

The CVM list consists of cardholder verification rules (CVRs) that are represented as 2-byte values. In addition to CVRs, the list also contains two amount fields that are referred to as "X" and "Y" values. These values contain amounts in the card application's currency (*Application Currency Code*, tag `9F42`) and are referenced by CVRs to define conditions under which the rules apply.

A CVM rule consists of a CVM Code byte and a CVM Condition Code byte. The first bit of the CVM Code byte is reserved for future use and the 7th bit indicates whether the terminal should proceed to the next cardholder verification method when the current method fails. Other bits identify one of the following methods to be used:

- **Failed CVM Processing** (`000000`) – in cases where this method is used and evaluated the cardholder verification is immediately considered as failed. It is often used to terminate the CVM list to end the cardholder verification when no other matching rule was previously found;

- **Cleartext Offline PIN** (`000001`) – the cardholder should be prompted for the card's PIN that will later be sent to the card in clear-text for offline validation;
- **Online PIN** (`000010`) – the cardholder should be prompted for the card's PIN that will later be sent to the issuer in the PIN Block format for validation;
- **Cleartext Offline PIN and Signature** (`000011`) – same as the *Cleartext Offline PIN* method, but additionally requires the cardholder's signature to be verified;
- **Enciphered Offline PIN** (`000100`) – same as the *Cleartext Offline PIN* method, but the PIN entered by the cardholder is enciphered before being sent to the card;
- **Enciphered Offline PIN and Signature** (`000101`) – identical to the *Cleartext Offline PIN and Signature* method, but the PIN entered by the cardholder is enciphered before being sent to the card;
- **Signature** (`011110`) – cardholder's signature has to be verified by the merchant as described in 2.4.3;
- **No CVM** (`011111`) – cardholder verification method is not required.

Content of the CVM Condition byte defines under which conditions CVM rules apply and refers to previously mentioned "X" and "Y" values (see table 3.5 for possible CVM Condition Code byte values).

During cardholder verification the terminal requests the CVM list from the card's ICC and starts to evaluate the CVM rules, taking the terminal's capabilities and application's currency into consideration. If CVM is unsuccessful, the terminal aborts the processing of the transaction.

3.3.7 Terminal Risk Management

Although the connection to the internet is generally more available and stable than it was back in the days when EMV technology was incepted, terminals still need to be able to perform offline card authorizations and make decisions about transaction's validity without first consulting it with the card's issuer.

The EMV standard defines a set of checks referred to as *Terminal Risk Management* that evaluate the TVR register. Based on the terminal risk management outcome it is determined whether the terminal should force an

Value	Description
0x00	The CVM rule must always be applied.
0x01	The CVM rule must be applied when the transaction takes place at an ATM (unattended cash transaction type).
0x02	The CVM rule must be applied in cases where the transaction is not one of the following types: unattended cash, manual cash and a purchase with a cashback.
0x03	It is acceptable to skip the CVM rule when it is not supported by the terminal.
0x04	The CVM rule must be applied when the transaction is of the manual cash type.
0x05	The CVM rule must be applied when the transaction is of the purchase with cashback type.
0x06 / 0x07 / 0x08 / 0x09	The CVM rule is applicible when the transaction's amount is under/over than the "X"/"Y" value of the application currency.
0x0A - 0x7F	Reserved for future use.
0x80 - 0xFF	Reserved for proprietary use.

Table 3.5: Description of possible CVM Code byte values.

online authorization of a particular transaction or settle with an offline authorization. Three major types of checks are defined: *Floor Limit*, *Random Transaction Selection* and *Velocity Checking*.

Floor Limit

The floor limit represents the amount of money above which it is required that a card transaction is authorized online by the card's issuer. Specific floor limit values are set by acquirers and based on rules defined by card schemes that may differ between various currencies.

To prevent fraudulent transactions that authorize amounts just under the floor limit to circumvent the threshold, the terminal may log authorized transactions to a special log file, and each time before a new transaction authorization it is checked whether the sum of recent transactions performed by a particular card does not exceed the floor limit. Such transactions, which are knowingly made to circumvent the threshold, are in the card industry

referred to as *split sales*.

Some acquirers may also choose to set the floor limit to zero in order to disable offline authorizations and rely solely on online authorizations.

Random Transaction Selection

Random transaction selection forces a percentage of all transactions not exceeding the floor limit to be authorized online. Each transaction has a probability to be flagged as a transaction requiring an online authorization. The properties of the probability distribution used in a terminal are defined by the acquirer and are usually configured in such a way, that the probability of a transaction requiring an online authorization is increasing as the transaction amount gets closer to the floor limit.

Velocity Checking

A great number of consecutive offline transactions may raise a suspicion that the card is being misused for performing fraudulent transactions. Issuers attempt to combat this behavior by implementing a soft and a hard limit to the number of consecutive offline transactions.

The soft and the hard limits are present in the *Lower Consecutive Offline Limit* (tag `9F14`) and *Upper Consecutive Offline limit* (tag `9F23`) data objects. During transaction processing the terminal reads the **ATC** (Application Transaction Counter) value of the last transaction that was authorized online from the card by requesting the *Last Online Application Transaction Counter (ATC) Register* data object (tag `9F13`). This value is subtracted from the card's ATC counter (fetched from the *Application Transaction Counter*, tag `9F36`) and the result of the subtraction is compared to the soft and the hard limit. If it exceeds the soft limit but does not exceed the hard limit, the terminal attempts to authorize the transaction online, however, if the connection to the issuer is unavailable, the terminal attempts to authorize the transaction offline. If the difference exceeds the hard limit, the terminal forces online authorization that must be available and succeed, otherwise, the transaction is declined and aborted by the terminal. In other cases, where the difference is smaller than the soft limit, the transaction is allowed to be authorized offline.

3.3.8 Terminal Action Analysis

After terminal risk management has been completed, the first decision on whether the transaction should be declined offline, authorized offline or sent

to the issuer to perform online authorization is made. It is based on optional *Issuer Action Code* and *Terminal Action Code* data elements. As the names imply the former data elements may reside in the ICC (specified by the issuer), while the latter data elements may reside in the terminal (specified by the acquirer). They are further categorized based on conditions under which they apply.

Offline action codes define when a transaction should be declined and *Online* codes define when an attempt to authorize a transaction online must be made. To cover cases where an online authorization was attempted but could not be completed (for instance, when the internet connection becomes unavailable), *Default* codes can also be defined and specify conditions under which a transaction must be declined when an online authorization was not completed.

Bytes of action codes directly refer to the TVR register and the OR function is used to combine issuer and terminal action codes. Combined bytes of action codes are then compared to the TVR register using the AND function. If the result contains at least one bit set to `1`, then the rule for the given code should be applied.

Action codes are evaluated in the following order: offline, online and default.

If offline codes are not set, then the codes with all bits set to `0` are used. If online codes are not set, then the codes with all bits set to `1` are used (the same applies to default codes).

For example, the 7th bit ($b_{2,7}$ bit) of the second TVR byte indicates whether the selected application is expired. When the $b_{2,7}$ bit of the online terminal action code is set to `1`, it means that if the application is expired the terminal must attempt to authorize the transaction online. If the $b_{2,7}$ bit of the default terminal action code is set to `1`, then the terminal must not attempt to authorize the transaction offline if the online authorization was not finished successfully and the selected application is expired. These two checks would have been skipped if the $b_{2,7}$ bit of the offline terminal action code was set to `1` and the selected application was expired because the transaction would be declined offline.

3.3.9 Application Cryptogram Generation

The last step of the EMV transaction flow is the generation of an application cryptogram, that is performed by the ICC when requested by the terminal. The terminal asks the card to generate an application cryptogram by issuing the `GENERATE AC` command. Response to the command includes

the generated cryptogram as well as other authorization data, and together with transaction data these fields are verified and later used by issuer banks during clearing.

An application cryptogram is computed by gathering input data specified by CDOL1 and CDOL2 data objects that reside on the terminal. This data is then encrypted using a session-specific key derived from one of the terminal's master keys. Aside from a few exceptions and a random number (*Unpredictable Number*, tag `9F37`) the EMV standard does not mandate a particular list of fields to be used in the generation of the cryptogram but recommends using the transaction amount, transaction date, transaction type and the value of the transaction counter.

There are 3 types of application cryptograms that can be issued by the ICC when requested by the terminal:

- **TC** (Transaction Certificate) – generated when a transaction has been authorized offline,
- **ARQC** (Authorization Request Cryptogram) – generated to be used during online authorization,
- **AAC** (Application Authentication Cryptogram) – generated when a transaction has been declined.

During online authorization, the terminal asks the ICC to generate an ARQC that is then sent to the issuer by the terminal for verification. After the issuer decrypts the cryptogram using keys assigned to that particular card and verifies its validity, it generates an **ARPC** (Authorization Response Cryptogram) that is sent back to the terminal. The terminal authenticates the issuer by sending the ARPC in another GAC command or by issuing an `EXTERNAL AUTHENTICATE` command to the ICC if available.

Once the issuer's response has been validated or a transaction has been successfully authorized offline, it is considered finished.

3.3.10 Script Processing

The issuer may choose to include a script in the response to the ARQC to be remotely executed on the ICC. For example, this allows issuers to remotely block cards so they cannot be used in offline authorization when it is suspected that the card might have been stolen or misused for performing fraudulent transactions.

3.4 Key Hierarchy

A chain of trust between card schemes, issuers, terminals and cards is established to allow individual entities participating in the card transaction processing to verify the authenticity of each other's messages and signatures.

At the top of the key hierarchy (see fig. 3.2) stand card schemes as root certificate authorities that generate their own key pairs and accept public keys of issuers for signing. When an issuer creates a new card, the card's public key is signed by the issuer's private key. Cards usually contain more than one key pair that has to be signed, because they use different keys for different actions such as PIN encipherment and personalized data signature. In the key hierarchy used in EMV transaction processing issuer banks and acquirers are usually referred to as L1 nodes while cards and terminals are referred to as L2 nodes.

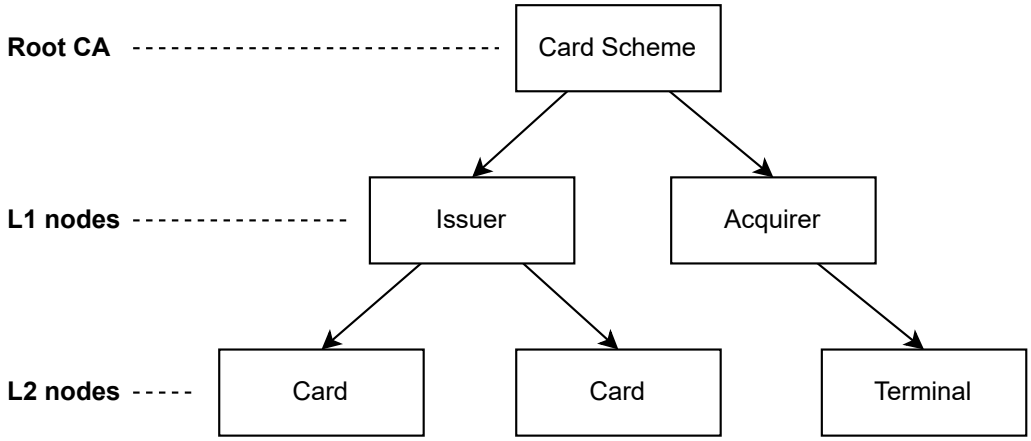


Figure 3.2: A diagram visualizes the chain of trust established between entities involved in the processing of card transactions.

Cryptographic keys play a critical role in the security of card transaction processing and must be handled with extreme caution. For example, in some cases, it is required that process of dual-control (a process that involves two or more entities, usually persons, equally responsible for the physical protection of sensitive functions or information) is used when a terminal is being manually loaded with key-pairs containing keys such as **TMK** (Terminal Master Key), **TEK** (Traffic Encryption Key) or **KEK** (Key Encryption Key).

3.5 EMV Contactless

Contactless payment transactions covered by the EMV Contactless standard offer additional convenience to customers when presenting cards or other payment instruments to the merchant at the point of sale. Technologies such as RFID and NFC allow for an exchange of data between devices in close proximity using unlicensed global radio frequencies. In other words, instead of plastic cards, mobile phones or other smart gadgets such as smartwatches can be used. In addition to the convenience of not having to insert a card into a payment terminal, contactless transactions may also provide some security benefits. For example, a cardholder verification during the processing of a transaction can utilize the capabilities of mobile phones to perform biometric authentication of the user. This includes facial recognition or authentication of the cardholder using the device's fingerprint reader.

In contrast to chip transactions, the card is tapped to the terminal's reader for a short period of time and it for example cannot be guaranteed that it will still be tapped to it when the terminal receives a response from the issuer during online authorization. This also means that in the case of a PIN CVM the card is not available to the terminal after the PIN has been entered by the cardholder, resulting in the prohibition of the offline PIN verification method. The EMV standard only allows the Online PIN CVM for contactless transactions.

However, in most cases, a contactless transaction flow is to a great degree similar to a chip one. The only major difference is in the presentation method and the limited time the card is available to the terminal.

To provide support for environments that do not fully support EMV transactions, the "Magstripe" mode is defined by the EMV Contactless standard. This mode mandates that all transactions are authorized online and that all necessary data for the authorization of a transaction is gathered from the card using an EMV dialogue. This data should be then embedded into the discretionary data part of the Track 2 data before being sent in a legacy format to the issuer.

3.5.1 Kernels

In the EMV Contactless standard, a kernel is the software in the POS system that is used for processing contactless transactions [13]. The standard introduces specification of several kernels to be used in contactless card transactions (see table 3.6). Terminal manufactures and solution providers then use the standard to create kernel implementations.

Kernel	Supported Card Schemes
Kernel 1	JCB, Visa (fallback)
Kernel 2	MasterCard
Kernel 3	Visa
Kernel 4	American Express
Kernel 5	JCB
Kernel 6	Discover
Kernel 7	UnionPay

Table 3.6: List of kernels and supported card schemes [19].

A kernel may reside in a card terminal, a kiosk or a mobile phone and usually provides a simplified interface to process an EMV transaction. It contains a set of functions that allow for establishing an efficient data exchange with the payment card. A kernel eventually communicates with the application present on the card in a similar way as described in 3.2.

The transaction flow when using a kernel is similar to the flow of a classic EMV chip transaction, but the way payment applications are selected differs since the POS system (terminal) has to select a combination of a kernel and a payment application. One kernel may support multiple applications.

Kernels may use proprietary commands and tags to communicate with supported card applications to deliver the best cardholder experience. For example, the Kernel 2 (Mastercard) supports both Contactless Magstripe and EMV transactions with some methods to recover from premature removal of the card from the reader's vicinity. The Kernel 3 (Visa) uses a single `GET PROCESSING OPTIONS` command during the transaction processing and receives an application cryptogram in the answer as opposed to a standard contact EMV transaction where it is required to issue multiple commands to receive the cryptogram. In addition to that, the Kernel 3 supports that a card is tapped for the second time during which any issuer scripts to be processed by the ICC are transferred.

3.5.2 Entry Point Processing

At the start of each contactless transaction, the Entry Point processing that consists of five main functional sections is initiated.

The Pre-processing step is present in variable-value transactions and includes an analysis of the transaction data. For example, it allows the terminal to predetermine which AIDs are available to the current transaction.

This step is skipped in environments where the transaction amount is predefined and does not require merchant interaction.

During the Protocol Activation step, card discovery is started by initiating the polling of the electromagnetic field in front of the contactless card reader. When multiple cards are present in the polled magnetic field, these collisions are detected and the transaction processing may continue only when a link with a single card has been successfully established. This step is comparable to the Anwer-to-Reset interaction of contact chip transaction processing.

Same as in the contact chip transaction processing an application residing on the ICC must be selected. In addition to the application, a kernel must also be selected. The result of the Combination Selection step is a combination of a kernel and a card application which is to be used during the authorization of the transaction. Furthermore, a **PPSE** (Proximity Payment System Environment) file must be present in the ICC's file system under the name of `2.PAY.SYS.DDF01`. This file is used to define the list of supported applications, their priorities and associated kernels. As opposed to chip card processing, where a PPSE file (`1.PAY.SYS.DDF01`) is not mandatory and direct application selection can be used, here a PPSE file is required to allow for quick application selection since the duration during which the card is tapped to the reader is limited.

Once the combination of a kernel and an application is selected, the next step is the Kernel Activation step, during which the selected kernel is activated and initiates communication with the card to continue the processing of the transaction. The result is then processed in the Kernel Outcome Processing step that is described in the section 3.5.3.

The Entry Point processing may be initiated in any of the first four described steps and the standard describes four starting points:

- **Start A** – enters the transaction processing flow at the Pre-Processing step. It is the most commonly used starting point as it is used in standard payment transactions where the merchant sets the transaction amount based on the value of goods the customer wants to buy;
- **Start B** – bypasses the Pre-Processing step and enters the transaction processing flow at the Protocol Activation step. This starting point is used in transactions with fixed amounts or when a card needs to be tapped again to the reader;
- **Start C** – starts at the Combination Selected step and is usually used in scenarios where the selected combination of a kernel and an applic-

ation was not able to process the transaction. Another combination with the highest priority, if not attempted previously, is selected for the next attempt to process the transaction. Eventually, this start might be used several times during the transaction processing until all suitable combinations have been attempted;

- **Start D** – starts at the Kernel Activation step in scenarios where the kernel was previously activated but was restarted during the processing of the transaction, for example, during an online authorization.

3.5.3 Kernel Outcome Processing

The processing of a transaction done by the kernel may result in one of the seven outcomes defined by the EMV Contactless standard. Based on the kernel's outcome further actions are made that may result in the final decision on whether the transaction has been accepted or declined.

Select Next

The Select Next outcome indicates that the selected combination of a kernel and an application is not suitable for the processing of the transaction. It instructs that the terminal should try to use another combination. Suppose all combinations have been attempted, none of the attempts were successful and the last kernel returns this outcome. In this case, the Entry Point returns the End Application outcome to the POS and concludes the transaction.

Try Again

When the selected kernel requires the card to be presented again to the reader the Try Again outcome is returned. This outcome can be a result of "tearing" that happens when the cardholder's card is removed from the card reader's vicinity before the required interaction between the terminal and the card is finished [20]. Another scenario when this outcome can occur is when a mobile phone is used to present a card to the terminal and the kernel requires an action to be performed on the phone before presenting it to the terminal again.

Approved

The Approved outcome indicates that the transaction has been approved and the terminal forwards it to the POS. This outcome is returned when the kernel has successfully authorized a transaction offline or when a response

to an online authorization request from the issuer has been authorized by the kernel.

Declined

When a transaction has been declined the Declined outcome is returned by the kernel and is forwarded to the POS. Similar to the Approved outcome, this outcome is returned during offline authorization or authorization of the issuer's response.

Online Request

By returning the Online Request outcome the kernel informs that the online authorization is required to determine the approved or declined status. The kernel can also indicate whether it requires to be restarted when the response to the online authorization request is received. This outcome is forwarded to the POS.

Request Online PIN

The Request Online PIN outcome is returned by the kernel and forwarded to the POS when the issuer requests the cardholder's PIN to authorize the transaction.

Try Another Interface

The Try Another Interface outcome is returned by the kernel in the following scenarios:

- the kernel indicates based on the terminal configuration data that another terminal interface (e.g., contact chip or magnetic-stripe) should be used to process the transaction since it was not able to complete the transaction with the selected contactless card application,
- no compatible contactless card application was found,
- it was required by the issuer in the response to the online request that another interface should be used to process the transaction.

End Application

The kernel returns the End Application outcome in the following scenarios:

- transaction processing has been completed and the kernel requires no further actions,
- the kernel requires a restart after the card has been removed,
- the kernel experienced an unrecoverable application error that will not be resolved if the transaction is attempted again with the same application,
- no compatible contactless card application was found and the cardholder should present another card.

4 Payment Card Tokenization

Card tokenization is most often used to increase the cardholder's privacy and security of his credentials by hiding sensitive card data, such as the card's identifier and its date of expiration, from other transaction processing entities, reducing the attack surface and allowing the use of other devices instead of the card.

Furthermore, it improves the comfort of using contactless payments by allowing mobile devices such as mobile phones or smartwatches to be used as payment instruments. Rather than carrying multiple cards in a wallet or a pocket, the cardholder can have these cards loaded into his digital wallet and simply select the card to be used at the point-of-sale using the digital wallet's user interface.

It is most often used in technologies such as Google Pay [21] or Apple Pay [22] that allow consumer mobile phones running on the iOS or the Android system to securely store payment card tokens, making it possible to use them as contactless payment cards. Both of these technologies use the NFC technology to transmit card tokens to contactless card readers.

4.1 Tokenization Architecture Overview

The EMV Co. card tokenization guidelines introduce the **TSP** (Token Service Provider) entity that participates in the card tokenization process by being responsible for generating and managing tokens that are created based on cardholder data [23]. TSPs are also responsible for storing sensitive cardholder data in a secure manner. To tokenize the card the TSP must also communicate with the card's issuer and authorize the tokenization request. Both Visa and Mastercard offer tokenization services.

Entities that submit card tokenization requests to TSPs are in the EMV Co. card tokenization guidelines called *token requestors*. In the world of mobile payments, these are usually Google and Apple companies that each offer their own digital wallet product in the form of a mobile application that allows cardholders to load their cards into their mobile phones.

See Figure 4.1 for a high-level overview of the card tokenization process.

When the cardholder decides to tokenize his card, he opens the digital wallet application (e.g., Google Pay or Apple Wallet) and submits his card data into it. This most often includes the card's PAN, its date of expiration and the name of the cardholder.

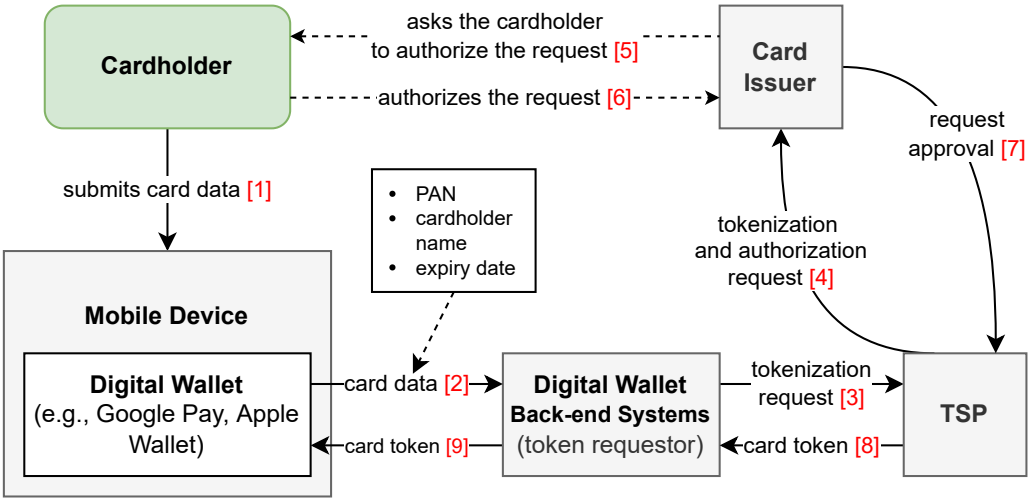


Figure 4.1: Process of card tokenization involving various entities.

The card data is then processed on the back-end systems of the digital wallet provider in order to create a tokenization request that is sent to a TSP, who performs initial processing and based on the PAN decides which issuer should he ask for tokenization request authorization [24].

The issuer of the card requires that the request is authorized by the cardholder. This is usually done by sending an SMS with an authorization code to the cardholder or requiring him to confirm the request in the issuer bank's web interface. Once the TSP receives the authorization response (that might include additional data required to tokenize the card, such as token-specific cryptographic keys) the card is tokenized and the token is sent back to the back-end systems of the digital wallet in order to be loaded into the cardholder's mobile phone.

Because it is required that the token can be used in traditional payment networks its format is identical to the format of a PAN, however, to be able to distinguish between real and tokenized cards, tokens are assigned identifiers from special BIN ranges.

Additionally, to participate in EMV transactions, digital wallets have to be EMV compliant and support the generation of ARQC cryptograms during transaction authorization. To do so the wallet must be in possession of the cryptographic key specific to the generated token. Because of that, in addition to the generated token the TSP sends the encryption key that is tied to it. This key is then used during the generation of ARQC cryptograms to encrypt the data specified in the CDOL tag (as described in 3.3.2).

4.2 Apple Pay

Apple Pay was first announced in September 2014 and provided support for NFC payments on the mobile devices iPhone 6 and iPhone 6 Plus [25]. Initially, iPhone users had to add their payment card to their iTunes Store account to be able to use the phone to pay for goods at merchants that had contactless card terminals. Later during the lifetime of the product, cards to be tokenized had to be submitted using the Apple Wallet application. In the Apple Pay release press, Apple claimed that at that time Apple Pay was supported by the three major payment networks: American Express, MasterCard and Visa, and by the most popular banks including Bank of America, Capital One Bank, Chase, Citi and Wells Fargo, that represented up to 83 percent of all credit card purchase volume in the US. Apple Pay was released later that year in October as a part of the free iOS 8 system update.

Not a lot of technical details on how does Apple Pay work is publically available, however, the following text is based on the brief description of Apple Pay that can be found on the official Apple Support website [22].

When a cardholder adds his card to the Apple Wallet application to enable the Apple Pay feature on his device, the information entered is sent to Apple servers where the card's payment network is determined. On Apple servers, the card data is encrypted by a secret key based on the card's payment network and is used to create a request to the cardholder's bank via a TSP to generate a Device Account Number that is linked to the card being tokenized. The generated Device Account Number is then encrypted by a key inaccessible to Apple and delivered to the device, where it is stored in a protected component called Secure Element.

Secure Element is a tamper-resistant secure microcontroller capable of hosting applications and their data that is isolated from the device's operating system [26]. Different form factors of a Secure Element exist and may include embedded and integrated microcontrollers, SIM cards, microSD cards or smart cards. It is most often used to store high-value sensitive data such as passwords, cryptographic keys or card data. On iPhones, applications hosted on a Secure Element have direct access to the device's NFC controller.

When Apple Pay is used in stores to pay for goods and a device is tapped to a card terminal, the encrypted Device Account Number is transmitted directly from the device's Secure Element bypassing the device's operating system (see fig. 4.2).

Alongside the Device Account Number, a one-time transaction-specific

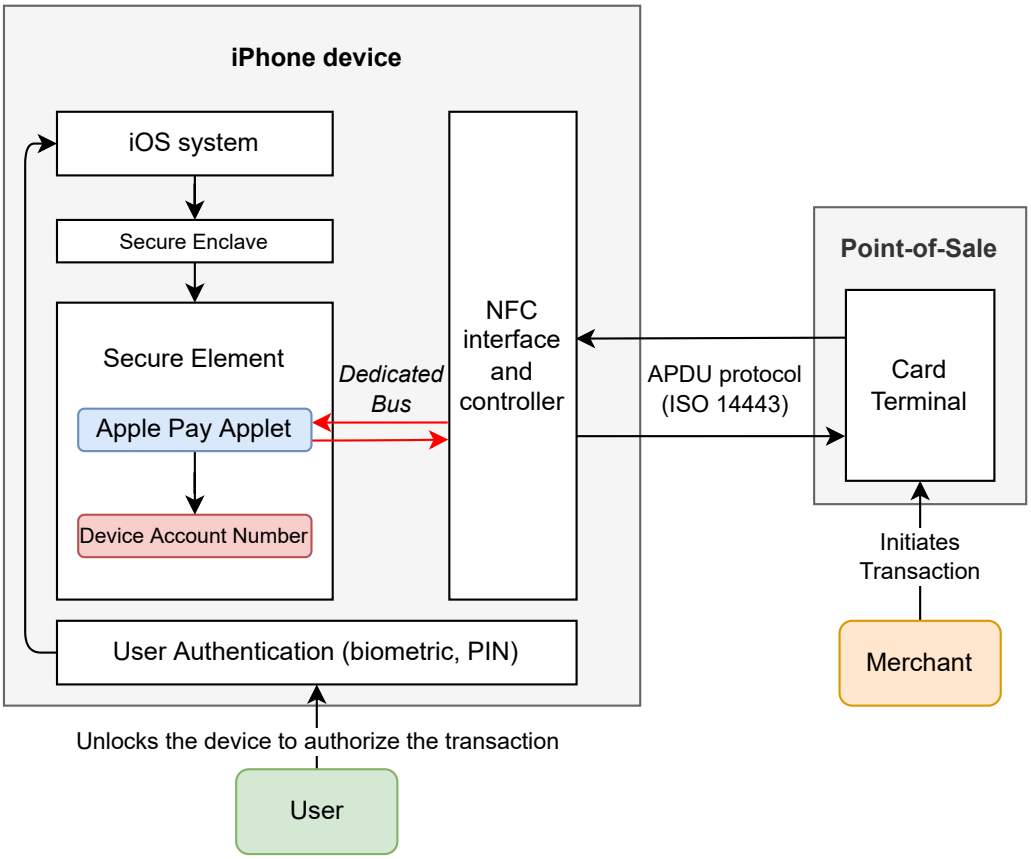


Figure 4.2: A diagram visualizing Apple Pay card emulation architecture.

security code (that is based on transaction data and the transaction counter) and an application cryptogram are sent to the card terminal, where it is forwarded to the payment network based on the BIN identifier. The payment network uses a TSP to translate the Device Account Number back into the real PAN identifier that is then used to process the transaction as if it was a standard card transaction.

Because the Device Account Number is stored in a Secure Element located inside a mobile device, it can be used to present the tokenized card even when the device is not connected to the internet. Additionally, since Apple claims that cardholder's data is not saved in any way when being forwarded to the cardholder's bank, card details cannot be leaked even if Apple's back-end systems were compromised as they are present only in the device's protected storage.

It should be noted, that while Apple Pay can make contactless payments more convenient, it can also introduce new vulnerabilities to them. One example of such vulnerability is one that was discovered after Apple added a new "Express Transit/Travel" feature to Apple Pay. This new feature allowed

to use Apple Pay in public transport in selected cities without requiring to unlock the iPhone, bypassing user authentication. This feature is available in London (TfL), New York City, Portland, Chicago, Los Angeles, Washington, Beijing, Shanghai, Hong Kong and Japan) [27], and a group of researchers from the University of Birmingham discovered that a non-standard sequence of bytes is being sent in Transport For London ticket-gate terminals, which allows performing contactless transactions without user authentication [28]. This group of researchers was then able to exploit this sequence in a relay attack to perform transactions with standard card terminals without any user interaction. Additionally, it enabled to authorize transactions over the CVM limit without having to unlock the device.

4.3 Google Wallet

In 2011 Google announced its first mobile payments service – the Google Wallet [29], a result of the partnership with companies MasterCard, Citi, First Data and Sprint, enabling NFC capable mobile phones to emulate payment cards that could be used instead of traditional cards (see fig. 4.3 for a screenshot of the Google Wallet mobile application). In addition to mobile payments, the application offered access to special sale offers, loyalty rewards, promotions and discounts.

In a similar way Apple Pay emulates payment cards, Google Wallet also used a Secure Element component to enable mobile devices to be used as payment cards. When a device with Google Wallet enabled was presented to the card terminal, the payment application located in the device's Secure Element transmitted all the required card data to it and if necessary, the application prompted for the 4-digit PIN (that was set by the user during the initialization of the application) before allowing to use the card emulation feature.

It should be noted that the usage of a Secure Element does not implicitly mean that the solution was secure in all aspects. A great example of this is a PIN exposure vulnerability that was discovered soon after Google Wallet was released [30]. The hash of the application's PIN, which was required to open the application and authorize transactions, was not stored in the device's Secure Element component, but rather in the Android OS file system, precisely in the folder specific to the Google Wallet application. This is a major issue as Android applications may access data of other applications on rooted devices without any limitations and since the application's PIN consisted only of 4 digits, the hash of the PIN would be enough for the

attacker to be able to reveal the original PIN using a brute force attack, trying out all possible combinations of the PIN.

Even though using a Secure Element may introduce benefits related to the overall security and privacy of the solution, its usage was one of the main downfalls of Google Wallet as only a limited subset of Android devices was equipped with it. Additionally, Google was unable to force device manufacturers to include a Secure Element component on their devices, and so the product was not widely adopted by the public.

Google Wallet was eventually replaced in 2015 by Google's other mobile payments service – Android Pay [31].

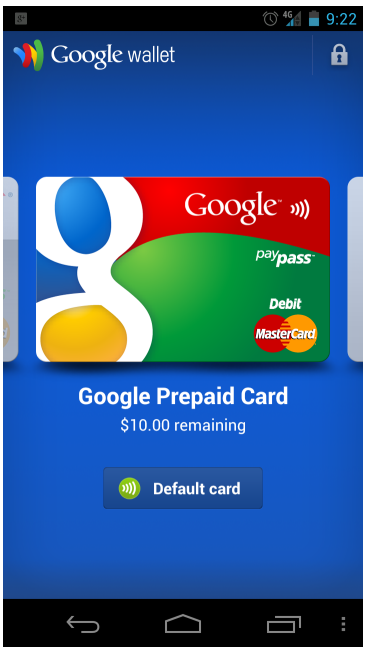


Figure 4.3: Screenshot of the Google Wallet application.

4.4 Google Pay

Google Pay (formerly Android Pay) can be described as a digital wallet platform that enables users to make contactless payments with Android mobile devices, such as phones, tablets or watches. It also provides online payment features, that make checkouts in online stores easier and more secure [21].

In some aspects Google Pay works similarly to Google Wallet, however, it does not require a dedicated PIN to be entered to unlock the application. Instead, it relies on the system authentication services used by users to

unlock their devices.

To enable mobile devices to be used as payment cards it uses the process of card tokenization. Users simply put their card credentials into the Google Pay application and after authorization, the card is tokenized and a token is stored in the application, being available to be used in stores to pay for goods.

Google does not share many details regarding the implementation of its mobile payment solution, and the following text is mostly based on the brief description from the Google Pay support website [32].

For transactions under the CVM limit no user authentication is required, however, the device's screen must be powered on[1]. During authorizations of transactions above the CVM limit it is required to unlock the device (e.g., using the device's PIN, unlock pattern or biometrics) to perform CDCVM. In some cases, typically when multiple transactions in a short period of time are performed, the Google Pay application may require CDCVM even when the device is unlocked.

When compared to products like Apple Pay or Google Wallet there is a great difference in the way cards are emulated. As opposed to the aforementioned digital wallet solutions, Google Pay does not use a Secure Element to emulate payment cards [24]. Instead, it uses the **HCE** (Host Card Emulation) software architecture to transmit card data via NFC [33]. This means that card emulation is being done in an application running in the host operating system, rather than in a separate secure component (see fig. 4.4). Running in the host operating system introduces a great attack surface that attackers may theoretically exploit to steal cardholder data or perform fraudulent transactions. The NFC interface of a device may also be accessible to other applications running in the system when a device is presented to a card terminal.

One of the reasons why Google may have decided to ditch the dependency on a Secure Element component to emulate cards in favor of HCE is that when using HCE the solution does not mandate that a device is equipped with a special hardware component, allowing it to operate the solution on a wide variety of devices [34].

To mitigate some of the security risks that result from the absence of a Secure Component in the solution, Google Pay takes a different approach to the management of card tokens. During card tokenization the card token alongside the token-specific cryptographic keys is stored in Google cloud

[1]After 3 consequent transactions without CDCVM a CDCVM is required even for transactions under the CVM limit.

servers[2]. The mobile application then receives several one-time tokens derived from the main card token that is stored in the cloud. Each of these tokens can only be used once to perform a payment transaction, and after a token is used it is disposed.

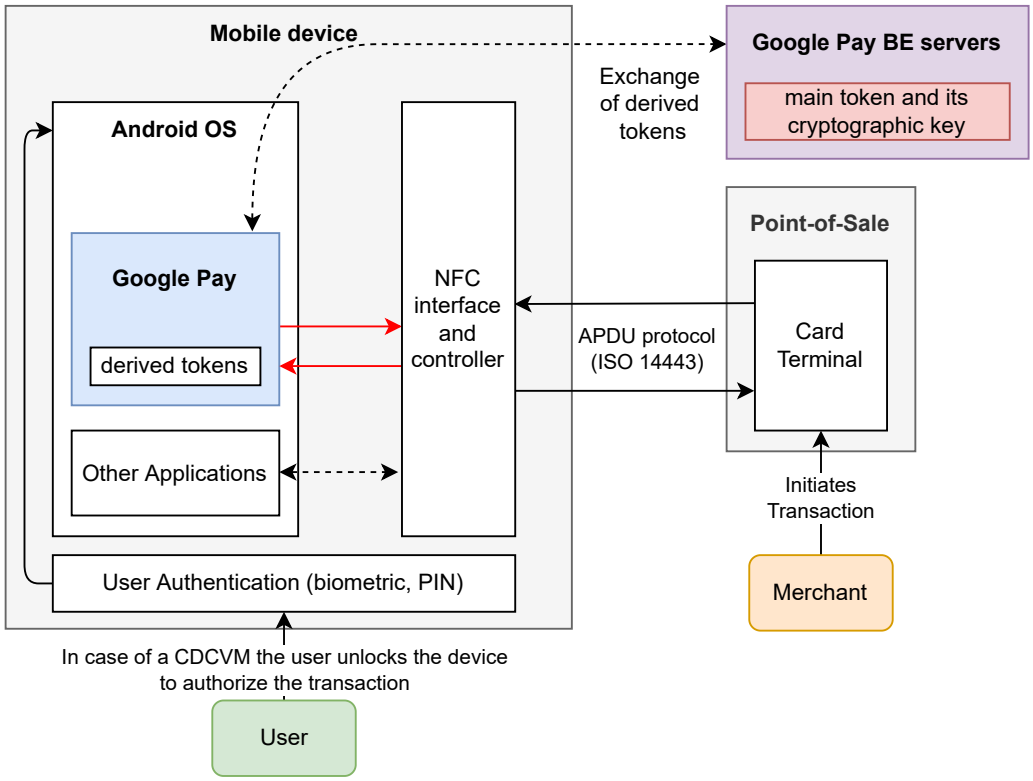


Figure 4.4: A diagram visualizing Google Pay card emulation architecture.

As a consequence of this, Google Pay can be used only for a limited number of transactions when the device is offline. Once the device runs out of card tokens, no more payment transactions can be performed and the device must go online in order to receive additional tokens from the Google Pay servers.

During testing, the author of this master's thesis was unable to run out of available tokens on his device and he managed to perform 25 consequent transactions when his test device was offline. This may imply that number of tokens fetched from the Google Pay back-end servers may be specific to the environment of the cardholder, including his bank issuer, spending habits or the country he is currently located in.

[2]This is often referred to as *cloud-based Secure Element*.

5 Software-based PIN Entry on COTS

Software-based PIN Entry on **COTS** (Commercial off-the-shelf) solutions allow to perform EMV contactless and contact transactions using merchant's consumer devices to enter a PIN using a secure PIN application, a **PCI SSC**[1] (Payment Card Industry Security Standards Council) verified secure card reader, and a back-end for transaction processing and monitoring the COTS device for fraudulent behavior and it's integrity. This chapter covers the **SPoC** (Software-based PIN Entry on COTS) standard published by PCI SSC [35].

The main advantage of SPoC solutions is the reduction of the initial cost when a merchant applies for a card terminal, which may lead to an increasement of merchants that accept payment cards. Previously, only specialized and certified devices were allowed to capture a customer's PIN. This changes with the introduction of SPoC solutions, that make PIN entry on commercial devices possible (see fig. 5.1).

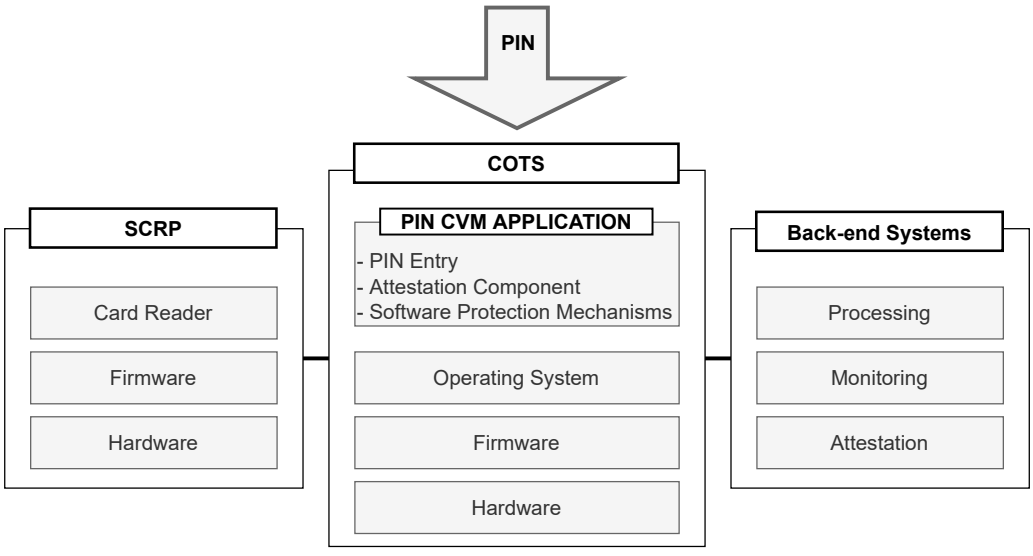


Figure 5.1: A diagram visualizing a SPoC solution.

An **SCRP** (Secure Card Reader for PIN) device provides the system with the ability to read protected cardholder data sourced from the payment

[1] An established entity that oversees policies and technologies used to process card transactions (`https://www.pcisecuritystandards.org/about_us/`).

instrument (card). It also decrypts and encrypts PIN data received from the PIN CVM application and translates it into a required PIN-block format when needed during online PIN verification. The SCRP used in a SPoC solution must be PCI certified and listed on the PCI SSC website[2].

To allow a PIN entry a PIN CVM application resides on the COTS device, providing a secure UI for PIN entry. It encrypts the PIN entered and delivers it to the SCRP for further processing. To maintain its own integrity against attacks it contains various software protection mechanisms and periodically passes attestation[3] health-check data to the monitoring component.

The standard defines back-end as the component that performs following functions for the solution:

- **attestation** – processing of health-check data from the PIN CVM application,
- **monitoring** – system monitoring, alert processing, and mitigation of suspected threats and attacks against the system,
- and **processing** – processing of encrypted cardholder details and PIN data from the SCRP to perform a payment transaction.

The COTS may be any device that is capable of hosting the PIN CVM application. It is operated by the merchant and is handed over to the customer when a PIN entry is required for the transaction to be processed. The standard does not require the device to be PCI certified.

The flow of a PIN transaction can be summarized in the following steps:

1. The PIN CVM application and the SCRP in the SPoC system are securely initialized with all required keys.
2. PIN CVM application establishes a secure connection channel with the back-end monitoring system.
3. Security status of the whole system is evaluated by the back-end monitoring system using the attestation component.
4. An EMV card is presented to the SCRP.

[2] `https://www.pcisecuritystandards.org/assessors_and_solutions/pin_transaction_devices`

[3] A process of determining the current security state of a prover (an entity that proves its security status) based on measurements defined by a verifier (an entity that verifies the security status of another entity).

5. A PIN entry screen is rendered on the COTS platform by the PIN CVM application and after a PIN is entered it is enciphered and sent to SCRP.

6. Cardholder's data is enciphered by SCRP using preloaded keys and sent to the back-end.

7. The payment transaction is processed by the back-end.

The SPoC standard splits requirements into modules and submodules, each covering an area of the solution that is vulnerable to attacks. The following text summarizes each module.

5.1 Core Requirements

Since the system relies on publicly available devices (such as the merchant's mobile phone) for PIN entry, it must be taken into consideration that an attacker trying to steal cardholder and PIN data could have full access to the software on the unknown and untrusted platform. As opposed to specialized **PED** (PIN Entry Device) hardware that has all necessary protections implemented within the device itself, no assumptions about the COTS' integrity and security can be made, and therefore the PIN application and the system as a whole must proceed with extreme caution when capturing and processing sensitive data.

To ensure the protection of cardholder's PIN and data and to support secure mobile payment-acceptance transactions all solution security requirements must work together in concert.

All solution providers are responsible for making sure that all components of their solutions meet all core security requirements. Without it, the prevention of theft of data or data manipulation is not guaranteed and the solution won't be certified for usage in the production environment.

5.1.1 Protection of Sensitive Services

Functions affecting processes that support the sensitive data (cryptographic keys, PINs and cardholder data) are called *Sensitive services*, and their identification, integrity and availability are required. These services must not in any way reveal or modify sensitive data when used.

All sensitive services used by components present in the solution must be properly documented and the documentation has to be updated at least an-

nually. Furthermore, documentation is mandatory for all critical processes, such as key loading and signing of software component binaries.

A dual-control process must be present in the act of generating cryptographic keys used for digital signatures.

5.1.2 Random Number Generation

Since the generation of random numbers, that are unpredictable and unknown to others, plays an essential role in the system's security (and in cryptography in general), all random numbers, when required, must be generated using a process utilizing sufficient entropy to reduce the risk of replay attacks. Because of that, each dependency on random number generation must be properly documented.

For security purposes, all random numbers generated on the COTS device must be seeded from a value that has been generated from an **RNG** (Random Number Generator) on a PCI-certified SCRP device. This is not required in cases when the native RNG of the COTS' OS cannot be seeded during establishment of secure communication protocols.

This requirement is caused by the fact that sufficient entropy in COTS devices is not guaranteed, and because of that, the seed has to be taken from a secure and trusted source, such as an SCRP that has undergone an evaluation and passed a PCI certification.

Furthermore, the module also requires that all random numbers generated on the back-end used for security purposes must be seeded from an RNG that conforms to the FIPS 140-2 Level 3 or from a PCI-approved **HSM** (Hardware Security Module).

5.1.3 Acceptable Cryptography

The use of any cryptographic algorithms must be well thought out and solution providers should not implement their own proprietary cryptographic algorithms, but rather use such algorithms, that conform to global industry-standards and are considered secure.

For the solution's security assurance to be evaluable a documentation listing all cryptographic processes and operations used must exist. This applies to cryptographic algorithms used, key identification and hierarchy, key generation and key agreement processes.

The minimum length of keys used is specified as well as the requirement to use a unique key per every session to protect the system against replay attacks when components in the system communicate.

To limit the consequences of a potential compromise of a key it is required that no key used in the system may serve more than one purpose. Each key and its usage description must be present in the documentation.

Any signatures and fingerprints of keys used should not be created in a way that it reveals any details about them, reducing the threat of key compromise. Specifically, it specifies that **KCV** (Key Check Value) values should be limited to five bytes or less.

A mechanism to monitor and identify expired keys must be implemented as none of the keys used mustn't be expired since that could be an indication of a security breach.

5.1.4 Key Management

As the security and integrity of keys used in cryptographic processes are critical to the system, it is required to approach key management with great care, as the compromise of keys used could lead to invalidation of the system's integrity and credibility. This means that solution's approach to key integrity, security, and lifecycle management is essential and must be documented to be evaluable by a third party.

The solution must be very delicate whenever private or secret keys are manipulated with. Such keys must be stored only in approved forms such as the following:

- encrypted using a key of equal or greater length,
- stored within a **SCD** (Secure Cryptography Device),
- or managed as two full-length components or more.

Mechanisms of key revocation and guidelines on actions to be taken whenever a security threat or incident is suspected are mandatory.

Additionally, it is also required that audit logs must be kept for all activities related to key management.

5.1.5 Development

The standard also imposes broad development process requirements, such as enforcement of code reviews, penetration tests and proper developer training to ensure all secure development practices are followed.

5.2 PIN Cardholder Verification Method Application Requirements

The second module defines requirements that apply to the PIN CVM application and cover its fundamental aspects.

5.2.1 Development

Each instance of the PIN CVM application must be uniquely identifiable by the back-end systems and it is prohibited to communicate with unknown or untrusted SCRPs.

To protect the cardholder's data, any form of screen capture must be prevented whenever the application is in front. This also applies to cases when the application is minimalized and its preview is shown in the task manager.

In addition to PIN Entry features and components, the PIN CVM Application also contains features that are not directly related to security services and PIN entry such as a general merchant UI. To be able to make updates to such features without affecting security code, these parts must be logically separated so they can be independently changed.

Since protection against tampering, reverse-engineering and fault injection is mandatory, all measurements implemented to prevent such attacks in the application must be documented.

Both the data-flow diagram, describing the flow of PIN processing, and the block diagram, displaying the flow of sensitive data through the system, have to be present in the documentation.

Furthermore, to provide merchants with guidance regarding how to ensure the PIN is entered in a way that it cannot be observed by bystanders, an application instruction manual must be maintained.

Any additional cardholder data entered during the processing of a transaction can be viewed in clear text only during its initial entry and only for the purposes of error correction by the cardholder. If this data has to be presented again during the transaction processing it must be masked so that it does not reveal any data correlatable to the cardholder.

One of the requirements specifies that the application's buffers containing sensitive data must be explicitly cleared whenever a transaction has finished or a tamper-detection event has been observed.

5.2.2 Secure Provisioning

All platforms supported by the solution must be defined and the PIN CVM application must not be operated on unsupported systems since they do not receive security updates patching discovered vulnerabilities. Additionally, the standard imposes the following requirements on supported platforms:

- the platform's operating system must validate its integrity on boot as well as enforce a mandatory access control framework,
- application's signature and checksum must be verified before its installation or execution by the platform,
- a mechanism must exist to prevent applications that are not present in the foreground from accessing details regarding touch events.

To support the authenticity of the PIN CVM application the standard requires that the application might be installed using only the application store of the operating system. This also applies to any future updates to the application.

After the initial installation of the application, it is required that during the first execution it becomes uniquely identifiable to the back-end systems. This includes white-box keys used in the PIN CVM application that must also be unique to each application's instance. Any initial data, such as cryptographic keys, must be securely downloaded and stored safely.

5.2.3 Tamper Checks

To reduce the ability of potential attackers to perform tampering with the PIN CVM application using the process of reverse-engineering, the application must implement tamper-resistance measures to code that is involved in the use of security features. One of the measurements that can be implemented may be code obfuscation, which makes it more difficult to perform useful code decompilation for the attacker. All tamper-resistance measurements must be documented.

When a COTS device has been rooted or jailbroken, it is considered vulnerable to malicious tampering and the PIN CVM application must be able to detect it and not accept PIN data.

5.2.4 PIN Entry

During PIN entry the cardholder enters their PIN using the PIN CVM application that must render a custom keyboard because the keyboard provided

by the OS is susceptible to spoofing and whenever one of the following events happen during a PIN entry, the session must be terminated and all session data cleared:

- screen focus of the application lost (includes switching between applications),
- attempts to perform screen capture,
- application is not running in a full-screen mode,
- device sensors have been accessed by any other application,
- application is running in developer or emulator mode.

It is not allowed for the application to give away any visible or audible signal related to the touch event, since it may give away the cardholder's PIN. Additionally, the PIN entered must be fully masked and the application mustn't cache it.

The PIN entry is available only for EMV-based transactions that are processed online.

5.2.5 PIN Encryption

The PIN entered must be immediately encrypted and remain encrypted when sent to the SCRP. The application must protect the entered PIN from any extraction attempts till it's present in the memory.

PIN encryption keys in the application must be securely established and adhere to requirements specified in Appendix C of the standard. This also applies to all cryptographic algorithms related to PIN encryption.

It is mandatory that all white-box cryptography keys are changed monthly, at a minimum.

5.2.6 Audit Logs

To help with intrusion detection, problem identification and reconstruction of events, the PIN CVM application must securely send logs to the back-end monitoring system. Logs created must contain the information required by the standard and must support the reconstruction of the following events:

- access to security functions and application's authentication mechanisms,

- application lifecycle events (initialization, stopping or pausing),
- access to PIN data.

However, it is forbidden that logs contain any data correlatable with PIN data.

5.3 Back-end Systems – Monitoring/Attestation

The third module defines requirements related to components responsible for monitoring and attestation.

The interaction between a verifier and a prover to evaluate the prover's security and integrity is called attestation, during which the verifier is considered trusted and the prover is considered untrusted. The verifier predefines measurements and thresholds on which the evaluation is based. The process of attestation is crucial and must be performed periodically, as it provides the required assurance to the verifier that the prover can be trusted to accept and process sensitive data.

5.3.1 Attestation Types And Components

Different attestation types based on possible locations of the prover as defined by the standard are listed in Table 5.1:

Type	**Prover**	**Verifier**
1	SCRP	Back-end / PIN CVM application attestation component
2	COTS platform	Back-end / PIN CVM application attestation component and SCRP
3	PIN CVM application attestation component	Back-end attestation component and SCRP

Table 5.1: Attestation types based on the location of the prover.

The diagram of the attestation process can be found in Figure 5.2 and the annotated arrows have the following meaning:

- **1a** – the SCRP is verified by the PIN CVM application, but since the PIN CVM application itself has to be verified, further verification of SCRP is required,

- **1b** – the SCRP is verified by the back-end system, and its integrity is assured,
- **2a** – the COTS platform is verified by the PIN CVM application, but since the PIN CVM application is running on the COTS platform, further verification is required,
- **2b** – the COTS platform is verified by the back-end system, and its integrity is assured,
- **3a** – the PIN CVM application is verified by the back-end system, and its integrity is assured.

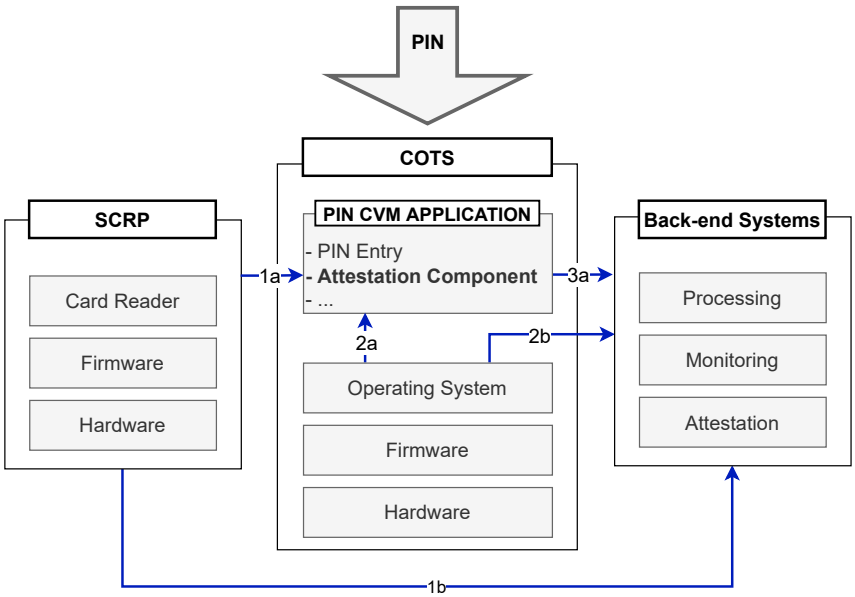


Figure 5.2: Attestation flow diagram.

Additionally, the SCRP may also perform verification of the PIN CVM application and the COTS platform to further increase the security of the system as a whole.

5.3.2 COTS System Baseline

The solution provider has to define a set of requirements that COTS devices and operating systems must fulfill for the PIN CVM application to be executed there. All processes in use that determine the subset of all currently deployed COTS platforms must be documented as well as the process that is used to discover new bugs and vulnerabilities in the system.

If a COTS platform becomes unsupported (for example due to a new vulnerability discovered) all COTS devices using that platform must be prohibited from processing transactions.

5.3.3 Attestation Mechanism

It is required that the process of attestation must be performed whenever the PIN CVM application has been initialized or when five minutes have passed since the last attestation has been made.

Since a potential attacker could use the attestation mechanism to introduce a forged component into the system all attestation messages and communication must be cryptographically signed.

Similarly to the code of security features, all attestation code implemented in the PIN CVM application must be protected by tamper-resistance features.

As some attestation messages may require a manual process (e.g., during a potential tamper event) they must be escalated to vendor staff, which validates and actions them within 48 hours.

Any manual updates must be done according to the documented procedures and deployment of such changes must utilize the dual-control process.

5.3.4 Attestation of SCRP (Type 1 Attestation)

Attestation of the SCRP must occur whenever the system has been initialized, the SCRP is reconnected to the COTS platform, or when it has been requested by the monitoring environment. Additionally, prior to the first transaction, the attestation must also occur. The monitoring system may also poll the SCRP for attestation at unpredictable intervals.

During attestation of the SCRP, it has to be verified at a minimum that SCRP's identifier matches the identifier of the SCRP associated with the COTS, its firmware version is supported and that it is operating in a secure state (it hasn't been tampered with).

5.3.5 Attestation of COTS (Type 2 Attestation)

The following events must be detectable by attestation mechanisms of the COTS:

- the COTS platform OS or the PIN CVM application has been tampered with,

- the PIN CVM application has been executed in developer or debug mode or an emulator is used to run it,
- relay attacks were performed on the PIN entry component,
- attempts to root the COTS platform OS were made.

The COTS platform must undergo an attestation at a minimum whenever:

- the PIN CVM application is started and prior to the first transaction of the day,
- the PIN CVM application is minimized and brought to the front,
- initiated by a monitoring environment request,
- major changes were made to the solution configuration.

Attestation responses of the COTS platform must also include complete configuration information such as SCRP and merchant ID, version of the application and platform's firmware. However, responses mustn't leak information about the attestation mechanism implemented or interrupt an ongoing payment transaction.

To protect against man-in-the-middle attacks it is required that responses are unclonable and complete within an expected timeframe.

5.3.6 Monitoring Environment Attestation of PIN CVM Application (Type 3 Attestation)

A prerequisite for the attestation of the PIN CVM application is the successful attestation of the COTS platform. The solution provider must also define a set of rules for analysis of the attestation responses and have a risk-severity rating assigned for it.

Requirements specifying when attestation of the PIN CVM application has to occur are similar to those that define when attestation of the COTS platform must occur.

The solution provider must also define a minimum PIN CVM application supported version and enforce that participation of unsupported versions of the PIN CVM application in payment processing is prohibited.

Additionally, to prevent abuse of the PIN CVM application attestation mechanism, the solution must have controls established to protect the subversion of the prover (PIN CVM application), such as defense against DDoS or data poisoning attacks.

5.3.7 Basic Protection

Back-end components related to attestation and monitoring that reside in **CDE** [4] must be PCI compliant and all attestation traffic must be encrypted and signed to ensure that unauthorized components and subjects cannot gain access to attestation data.

5.3.8 Operational Management

Processes that support the operation of the monitoring environment must be well documented and the staff of the monitoring environment must be provided with up-to-date security training. Reviews, that verify operational security processes, must be performed at least quarterly.

5.4 Solution Integration Requirements

The Solution Integration Requirements module defines requirements mainly related to the communication between components of the solution.

5.4.1 Pairing of Disparate Components

It is required that both the PIN CVM application and SCRP components are uniquely identified and their pairing is validated before any communication between the PIN CVM application and the SCRP happens. In addition to that, any PIN-based transaction must be associated with a specific merchant, COTS and SCRP components that took place in its processing.

5.4.2 Secure Channels

A secure channel that provides mutual authentication, so that each component participating in an exchange of sensitive data can be identified, must exist between each of the disparate components. Keys used to encrypt data going through a secure channel must not be the same keys used for data encryption.

5.4.3 PIN CVM Solution Requirements

The solution is allowed to initiate PIN entry sessions only when online (connected to the back-end monitoring and attestation systems) and authorized

[4] Cardholder Data Environment – Processes and people that store or process sensitive cardholder data.

by the monitoring environment.

All transactions performed using the solution must be analyzed for anomalous and potentially fraudulent activity, either by local or remote detection systems.

When a solution provider submits the solution to the PCI-approved lab for certification, it is also required that the lab is provided with a test environment that the solution runs in with full accessibility and visibility so that it can be evaluated in-depth. A solution provider must also have a risk-assessment policy as well as a threat-management process document, which is reviewed at least annually.

The standard further requires that the injection of all encryption keys is done by a key-injection facility compliant with PCI PIN Security Requirements.

5.5 Back-end Systems – Processing

Since the back-end system is the only component of the solution that is allowed to perform decryption of cardholder data and PIN data, it must adhere to the PCI PIN Security Requirements as well as to PCI DDSS DEV requirements.

5.6 Secure Card Reader (SCRP)

The standard allows only PCI-approved SCRP devices to be used in the solution for reading chip cards of customers.

6 Contactless Payments on COTS

This chapter covers the **CPoC** (Contactless Payments on COTS) standard [36], where contactless-enabled cards and devices such as smartwatches and mobile phones can be accepted using COTS devices with embedded NFC capabilities to authorize contactless card payments, making card acceptance easier and more available for merchants. All hardware that is required to accept cards is a COTS device with NFC capabilities running a supported version of an operating system and the associated CPoC application connected to back-end processing systems (see fig. 6.1).

This solution further reduces the initial cost when a merchant decides that he would like to accept card payments as it is very likely that a merchant already owns a supported COTS device. As no special hardware such as an SCRP or an external pin-pad is required, the process of merchant onboarding is typically faster in comparison to onboarding with a dedicated card terminal.

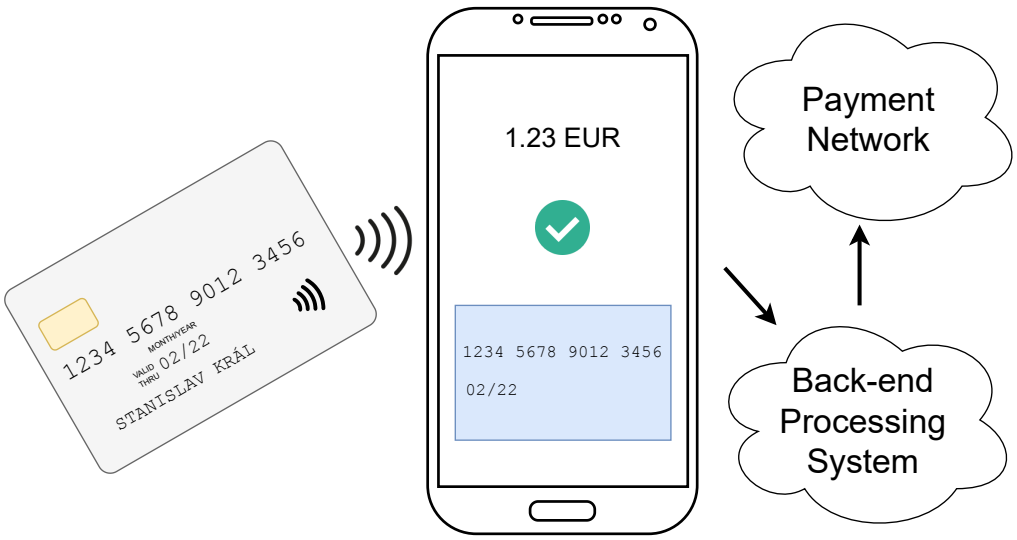


Figure 6.1: A diagram visualizing card acceptance using a CPoC solution.

Cardholder data is wirelessly transmitted to the COTS device using the device's embedded NFC antennas, where it is initially processed by the CPoC application that periodically undergoes security attestations. After that, it is sent to the back-end systems of the solution, where it is eventually forwarded

to the payment network. The authorization result is later received by the COTS device and displayed to the merchant by the CPoC application.

For example, this solution may prove useful for traveling merchants as they won't have to carry any additional equipment during their sales. Additionally, the quick setup and minimal hardware requirements make it suitable for temporary points of sale (such as payment on delivery) or philanthropy events where it was previously costly and inefficient to accept card payments. Merchants can accept contactless payments using the Android device they already own.

A CPoC application residing on the COTS device provides a channel to the embedded NFC interface to be able to initiate the reading of a contactless payment instrument. Cardholder data read by the NFC controller using a contactless kernel are encrypted and delivered through a secure channel to the back-end processing environment to be decrypted so that it can be passed for subsequent transaction processing. Furthermore, the application periodically passes attestation health-check data about the platform to the back-end monitoring system and using implemented software protection mechanisms protects the solution against attacks. The architecture of a CPoC solution is visualized in Figure 6.2.

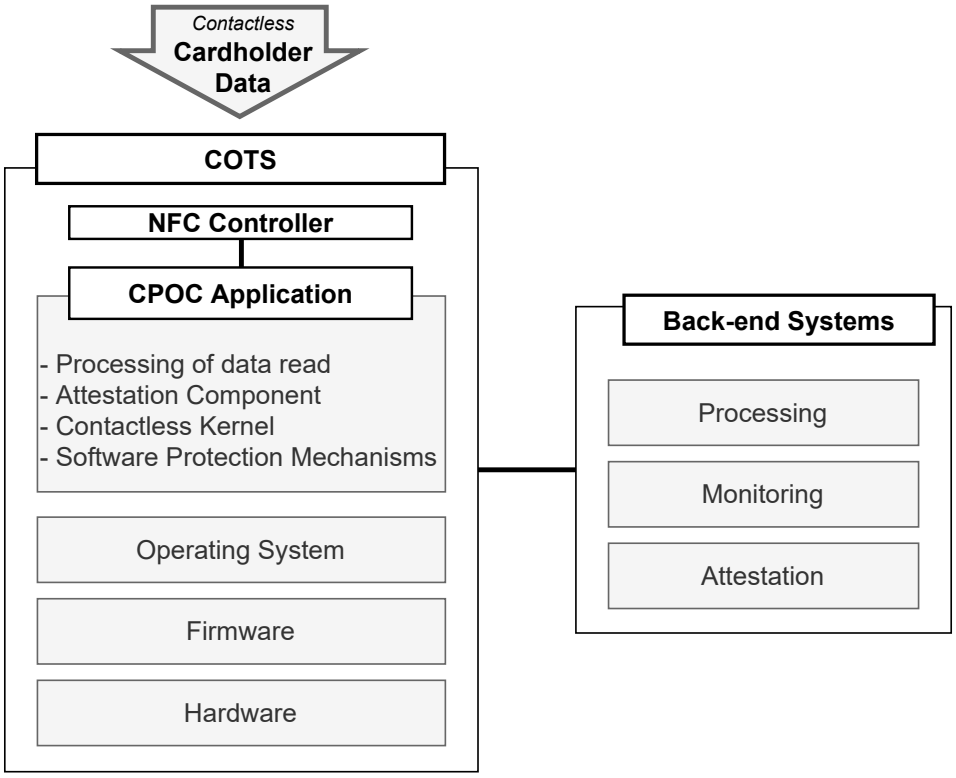


Figure 6.2: A diagram visualizing the architecture of a CPoC solution.

The commercial-off-the-shelf device may be any device with NFC capabilities that runs a supported up-to-date operating system. This may include various Android devices, such as smartphones or tablets.

A set of back-end systems is responsible for the periodical evaluation of the solution's integrity which is done by providing monitoring and attestation functions. It is also responsible for forwarding the transaction and cardholder data received from the CPoC application to the payment network for authorization.

The flow of the processing of a transaction in a CPoC solution is described in the following steps:

1. The CPoC application is downloaded from the OS application store.
2. A secure communication channel between the CPoC application and back-end systems is established.
3. The security status of both the COTS platform and the CPoC application is determined by the attestation component.
4. Merchant data and cryptographic keys are used to initialize the CPoC application.
5. Device's proximity is probed for contactless payment instruments using a contactless kernel after the merchant initiates the payment transaction by setting the amount for authorization.
6. A contactless payment instrument is presented to the COTS device and required data is acquired from it.
7. The CPoC application encrypts the transaction data and sends it to back-end systems for subsequent processing.

The CPoC standard does not cover the printing of a transaction receipt, as it is not expected that COTS devices are equipped with a receipt printer. Merchants may utilize external printers or paperless receipts that can be sent through an SMS or email.

6.1 Security Requirements

It is not surprising that most of the security requirements specified by the standard that the solution must meet are similar to those that are defined for SPoC solutions as both solution types operate with sensitive information related to financial operations.

Similarly to the SPoC standard security and test requirements are divided into 5 modules:

- **Core Requirements** – defines base security rules for cryptography and key management, random number generations, privileged access to sensitive data and secure services, software development and operational management.
- **CPoC Application** – contains requirements for tamper-resistance and code obfuscation methods implemented in the CPoC application, transaction processing, encryption of cardholder data and maintaining audit logs.
- **Back-end Systems – Monitoring/Attestation** – describes required properties of the monitoring and attestation components as well as types of attestations to be performed.
- **Back-end Systems – Processing** – states that the decryption of all cardholder account data must occur only in back-end payment processing environments that maintain and comply with PCI DSS requirements.
- **Contactless Kernel** – defines contactless kernel requirements such as that the solution must use implementations of kernels approved by payment brands and complying with PCI DSS requirements.

During attestation the security of the COTS platform is evaluated and at a minimum, the following must be reported:

- whether the device is rooted or operating in developer mode,
- asynchronous rooting and unrooting of the COTS OS has been detected,
- platform's support for secure compilation and application execution,
- modifications or tampering of the platform's OS,
- rollback of the platform's OS or the CPoC application,
- NFC interface access logs if available.

If any of the following events are detected during transaction processing the payment session must be immediately terminated and all captured data deleted:

- tamper-detection event is signaled,
- it is detected that the application is running in a developer or emulator mode,
- the application loses foreground, focus, pauses or stops executing.

When reading cardholder data from the contactless payment instrument the CPoC must attempt to lock the NFC interface so that it cannot be used by other applications residing on the platform. Not doing so might allow other applications to monitor and sniff data exchanged between the customer's payment instrument and the CPoC application. A solution provider should carefully examine the API of the platform's OS to find suitable locking options. For example, the OS may only allow applications in the foreground to access the NFC interface. The CPoC application can utilize that restriction by reacting to the loss of focus and aborting any transaction processing once it is removed from the foreground, effectively locking the NFC interface for itself.

Similar to the locking of the device's NFC interface, the application must also attempt to lock the device camera during the processing of a transaction as the camera may be misused by potential attackers to visual capturing of cardholder data when it is in proximity to the COTS device. A solution provider should examine the possibilities of locking the camera in the API of the platform's OS, and if such locking is unavailable, it should at least prompt the user to disable the camera manually and not allow to initiate a transaction until the camera is disabled.

6.2 Limitations

Since the CPoC application may run on a wide variety of uncertified devices to which users may have full privileged access and weren't initially designed to accept payments, the standard prohibits offline authorizations as well as the PIN CVM method as the application may be exposed to attacks from malicious software residing on the device. This means, due to security challenges and complexities of protecting cardholder data in an uncertified environment, traditional contactless transactions that are over the contactless amount limit cannot be performed as they require the cardholder to enter his PIN. Furthermore, the COTS device must be connected to the internet during the processing of a payment transaction as all transactions are required to be authorized online.

The prohibition of PIN CVM methods is one of the biggest disadvantages of CPoC solutions, as it forbids a particular group of transactions. However, as the popularity of card tokenization is increasing [37], cardholders are more often being verified using the CDCVM method, which does not require a PIN to be entered on the merchant's payment acceptance device, but uses the customer's mobile phone to enter the PIN or completely avoids it by utilizing biometric authentication.

To allow both PIN entry and the reading of cardholder data on COTS devices, it is speculated that the PCI Security Standards Council will in the near future issue a new standard called CPoC + PIN [38] that will further increase the security requirements put on the solutions. Various technologies such as **TEE** (Trusted Execution Environment) and **TUI** (Trusted User Interface) that help implement a secure environment on the COTS device to capture cardholder PIN are emerging and will probably be used to meet the requirements of the new standard.

It should be noted that some card schemes such as Visa or Mastercard have published contactless kernels and guidelines that allow PIN transactions using a combination of CPoC and SPoC solutions [39, 40].

6.3 Contactless Kernels

Both Visa and Mastercard offer their own contactless kernels that can be used in CPoC solutions. These kernels also come in the form of Java libraries that can be used in Android applications.

6.3.1 Visa

Visa allows members of the Visa Ready program to use the Visa Tap To Phone SDK, which can be used in Android applications to accept contactless Visa cards. This SDK contains the Visa Contactless Kernel as well as other functions that support transaction processing on Android COTS devices.

Additionally, solutions created with the Visa SDK that are compliant with the Visa PIN Entry guidelines may support the Online PIN CVM method. This means that when using this solution cardholders can present their cards to a COTS device and be prompted to enter their PIN on the same COTS device, as opposed to solutions built on the base CPoC standard, where transactions requiring the Online PIN CVM method are prohibited.

6.3.2 Mastercard

During the year 2020 Mastercard has developed and released their Tap on Phone pilot SDK that contains a selection module and a kernel compliant with Mastercard contactless specifications, allowing compatible Android devices to read contactless Mastercard cards [40]. Tap on Phone with PIN pilots can perform PIN CVM methods and compliance with PCI CPoC is not required as compliance with Mastercard Security Principles is sufficient. As of 2022, this SDK is free of charge and is available after email communication with Mastercard.

6.4 Successful Card Read Rate

When trying to tap the card to a COTS device, one may observe that it requires more time and precision to correctly place the card to the NFC reader of the device to successfully present it. Additionally, the orientation of the card when tapping it affects the rate of successful presentations [41]. The process of reading a card is certainly not as smooth as when using traditional card terminals. However, this is to be expected as COTS devices were not designed for card acceptance as opposed to card terminals that have their NFC antennas optimized in a such way, that the successful card read rate is very high.

6.5 Existing Implementations

6.5.1 Android OS Platform

As of Q1 2022, there are several applications on the Google Play application store that enable Android devices with NFC capabilities to accept payment cards. Example of such applications include UniCredit Bank SoftPOS[1], Revo SoftPOS[2] and SoftPOS[3].

These applications require that a merchant undergoes a registration process and signs a contract with an acquiring bank before they can be used to authorize transactions.

[1] `https://play.google.com/store/apps/details?id=com.provisionpay.softpos.unicredit`

[2] `https://play.google.com/store/apps/details?id=app.openmpos.eservice.RevoSoftPOS`

[3] `https://play.google.com/store/apps/details?id=com.mst.retail`

Based on its description in the application store, the Revo SoftPOS application claims that it supports the PIN CVM. None of the other mentioned applications do not explicitly claim that this feature is supported.

However, when taking the number of downloads and reviews of the mentioned applications into consideration, it seems like these solutions are not widely used. Negative reviews also suggest that users of these solutions are not entirely satisfied with them. In summary, some CPoC implementations do exist, their quality is uncertain and they aren't frequently adopted by merchants at this moment.

6.5.2 iOS Platform

At this moment, there are no solutions available that provide support of processing payment card transactions on iOS. This is caused by the fact that the iOS SDK currently forbids communication with payment-related card applications [42].

In February 2022 Apple has announced that later in 2022 US merchants will be able to accept payment cards on iPhones using partner-enabled iOS apps[4]. This new feature will be called Tap to Pay and will be available to application developers in the SDK of the upcoming iOS versions running on iPhone XS or later devices. This means that in the next years we can expect various iOS applications that will enable iPhones to be used as card terminals.

It should be noted that the referenced Apple press release does not mention whether the PIN CVM method will be supported.

[4] `https://www.apple.com/newsroom/2022/02/apple-unveils-contactless-payments-via-tap-to-pay-on-iphone/`

7 Extending the Dotypay Application

The main goal of the implementation part of this master's thesis is to verify that COTS devices running the Android OS can be used to accept contactless payment cards by extending the Dotypay[1] application with support for this group of devices.

Dotypay is a product developed by Smart software s.r.o.[2] that allows merchants to interface with payment cards to make electronic funds transfers via a payment terminal it offers. It is currently available in the Czech Republic and Slovakia with plans to expand to Slovenia and Hungary in the near future.

The terminal is a mobile device running an Android OS with a built-in chip and contactless card reader with the support of magnetic stripe cards. The fact that the terminal is running an Android OS means that it allows for the utilization of various cash register applications that can be installed directly on the terminal. This reduces the number of devices the merchant has to operate in order to process customer orders to just a single all-in-one device.

The terminal comes with two preinstalled applications: Dotypay and Dotypay Launcher. The former is used for creating and processing payment, preauthorization and return transactions with the ability to preview the transaction and settlement history. The latter is used for device configuration which consists of the following tasks:

- **cryptographic keys setup** – in order to securely process card payment transactions, the terminal must be properly configured in the means of having all required cryptographic keys securely stored in the device storage,
- **merchant personalization** – configuration of the terminal based on the merchant it belongs to (e.g., terminal ID, merchant ID, currency code, country code, etc.),
- and **maintenance of the software installed** – periodically check whether there are updates available to the installed software such as

[1] `https://dotypay.com/`

[2] `https://www.smart-software.cz/`

payment application, terminal firmware or cash register application.

Configuration and management of terminals happen on a dedicated web application, where merchants can also access transaction history, analytics and other useful insights of made turnovers.

After initial validation on the terminal, card transactions are routed to the systems of the Monet+ acquirer[3] for authorization.

7.1 Application Specification

The Dotypay application is an application for the Android OS platform and is written mostly in the Kotlin language with some of the legacy code written in Java. It was designed in a such way that it can only be run on the Landi A8 device[4] and supports all basic features one would expect from a fully-fledged card terminal, including standard payment transactions, refunds and pre-authorizations. The Landi A8 device is running version 5.1 of the Android OS. The application's dependency on the Landi A8 device is the main issue there is with the application, as it makes use of other devices impossible. It is also expected that the device vendor will soon no longer support the device with any updates.

The application uses a local SQLite[5] database to store the history of performed transactions and heavily relies on the reactive library RxJava[6] to control the application flow. Using database triggers, the local transaction history of each terminal is synchronized to the back-end systems by making HTTP requests to the REST API interface of the back-end.

Additionally, to allow integration with POS systems of a wide variety, the application exposes a REST API that can be used to initialize payment transactions without having to manually enter the transaction amount or currency in the application itself. The exposed REST API implements the Nexo protocol[7] used in the card payment industry to unify the card terminal interface.

It should be noted that the application does not allow offline authorizations. This means that every card transaction must be sent to the Monet+ acquirer. Before communication with Monet+ is possible, the device must be properly loaded with encryption keys and terminal configuration (such as terminal ID and merchant ID) to allow for secure communication with it.

[3] `https://www.monetplus.cz/emv-payments`
[4] `http://www.landicorp.com/en/product3_312.html`
[5] `https://sqlite.org/index.html`
[6] `https://github.com/ReactiveX/RxJava`
[7] `https://www.nexo-standards.org/standards/nexo-retailer-protocol`

7.2 Application User Interface

The user interface is rather simple and provides quick navigation using a subtle color palette to distinguish between the main UI elements of the application (see figures 7.1 and 7.2).

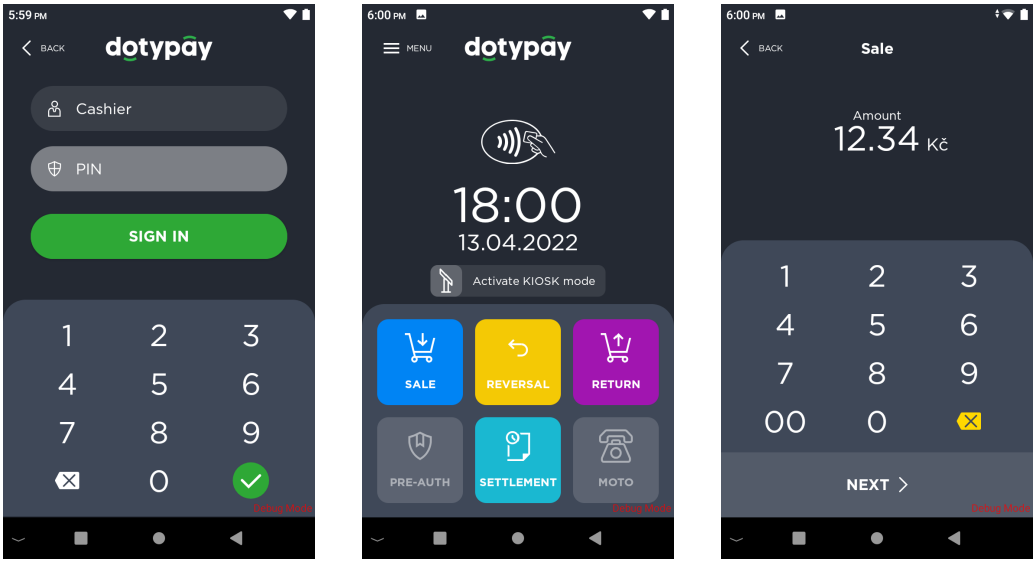


Figure 7.1: Screenshots of the Dotypay application screens during user sign up, action selection and transaction amount entry.

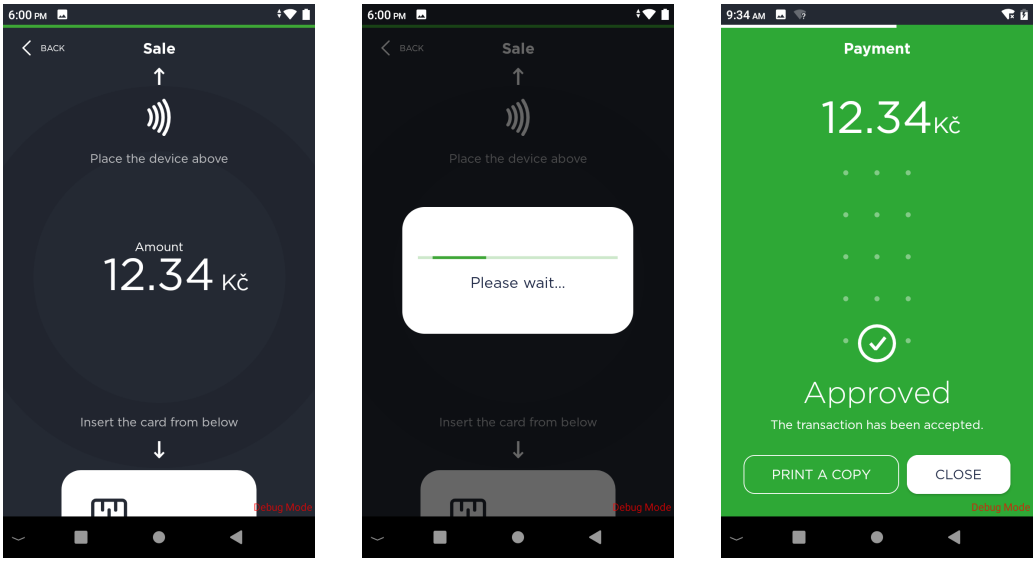


Figure 7.2: Screenshots of the Dotypay application screens during card presentation and after a transaction authorization is finished.

Transitions between the screens of the application are visualized in Figure 7.3. Some of the application screens that are not directly linked to the processing of payment transactions are omitted there.

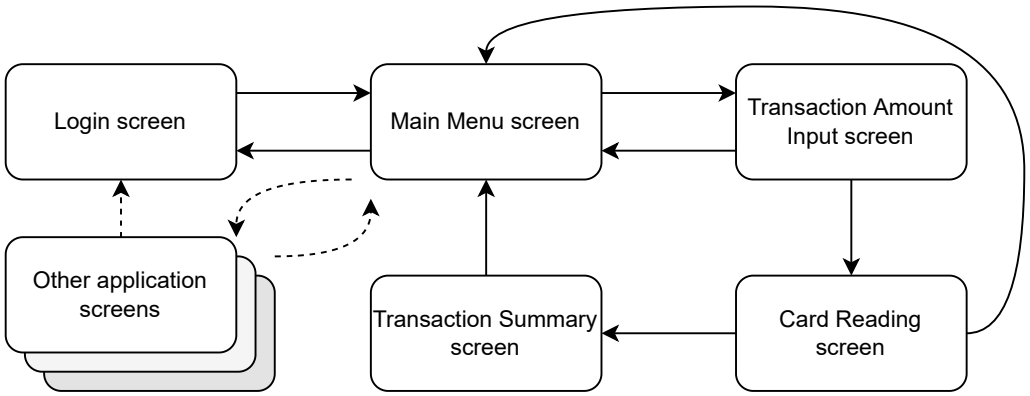


Figure 7.3: A diagram visualizing the transitions between screens of the application.

If enabled in the device's configuration, the application may prompt the merchant for a variable symbol or the customer to specify a tip that will be added to the transaction amount. Transaction parameters, such as the amount or variable symbol, can be automatically set by scanning a payment QR code.

Additionally, the device's configuration allows putting the terminal into a kiosk mode in which the application's main menu is hidden and transactions are most often initiated remotely (typically by a POS system).

7.3 Cryptographic Key Infrastructure and Exchange

To create a secure environment for card transaction processing, it is required that all components of the solution (card terminal, back-end processing systems, the acquirer, etc.) have established a secure communication channel between each other.

The following text is solely based on the integration with the Monet+ acquirer that is used in the Dotypay product to authorize card payments, however, it is expected that other acquirers and card terminal manufacturers follow a similar infrastructure.

To allow secure cryptographic key exchange card terminal manufacturers sell in addition to card terminals a specialized device called **KLD** (Key

Loading Device), that is used to safely inject symmetric cryptographic keys into terminals. The KLD device itself is injected with a master key and the premise is, that injected keys cannot be extracted from either the KLD or terminals. See Figure 7.4 for an example of a KLD device.

Figure 7.4: A KLD device from the Nexgo manufacturer.

An acquirer keeps the KLD master key secret and on-demand generates a list of **TTK** (Terminal Transport Key) keys that are associated with provided serial numbers of terminals to be introduced to the payment network as requested by a solution provider. Both the KLD and the list of enciphered TTK keys are then delivered to the solution provider, who then uses the KLD to inject the received keys into its terminals. During the dual-control process of injecting keys, the KLD decrypts the enciphered keys using its master key. After that, the terminal is prepared to be shipped to merchants.

Eventually, during merchant personalization of the terminal, additional cryptographic keys such as **TMK** (Terminal Master Key) and **TEK** (Terminal Encryption Key) must be injected into the terminal. These keys are delivered enciphered by the TTK to the terminal over HTTPS and must be decrypted before being stored in the device's storage.

The previous text is mostly based on the process of key exchange with the Monet+ acquirer that the Dotypay solution uses to forward authorization requests to card schemes. The key exchange itself is initiated by a separate application on the terminal called Dotypay Launcher that uploads required credentials to an **SFTP** (Secure File Transfer Protocol) server operated by Monet+. In response to the key exchange request, the acquirer uploads enciphered keys to the SFTP server that the Dotypay Launcher application downloads and loads into the card terminal using the device's SDK API. After that, the card terminal is ready to begin communication encrypted by acquired keys that is considered secure by both parties.

7.4 Application Architecture

The development of a card transaction processing solution involves an integration of many processes and services. Not only does the solution as a whole must be able to participate in an EMV dialogue, but it also must send transaction data to an acquirer for online transaction authorization. In addition to that, it is also expected that the solution allows merchants to view the transaction history in the card terminal application itself and also to view more sophisticated reports of sales in some form of a web interface. The card terminal application should also be able to initiate a card transaction after receiving a request originating from an authorized remote device located in a local or remote network.

From a business perspective of a solution provider, one of many crucial requirements is that the designed architecture of the application residing on the card terminal must be relatively easily portable to other terminals that it was not originally designed for. The probability of the risk that a need to change terminal vendors arises, be it due to an end of device support or due to changes to a vendor's pricing policy, is not negligible, but rather high, and its consequences are very severe if not handled properly.

Another component of the solution that can eventually change during its lifetime is the dependency on the acquirer. This may involve changes made to the interface exposed by the acquirer or integration of an interface of a new and previously unknown acquirer. New acquirers may also use different techniques of cryptographic key exchange.

The Dotypay card terminal application was initially developed specifically only for the Landi A8 device, which offers chip, contactless and magnetic stripe interfaces as well as a receipt printer. Because rapid progress on the product was required by the business during its initial development, some

aspects of the application's architecture weren't properly thought out, which resulted in dependencies on the Landi A8 device interface being scattered everywhere around the application's code, making it almost impossible to implement support for any other card terminals.

It was required to design a new architecture that would allow separating the device-specific code from other application code, making it easy to replace the dependency on the underlying device interface (see fig. 7.5).

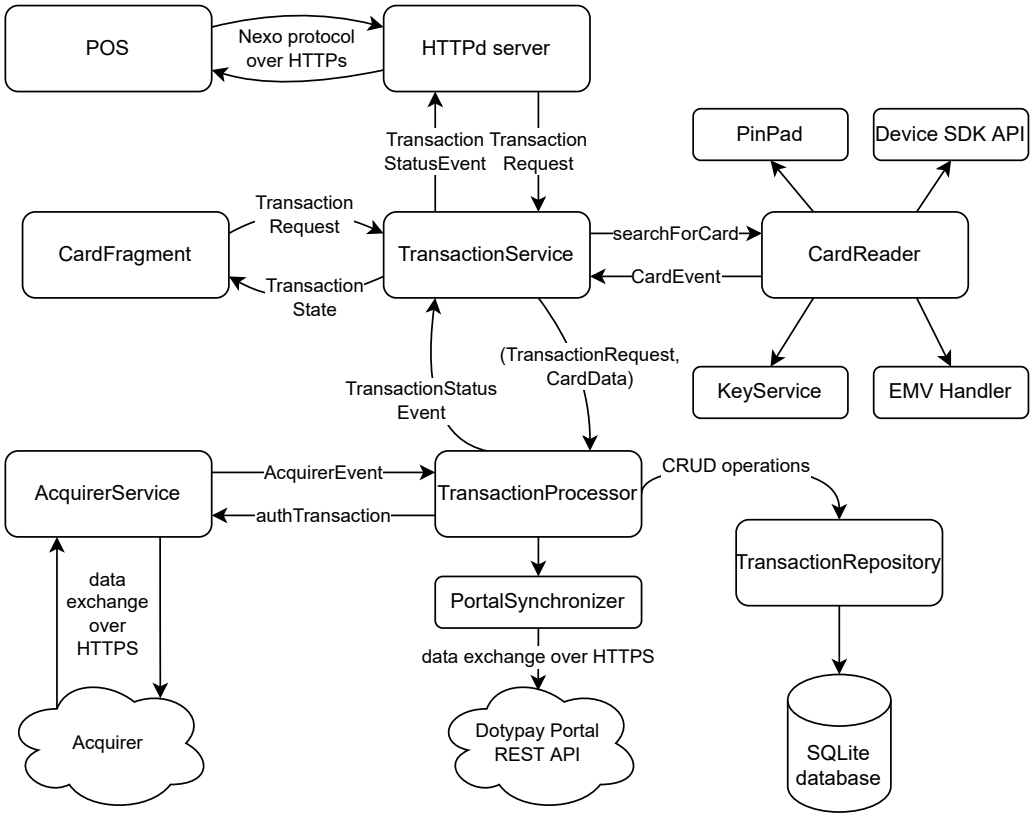


Figure 7.5: The newly designed architecture of the Dotypay application.

Aside from the promise of easier application maintenance, the main reason why it was necessary to design a new application architecture was the business requirement of being able to operate the Dotypay application on other devices (see fig. 7.6), including COTS devices and devices from the Nexgo manufacturer (specifically the Nexgo N86 device[8]).

[8] `https://www.nexgoglobal.com/smart-pos/n86-smart-pos-terminal.html`

Figure 7.6: The updated application running on Nexgo N86 and Google Pixel 6 devices.

7.4.1 Refactoring of the Application

Right after the new architecture has been designed, the author of this master's thesis presented it to the team behind the Dotypay application. After making some minor tweaks to the architecture the team started to refactor the application to use the new architecture. As a lot of Landi A8 device-specific code was scattered around the whole application codebase, this was a particularly challenging task. Before the planned support for devices from other manufacturers, such as Nexgo, Sunmi[9] or Kozen[10], the Dotypay team struggled whenever a new feature was requested or a new bug was found because the application's codebase was hard to navigate around and even small changes could potentially break other (presumably unrelated) features of the application. The new architecture also aimed to solve this issue as it has been drastically slowing down the development of the application. Additionally, the newly achieved modularization achieved by putting emphasis on separation of concerns, from which the application codebase benefited, made covering some parts of the application's code with tests much easier,

[9] `https://www.sunmi.com/en/`

[10] `https://www.kozenenterprise.com`

and unit tests targeted at critical parts of the application were created. Once the refactoring was done, the application's codebase was ready to be extended with the support for other devices, which meant implementing interfaces hidden behind the **CardReader** architecture component.

7.4.2 Transaction Service Component

Initiation of a payment transaction happens in the Transaction Service, where based on the input transaction request a new session is created and sensors of the card terminal are activated to probe for customer payment instruments, and when found, the card data alongside the transaction data is sent to the Transaction Processor component where it is further processed. Activation of card terminal sensors is delegated to the Card Reader component.

Additionally, the author of the Transaction Request is notified several times during the processing of the transaction with the latest transaction state. This allows displaying the current status of the transaction on the device's screen. In the Dotypay application, the instances of the `CardViewModel` class observe the status of the ongoing transaction and display it to the user.

7.4.3 Card Reader Component

The responsibility of the Card Reader component is to facilitate the communication between the terminal and the presented payment card. This involves the usage of the device's SDK to search for cards and to gain access to the device's encryption capabilities, which are for example used to encipher the entered PIN during cardholder verification or encryption of the communication with the acquirer.

Searching for a card is initiated by calling the `searchForCard` method, which returns a stream of events that are broadcasted whenever a card is interacted with. Events are also broadcasted to notify whenever an error during a communication with a card has occurred or when user interaction is required. One of the scenarios that require user interaction is when the presented card contains multiple applications and the cardholder is asked to select which application should be used for further processing. Some of the most important events broadcasted by this component include:

- `ReadCardStarted` – informs that probing for cards has been started,

- `SelectAid` – requests user interaction to select a card application from a candidate list to be used for further transaction processing,
- `OnlineRequest` – indicates that the transaction requires online authorization,
- `KernelVerificationSuccessful` – indicates that the response from the acquirer has been sucessfully verified,
- `WrongPin` – notifies that an invalid PIN has been entered.

Device SDK

The Device SDK component provides an interface to other device-specific features such as illumination of LED indicators or activation of the speaker to provide sound feedback. The device's capabilities to read cards are exposed by this component.

PIN Pad

The functionality of displaying a PIN pad to make entry of a cardholder's PIN possible is provided by the PIN Pad component. The layout of the PIN pad's keys can be specified as well as whether the PIN is being entered for online or offline verification. In the case of an online verification, the PIN is enciphered into a PIN block and is sent to the acquirer for verification.

The PIN Pad component also provides methods that allow using the secret keys stored in the device's storage to encrypt and decrypt sensitive data.

EMV Handler

The EMV Handler component provides an interface to communicate with a card using the EMV protocol. It is used when the card is being presented to the card reader using one of the available physical interfaces. Data exchanged are encoded using the TLV encoding.

7.4.4 Transaction Processor Component

Once a card has been successfully read and all required data have been gathered from it, the Transaction Processor receives this data and additional processing begins. Before being sent to the acquirer for authorization the transaction data is stored in the local database using the Transaction Repository component. After a response from the acquirer is received the

transaction status in the database entry is updated accordingly and the authorization result is further propagated to the Transaction Service component.

7.4.5 Acquirer Service Component

Communication with the acquirer that facilitates the incoming payment card transactions is encapsulated in the Acquirer Service component so that if a new acquirer's services must be integrated, the scope of changes required to be made in the solution is minimized. This component uses the features of the PIN Pad component to securely encrypt the communication between the terminal and the acquirer using the secret keys stored in the device's storage.

7.5 Modularization of the Application

To add support for other devices to the Dotypay application, it was required to create a device-specific implementation of most of the interfaces based on the architecture as described in the section 7.4. Additionally, all these modules and components associated with device-specific implementation have to be properly initialized during the application startup.

To be able to efficiently initialize and connect created modules and components of the designed architecture in the Dotypay application, a dependency injection container had to be introduced to the application. This involved choosing the right dependency injection library available for Android applications. The two most used dependency injection libraries in Android application development are the Dagger 2 [43] and Koin [44] libraries. Although the Koin library uses more readable constructions, offers native Kotlin API and does not negatively impact the compile time of the application, the Dagger 2 library, after a research, was selected, as it validates all configuration preconditions at compile time and uses code-generation to create component providers, meaning that almost all issues there might be with the dependency injection container, such as missing or circular dependencies, are discovered during compile time, rather than during runtime. This eliminates a certain domain of unnecessary runtime crashes, which is important for an application related to finances. The Koin library does not use any static validation or code generation and thus does not provide this benefit.

7.6 Implementation for Nexgo Devices

Adding support for Nexgo devices to the Dotypay application meant using the Nexgo Device SDK API to create an implementation of architecture's interfaces related to payment card discovery and interactions with payment cards.

The core of the implementation lies in the `NexgoCardReader` class from the `com.stpos.a8pos.devicesdk.impl.nexgo` package. Its most important method is the `searchForCard` method, which returns a stream of `CardEvent` objects that have to be sent in the right order so that the transaction is correctly processed. To even better modularize the code, following classes that are imported in the `NexgoCardReader` class were created:

- `NexgoEmvInitializer` – initializes the Nexgo EMV handler by setting the list of supported AIDs and issuer certificates,
- `NexgoEmvHelper` – contains code that is responsible for controlling the flow of an EMV transaction (e.g., prompting for PIN or gathering cardholder data)
- `NexgoCardReaderHelperImpl` – provides helper methods for discovering cards entering one of the device's interfaces.

As most of the EMV communication implementation is provided by the Nexgo Device SDK API, the most difficult aspect of the implementation was understanding how to use the API, as its documentation was quite hard to work with because it is not descriptive enough with some of the functionality of the API lacking important information, such as documentation of method parameters. This is particularly true for sections that describe how to load encrypted keys into the device's storage, where it is not clear at all what meaning the parameters of some methods have.

As a result of this, a great part of the implementation the author of this master's thesis had to experiment with the parameters of methods related to loading of encrypted keys. Before the loading of cryptographic keys was correctly implemented, it was impossible to communicate with the Monet+ acquirer.

7.6.1 Working With Cryptographic Keys

Being able to properly load the encrypted cryptographic keys into the device meant utilizing the `PinPad` class from the Nexgo SDK library to implement the **PinPad** component. The SDK separates keys into two classes: master

and work keys, but it does not specify based on what criteria keys should be categorized. However, as opposed to work keys, master keys cannot be directly used for the decryption of data as the SDK does not provide any methods that would accept data encrypted by a master key and return the data in a decrypted form. The decryption of encrypted data can only be achieved indirectly when loading other master keys that are encrypted by another master key. The decrypted form of master keys cannot be retrieved in any way.

During implementation, it has been decided that in the **PinPad** component implementation work keys are such keys that are specific to each payment transaction and change often. For example, this includes keys used to encrypt the PIN block before sending it to the acquirer for validation. On the other hand, master keys are set up only during terminal initialization or during merchant personalization and are used only when loading other master keys. Encrypted keys are loaded into the device by using the `writeMKey` (for master keys) and `writeWKey` (for work keys) methods of the `PinPad` class from the Nexgo SDK. In these methods the encrypted key to be loaded, its length and the key to be used for decryption of the key must be specified.

Aside from the Dotypay Application, the Dotypay Launcher application also had to be updated to call the `writeMKey` method from the NexgoSDK when master keys are downloaded from the acquirer during key injection.

7.6.2 CardReader Interface Implementation

The **CardReader** component of the application's architecture is responsible for discovering and interacting with payment cards. The core of the interface's implementation is done by using `CardReader` and `EmvHandler` classes from the Nexgo SDK.

At the start of each transaction, the list of supported AIDs and issuer certificates has to be passed to the EMV module of the Nexgo SDK. This was achieved by mapping the list of supported AIDs and issuer certificates already present in the Dotypay application to the structure required by the Nexgo SDK API.

This allows invoking the `searchCard` method of the Nexgo SDK `CardReader` class that begins to probe the device's interfaces for payment cards. Once a card is found, the EMV process is started using the `startEmvProcess` method using the `EmvHandler` class and a listener that reacts to certain EMV actions is passed.

When required, the device's software PIN pad is shown (see fig. 7.7).

This PIN pad overlays the Dotypay Application and is controlled by the Nexgo SDK itself. Its layout is randomized each time this PIN pad is displayed. Once the cardholder has finished entering his PIN the PIN pad is dismissed.

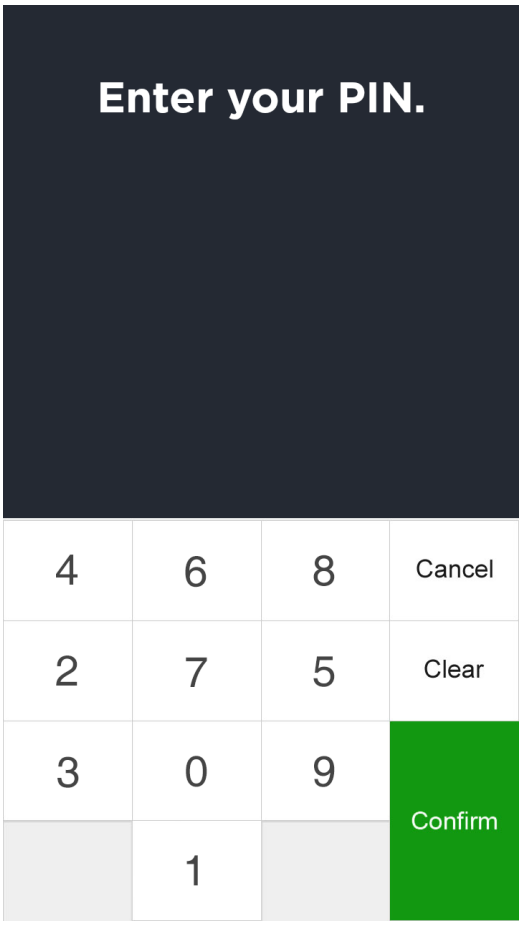


Figure 7.7: The software PIN pad displayed on Nexgo devices during PIN CVMs.

After the initial EMV communication is finished, all fields required by the acquirer, such as the values of Application Cryptogram or Unpredictable Number TLV tags, are gathered and sent to the output stream wrapped inside an instance of the `OnlineRequest` class. This data is then eventually forwarded from the **TransactionService** component to the **AcquirerService** component to create and submit a transaction authorization request to the acquirer. Later, using a callback, the response from the acquirer is verified by the EMV handler using the `onSetOnlineProcResponse` method of the `EmvHandler` class, and the EMV process is finished.

It should be noted that as opposed to the Landi A8 Card Reader imple-

mentation the Nexgo implementation does not currently support magnetic stripe transactions, as the focus of this master's thesis was primarily on EMV transactions which magnetic stripe transactions aren't. Furthermore, printing of receipts on Nexgo devices is not implemented yet in the current version of the application. Both the presentation of cards using a magnetic stripe and ability to print receipts will have to be added to the application in order for it to be ready to be used in the production environment.

7.7 Implementation for COTS Devices

Visa and Mastercard card schemes both offer a contactless kernel that can be used on mobile devices in combination with the device's NFC capabilities to accept card payments [39, 40]. These kernels are available to card terminal application developers for free after applying for the brand's partnership program and being accepted.

The Smart software s.r.o. company has already been accepted to the Visa Ready program and has access to the pilot Visa Tap to Phone SDK that includes a contactless kernel for accepting Visa brand cards.

Unfortunately, this master's thesis author's application for the Mastercard MPOS Partner program that contains the mobile contactless Mastercard kernel has been declined by Mastercard due to the reason that they do not offer the SDK for educational purposes. The Smart software s.r.o. company has not applied for this program yet and thus does not have access to it. As a result of that, the Dotypay application, when running on COTS devices, only supports Visa cards because it does not contain the Mastercard contactless kernel.

7.7.1 CardReader Interface Implementation Using Visa Tap to Phone SDK

To implement the `CardReader` interface for COTS devices it was required to use the Visa Tap to Phone SDK[11] (referred to as Visa TTP SDK in the following text) in combination with the device's NFC capabilities. This includes searching for NFC tags and initiating a communication with a found NFC tag using the transmission protocol specified in the ISO 14443-4 standard [45]. The implementation of the transmission protocol is available in the Android SDK by using the `android.nfc.tech.IsoDep` class [46] that works with instances of the `android.nfc.Tag` class [47].

[11] Requires Android API Level 26 (Android 8.0).

An example of reading and working with NFC tags is shown in Code Sample 7.1.

```
1  import android.nfc.NfcAdapter.*
2
3  // specify reader flags to define supported tag types
4  private const val nfcReaderFlags = FLAG_READER_NFC_A or
       FLAG_READER_NFC_B or FLAG_READER_NFC_F or
       FLAG_READER_NFC_V or FLAG_READER_NFC_BARCODE
5
6  // get the NFC adapter using the Android activity
7  val nfcAdapter = NfcAdapter.getDefaultAdapter(activity)
8  // start searching for NFC tags
9  nfcAdapter.enableReaderMode(
10     activity,
11     { tag ->
12         // process the found tag
13         val isoDep = IsoDep.get(tag)
14         isoDep.connect()
15         // send the command to select a proximity payment
               system environment
16         val receivedData = isoDep.transceive(SELECT_PPSE)
17         // process the received data
18         processPpseResult(receivedData)
19     },
20     nfcReaderFlags,
21     null // do not pass any extras
22 )
```

Code Sample 7.1: Example of Kotlin code used to search for NFC tags and exchange data with found ones.

When a tag is found, it is required to check whether it contains a supported application. As this functionality is not specific to Visa cards only, it is not part of the Visa TTP SDK, however, the `PPSEManager` class, which is part of the example application that comes with the Visa Tap to Phone SDK, supports the discovery of card's PPSE (described in 3.5.2) and the following application selection. The class does so by querying the NFC tag for a PPSE and when a PPSE is found, it is validated and the list of candidate applications is assembled. The first application with the highest priority in the list is selected and an appropriate contactless kernel is used. Because only a Visa contactless kernel is available at this moment, the user is asked to present another card when a non-Visa AID is selected.

If a VISA application is selected, initial terminal TLV tags based on the transaction and merchant data are generated. These tags are stored into an instance of the `ContactlessConfiguration` class that is used as one of

the parameters passed when calling the `performTransaction` method from the `ContactlessKernel` to initiate the kernel transaction (both mentioned classes are part of the Visa TTP SDK).

Online PIN CVM implementation

To properly set the CVM limit (described in 2.4) one must add the proprietary tag `DF01` from the Visa TTP kernel namespace to the list of TLV tags passed to the kernel, otherwise, transactions above the CVM limit won't require a CVM, in this case, the Online PIN CVM.

As opposed to the Landi A8 SDK or Nexgo SDK the Visa TTP SDK does not offer any software-based PIN pad to be displayed when the cardholder should be prompted for a PIN. This means that any application that integrates the Visa TTP SDK must implement its own software-based PIN pad if PIN-based CVMs are to be supported. In the Dotypay application, this was implemented by utilizing the already available `SimpleInputDialog` class, which is used to display a modal dialog prompting for a value to be entered (see fig. 7.8).

During Online PIN CVM this dialog is set to accept only numeric values and display a custom keyboard (PIN pad). Entered digits are masked and replaced with the asterisk symbol `*` when being displayed on the screen. The implementation of the custom Online PIN CVM also required extending the `CardReader` interface with a new event type (`PromptForSoftPin` class) to be used to notify the UI to display the PIN pad dialog.

Since the Monet+ acquirer does not currently allow a key exchange with COTS devices, placeholder keys are currently used to encrypt the PIN block when sent to the acquirer alongside other transaction data.

The implemented software-based PIN pad does not currently meet all the requirements imposed by Visa guidelines, however, at this moment, its main purpose is to demonstrate the application's ability to perform transactions that require the Online PIN CVM.

Kernel Outcome Processing

Once the kernel finishes the processing of the transaction an instance of the `ContactlessResult` is returned by the `performTransaction` method. The result is processed in the `processKernelResult` method where further actions are taken based on the result's `finalOutcome` attribute that may be one of the following values:

- `COMPLETED` – the kernel has authorized the transaction and the Kernel

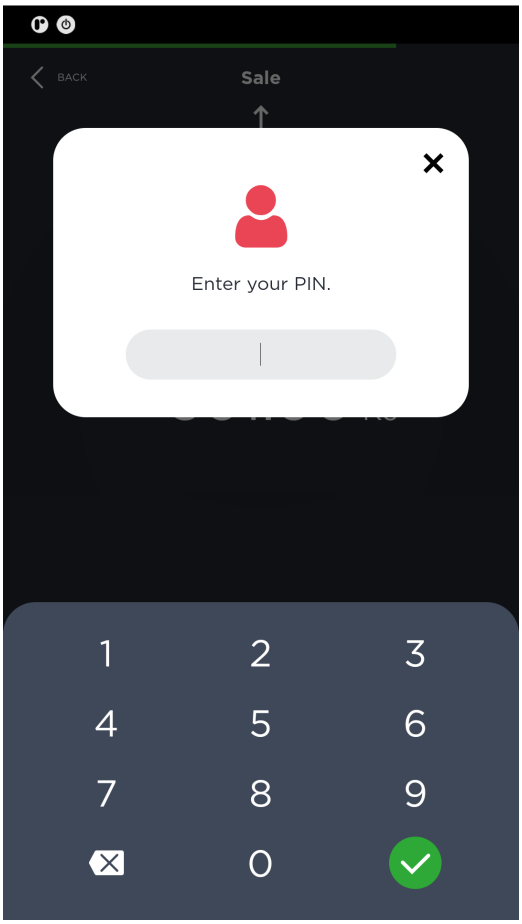


Figure 7.8: The software PIN pad used on COTS devices during the Online PIN CVM.

CVM Outcome (present in the tag `DF03`) should be examined to decide whether additional CVM should be performed (this may require the Online PIN CVM). When no CVM is required or it has been successfully performed, an online authorization request should be created and sent to the acquirer.

- `DECLINED` – the kernel has decided to decline the transaction offline, the user may choose to repeat the transaction and present another card.
- `ABORTED` – the kernel processing of the transaction was terminated, in this case, the user is prompted to present the card again.
- `TRYNEXT` – the selected application cannot be used to perform the transaction, the user is prompted to present another card.

- `SELECTAGAIN` – returned by the kernel when a mobile phone is presented but CDCVM is required, the user is prompted to follow the instructions on the mobile phone and present it again to the terminal.

Same as in the Nexgo **CardReader** implementation, when the initial processing has been done by the kernel the cardholder data alongside other data gathered from the card is wrapped inside an instance of the `OnlineRequest` event and eventually sent to the acquirer for online authorization. As opposed to the Nexgo implementation, here the online authorization result is not validated by the kernel (see the generation of the application cryptogram in 3.3.9) as the kernel does not support it at the moment.

Printing of Receipts

Since COTS devices usually do not have the capability to print paper receipts, other alternatives have to be considered when implementing a payment application. This includes utilizing a Bluetooth printer, sending the receipt to the customer's email address or displaying the receipt in form of a QR code to be scanned by the customer's phone. However, since this is out of the scope of this master's thesis, it has not been implemented yet in the Dotypay application. Further development of the application is required before being suitable to be released for production.

7.8 Recommendations for Future Development

From the observations made during the implementation part of this thesis, it would be recommended that the initialization of the EMV module (such as setting the list of supported AIDs and initial TLV tags) would be refactored, so that it can be shared between different **CardReader** implementations. Currently, the code responsible for this is duplicated in all 3 implementations of the **CardReader** component.

Additionally, some interfaces that were created based on the new architecture are difficult to be implemented, because they are too similar to the interface of the Landi A8 SDK. This is most likely caused by the fact that when the application was being refactored by the Dotypay team to use the new architecture, some parts of interfaces of the Landi A8 SDK were reused and the architecture was not implemented correctly in all parts of the code. As a result of this, some parts of the code cannot be easily shared between different implementations and there are several unused classes and methods in Nexgo and COTS implementations.

8 Testing the Validity of the Updated Solution

All major payment schemes mandate proper testing and certification before allowing card terminals to participate in their payment infrastructure.

One of the certifications required is the EMV Certification, which consists of 3 levels [48]:

- **Level 1** – consists of tests that verify whether the tested device meets the physical requirements and correctly implements all required low-level communication protocols,
- **Level 2** – validates the software that runs on Level 1 certified hardware. This includes the EMV Kernel/Library that implements the EMV communication protocol to exchange data with payment cards. Typically, the hardware supplier provides its own EMV kernel that runs internally within the device.
- **Level 3** – also known as EMV End-to-End certification, this certification can be achieved after successfully testing a payment application utilizing the Level 2 certified kernel running on a Level 1 certified device. This also requires the solution to already have chosen an acquirer or transaction processor to authorize transactions. Eventually, this level tests all the components that participate in an EMV transaction.

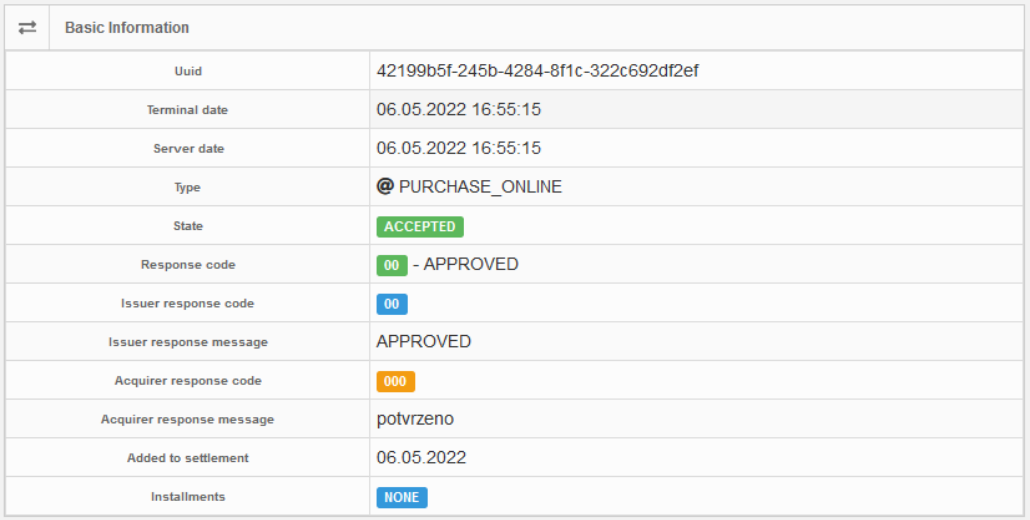
Another important certification is the PCI DSS Certification, which consists of a set of security standards formed by major card brands such as Visa, Mastercard or Discover [49]. It is governed by the Payment Card Industry Security Standards Council and aims to make card transactions as resilient against and fraud as possible. For example, PCI DSS security standards include the verification of the used network's security, encryption of transmitted cardholder data and guidelines for manipulation of sensitive data.

It is a challenging task for payment solution providers to achieve all the necessary certifications before being accepted by card scheme networks.

8.1 Transaction Authorization in a Test Environment

Both Visa and Mastercard card schemes allow providers of card payment solutions to test their products in a testing environment. Because the Monet+ acquirer used in the Dotypay application supports forwarding of transaction authorization requests to this environment, the extended application could be tested there. It should be noted that even in testing environments Monet+ requires that all transaction authorization requests are enciphered using keys loaded during a proper key injection.

Transactions initiated on Nexgo N86 devices running the Dotypay application were successfully authorized when forwarded to a testing environment (see figures 8.1 and 8.2).



Basic Information

Uuid	42199b5f-245b-4284-8f1c-322c692df2ef
Terminal date	06.05.2022 16:55:15
Server date	06.05.2022 16:55:15
Type	@ PURCHASE_ONLINE
State	ACCEPTED
Response code	00 - APPROVED
Issuer response code	00
Issuer response message	APPROVED
Acquirer response code	000
Acquirer response message	potvrzeno
Added to settlement	06.05.2022
Installments	NONE

Figure 8.1: A screenshot of the record of a successful Nexgo transaction authorization displayed in the Monet+ transaction history web application.

▼Terminal date	Terminal / Merchant	Amount	State / Approval code	IssRC/AcqRC / Processor / Processor Group	Dst stan / Rrn	Variable Symbol / Masked PAN
2022-05-06 16:55:15	T TSPAYN1	12.34 CZK	ACCEPTED	00 / 000	821737	
	M SOLITEAMERCH		005440	FIRSTDATA / FIRSTDATA	212616821737	541333******9130
2022-05-06 16:55:03	T TSPAYN1	12.34 CZK	ACCEPTED	00 / 000	821736	
	M SOLITEAMERCH		005439	FIRSTDATA / FIRSTDATA	212616821736	541333******9130
2022-05-06 16:41:53	T TSPAYN1	1.01 CZK	ACCEPTED	00 / 000	821709	
	M SOLITEAMERCH		092921	FIRSTDATA / FIRSTDATA	212616821709	476173******0027

Figure 8.2: A screenshot of successful Nexgo transaction authorizations displayed in the Monet+ transaction history web application.

8.2 The UL Brand Test Tool

The UL Brand Test Tool is a test tool that aims to ease the payment solution development and certification process preparations for acquirers, payment processors, terminal vendors and merchants that aim to connect to existing payment schemes [50]. It contains more than 100 guided test scenarios that cover all mandatory requirements imposed by EMV standards and major card schemes, such as Visa or Mastercard. To be able to execute these tests the tool includes all the necessary hardware such as the UL SmartLink Box that is used to emulate chip cards or UL SmartWave Box that is used to emulate contactless cards. It also contains a wide variety of preloaded test cards to be used during test scenario execution.

Using this tool it is possible to emulate payment cards and intercept, log and validate all the communication that is happening between the card and the card terminal, making it easy to trace bugs and issues.

Figure 8.3 contains an image of the setup UL Brand Test Tool used to test the Dotypay application running on the Nexgo N86 device. This figure contains the following annotations:

1. A SmartLink Probe connected to the SmartLink Box used to emulate chip cards.

2. The Nexgo N86 device running the Dotypay application that is currently waiting for a card to be presented to it to authorize a transaction.

3. A SmartLink Box that is powering the emulated chip card. To be able to display intercepted communication between the card and the card terminal this component is connected using a USB cable to a computer running the UL Tool application.

4. A SmartWave Box that is used to power and emulate contactless cards. It has a SmartWave Probe connected to it to be used as a payment card. Additionally, this component is connected to a SmartLink Box.

5. A SmartWave Probe on which the data of emulated contactless cards is loaded. It is powered by a SmartWave Box and can also be used in combination with any other standard card to intercept the communication between the card and the terminal.

The Windows application that is used in combination with the mentioned tool components contains a list of available test scenarios to be executed

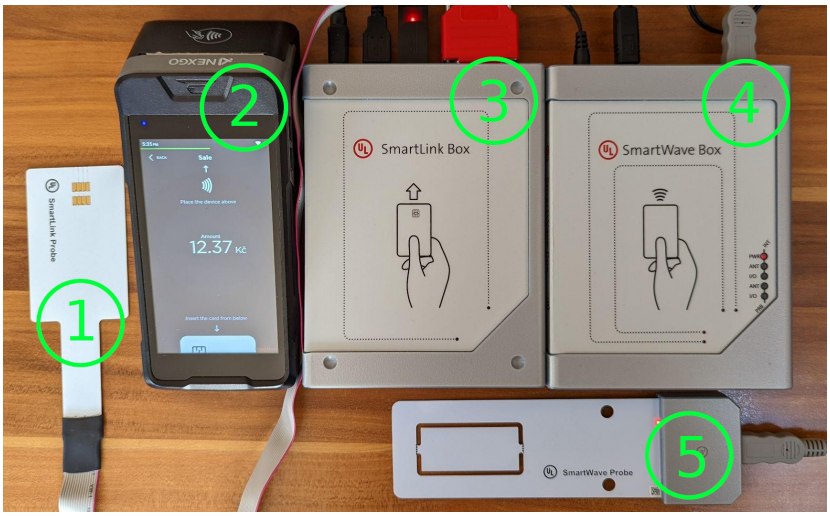


Figure 8.3: An image of the UL Brand Test Tool with annotations.

(see fig. 8.4). It provides text guidelines for each scenario to help the tester execute the test, as some tests require a particular transaction amount to be entered or some other manual action such as a cardholder signature. In some cases, the test evaluation is not automatic, but rather manual, where the user has to manually confirm using a checkbox in the application whether a particular condition has been fulfilled (e.g., the receipt has been printed).

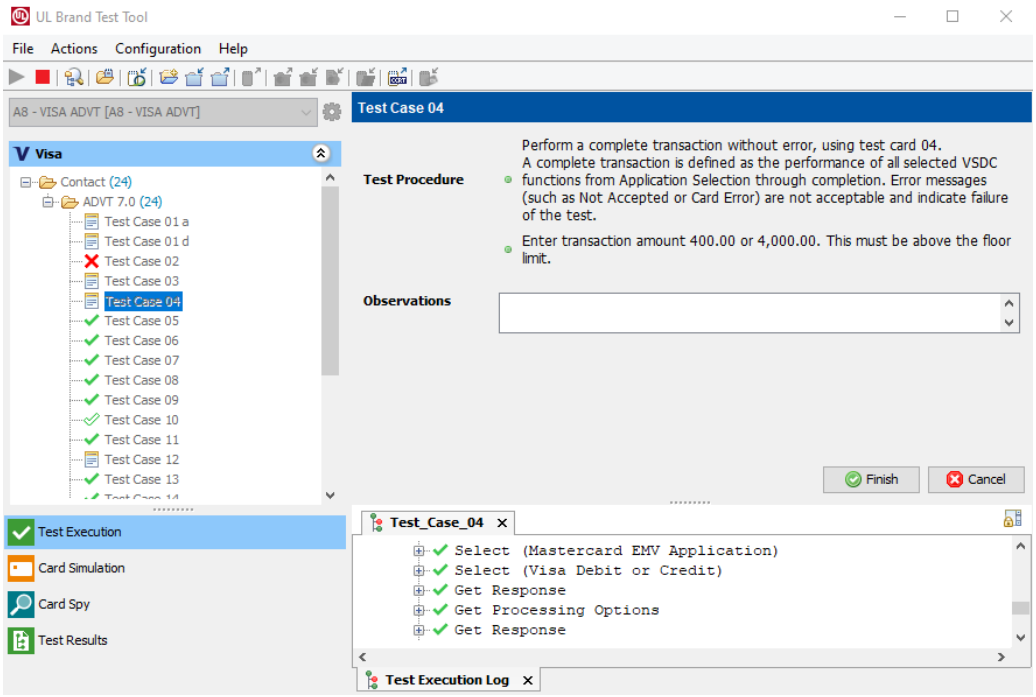


Figure 8.4: A screenshot of the UL Brand Test Tool Windows Application.

Additionally, the tool proved itself to be very useful when studying the EMV standard, as it provides necessary examples to better understand the theory behind the communication in the EMV standard.

8.3 Created Test Scenarios

To test the functionality of the Dotypay application running on Nexgo or COTS devices, 13 test scenarios have been designed that cover the fundamental features of a card terminal (see table 8.1).

Most of the test scenarios listed in the table 8.1 should result in the transaction being authorized online[1] and in scenarios, where PIN CVM is required, the tester should enter a valid PIN of the card used. The application should be tested on both Nexgo and COTS devices, however, scenarios requiring a card to be inserted into the device should be omitted on COTS devices as they do not support it.

It is important that test scenarios include both Mastercard and Visa transactions, as there might be some differences in the communication between the card and the terminal between these brands, that could cause issues in the transaction processing.

None of the tests include the use of the card's magnetic stripe as it is not currently implemented in any of the added device-specific implementations and it is expected that in the near future transactions that utilize the card's magnetic stripe will no longer be supported [51].

8.4 Test Results

To test the application running on Nexgo devices the Nexgo N86 device has been used during the testing. Oneplus 6 (Android 11) has been selected to test the application running on COTS devices.

8.4.1 Nexgo Devices

Out of the 13 test scenarios that were run to test the application on Nexgo devices 11 of them passed successfully. The tests that failed were tests #11 and #12 in which a tokenized Mastercard card was used to perform a contactless transaction. Due to unknown reasons, the Nexgo SDK EMV module always returns a general failure error when a tokenized Mastercard

[1]The Monet+ acquirer should forward transaction authorization requests to testing environments of card schemes.

#	Card type/brand	Amount	Description	UL-TOOL Test Name
1	Visa	< 500 CZK	Standard contact transaction with a PIN CVM.	ADVT 7.0 – Test Case 12
2	Visa	>= 500 CZK	Standard contact transaction with a PIN CVM above the CVM limit.	ADVT 7.0 – Test Case 12
3	Visa Contactless	< 500 CZK	Contactless transaction below the CVM limit, without a PIN CVM.	CDET 2.3 – Test Case 02
4	Visa Contactless	≥ 500 CZK	Contactless transaction above the CVM limit, requiring a PIN CVM.	CDET 2.3 – Test Case 01
5	Tokenized Visa Contactless	< 500 CZK	Mobile transaction under the CVM limit, without a CDCVM.	CDET – GP001
6	Tokenized Visa Contactless	*any*	Mobile transaction requiring a CDCVM.	CDET – GP004
7	Mastercard	< 500 CZK	Standard contact transaction with a PIN CVM.	M TIP06 Test 01 Scenario 01
8	Mastercard	≥ 500 CZK	Standard contact transaction with a PIN CVM above the CVM limit.	M TIP06 Test 01 Scenario 01
9	Mastercard Contactless	< 500 CZK	Contactless transaction under the CVM limit, without a PIN CVM.	MCD01 Test 01 Scenario 04
10	Mastercard Contactless	≥ 500 CZK	Contactless transaction above the CVM limit, requiring a PIN CVM.	MCD01 Test 01 Scenario 01
11	Tokenized Mastercard Contactless	< 500 CZK	Mobile transaction under the CVM limit, without a CDCVM.	CDET – GP103
12	Tokenized Mastercard Contactless	≥ 500 CZK	Mobile transaction above the CVM limit, requiring a CDCVM.	CDET – GP101
13	*any*	*any*	Use a card with multiple suitable applications, requesting an application to be selected by the cardholder.	ADVT 7.0 – Test Case 05

Table 8.1: A list of payment transaction scenarios designed to test the Dotypay application.

card is presented. To eliminate this issue, EMV transaction logs should be carefully analyzed to find out its cause. Most likely this is caused by a missing entry in the list of supported AIDs or a missing TLV tag in the initial

EMV configuration. Other transactions have been successfully processed on the terminal and were authorized by the payment network when applicable.

8.4.2 COTS Devices

On COTS devices, the only tests that could have been executed were tests #3, #4, #5 and #6. Other tests have been skipped because contact transactions cannot be performed on COTS devices and the Dotypay application is currently missing a Mastercard kernel.

All of the performed tests passed successfully and transactions initiated during these tests were correctly processed on the Oneplus 6 device, however, these transactions could not be authorized by the acquirer, as COTS devices currently do not support key injection. As a result of that, all transactions performed on the Oneplus 6 device were declined by the acquirer.

Although the UL Brand Test Tool did not report any errors during the transaction processing done by the COTS device, communication with the acquirer can't be established when the device has not been loaded with correct encryption keys. This suggests that if the COTS device was properly injected with encryption keys, all performed transactions would be authorized by the acquirer.

Additionally, the tests yielded the same results when the application was tested on Google Pixel 6 and Samsung S10e devices.

9 Conclusion

This master's thesis covers the fundamentals of the payment card industry and the set of EMV standards used in cards and card terminals to facilitate financial transactions. Because the volume of information in the covered standards is quite large (only the first four books of EMV consist of more than 500 pages), they could not be easily summarized in a compact way. However, a deep understanding of the aforementioned topics was necessary to design and implement a suitable architecture for a modular application that could be run on different devices with specific hardware capabilities related to the reading of payment cards.

A new architecture was designed and implemented within the scope of this thesis to extend the commercial Dotypay application by supporting Nexgo and COTS devices. Eventually, the application was able to successfully finish a transaction authorization on Nexgo N86. Although payment transactions are correctly processed on Android COTS devices running the Dotypay application (as verified by the UL Brand Test Tool), successful online authorizations are currently impossible there as the Monet+ acquirer does not support key injection on COTS devices at this moment. This is required to establish communication with the transaction authorization centre, but the application's architecture is prepared to be easily extended to support key injection on COTS devices when the necessary support and guidelines are available.

Furthermore, it would be advised to perform more refactoring of the application's code to better implement the new architecture, allowing easier maintainability of the codebase.

The UL Brand Test Tool is described in this master's thesis and was used to verify the ability of the extended application to correctly participate in EMV dialogues with payment cards. Almost all selected tests passed successfully when the Monet+ acquirer was used to route transactions to real payment networks on Nexgo N86. When additional work on the Dotypay application is done, such as the implementation of support for magnetic stripes or printing of receipts, the application will undergo various certifications to be able to be used in a production environment with Nexgo devices.

From testing the application on COTS devices, it has been observed that sometimes it is more difficult to successfully present a card to this type of devices, in comparison to standard card terminals. In general, it was confirmed that COTS devices can be used to accept payment cards and it would

be recommended that solutions using these devices are deployed in environments with lower transaction count, such as in barber shops, convenience stores, food stalls, etc., where small hold-ups during checkout are tolerable.

Additionally, based on the research done on card tokenization techniques, it is expected that COTS-based solutions will work best in markets with high mobile payment penetration, as they allow to replace the PIN CVM on COTS, which some customers might find unfamiliar and insecure, with CDCVM.

List of Acronyms

- **AEF** – Application Elementary File
- **AFL** – Application File Locator
- **AID** – Application Identifier
- **AIP** – Application Interchange Profile
- **APDU** – Application Protocol Data Unit
- **API** – Application Programming Interface
- **ARPC** – Authorization Response Cryptogram
- **ARQC** – Authorization Request Cryptogram
- **ATC** – Application Transaction Counter
- **ATM** – Automated Bank Teller
- **ATR** – Answer-to-Reset
- **BER-TLV** – Basic-Encoding-Rules - Tag-Length-Value
- **BIN** – Bank Identification Number
- **CDA** – Combined Data Authentication
- **CDCVM** – Consumer Device Cardholder Verification Method
- **CDE** – Cardholder Data Environment
- **CID** – Card Identification Number
- **COTS** – Commercial off-the-shelf
- **CPoC** – Contactless Payments on COTS
- **CSC** – Card Security Code
- **CVC** – Card Verification Code
- **CVK** – Card Verification Key
- **CVM** – Cardholder Verification Method

- **CVN** – Card Verification Number
- **CVR** – Cardholder Verification Rule
- **CVV** – Card Verification Value
- **DDA** – Dynamic Data Authentication
- **DOL** – Data Object List
- **DOL** – Data Object List
- **EMV** – Europay Mastercard Visa
- **FCI** – File Control Information
- **FID** – Fixed File Identifier
- **HCE** – Host Card Emulation
- **HSM** – Hardware Security Module
- **ICC** – Integrated Circuit Card
- **IC** – Integrated Circuit
- **IIN** – Issuer Identification Number
- **KCV** – Key Check Value
- **KEK** – Key Encryption Key
- **KLD** – Key Loading Device
- **MOTO** – Mail Order Telephone Order
- **NFC** – Near Field Communication
- **PAIE** – Proprietary Application ID Extension
- **PAN** – Primary Account Number
- **PCI** – Payment Card Industry
- **PCI SSC** – Payment Card Industry Security Standard Councils
- **PED** – PIN Entry Device
- **PICC** – Proximity Integrated Circuit Card

- **PIN** – Personal Identification Number
- **POS** – Point-of-Sale
- **PPSE** – Proximity Payment System Environment
- **PSP** – Payment Service Provider
- **RFID** – Radio Frequency Identification
- **RID** – Registered Identifier
- **RNG** – Random Number Generator
- **SCD** – Secure Cryptographic Device
- **SCRP** – Secure Card Reader for PIN
- **SDA** – Static Data Authentication
- **SFI** – Short File Identifier
- **SFTP** – Secure File Transfer Protocol
- **SPoC** – Software-based PIN Entry on COTS
- **TEE** – Trusted Execution Environment
- **TEK** – Traffic Encryption Key
- **TMK** – Terminal Master Key
- **TSI** – Transaction Status Information
- **TSP** – (Token Service Provider)
- **TTK** – Terminal Transport Key
- **TUI** – Trusted User Interface
- **TVR** – Terminal Verification Result

Bibliography

[1] I. Dubinsky, *Acquiring card payments.* Auerbach, 2019.

[2] J. Bengtsson, "Diving into magnetic stripe card skimming devices," *Digital Evidence and Electronic Signature Law Review*, vol. 5, 01 2014.

[3] S. Clar. (2022, Feb) Contactless payment transaction limit increases around the world. [Online]. Available: https://www.nfcw.com/2020/03/26/366173/table-contactless-payment-transaction-limit-increases-around-the-world/

[4] A. Woodyatt. (2020, Mar) Contactless card spending limit increased to tackle coronavirus spread. [Online]. Available: https://edition.cnn.com/2020/03/24/tech/contactless-payment-limit-covid-gbr-intl-scli/index.html

[5] R. d. Best. (2021, Jul) Visa, MasterCard, UnionPay Transaction Volume 2020. [Online]. Available: https://www.statista.com/statistics/261327/number-of-per-card-credit-card-transactions-worldwide-by-brand-as-of-2011/

[6] (2018) Authorization vs Clearing vs Settlement. [Online]. Available: https://doc.payneteasy.com/technology_overview/authorization_vs_clearing_vs_settlement_en.html

[7] (2020) Authorization and Reversal Processing Requirements for Merchants. [Online]. Available: https://usa.visa.com/dam/VCOM/global/support-legal/documents/best-practices-authorization-and-reversal-processing.pdf

[8] (2021, Oct) EMV Card-Present Transaction Percentage. [Online]. Available: https://www.emvco.com/about/deployment-statistics/

[9] (2011, Nov) Application Independent ICC to Terminal Interface Requirements. [Online]. Available: https://www.emvco.com/wp-content/plugins/pmpro-customizations/oy-getfile.php?u=wp-content/uploads/documents/EMV_v4.3_Book_1_ICC_to_Terminal_Interface_2012060705394541.pdf

[10] (2011, Nov) Security and Key Management. [Online]. Available: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_2_Security_and_Key_Management_20120607061923900.pdf

[11] (2011, Nov) Application Specification. [Online]. Available: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_3_Application_Specification_20120607062110791.pdf

[12] (2011, Nov) Cardholder, Attendant, and Acquirer Interface Requirements. [Online]. Available: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_4_Other_Interfaces_20120607062305603.pdf

[13] (2020, Mar) Book A - Architecture and General Requirements. [Online]. Available: https://www.emvco.com/wp-content/plugins/pmpro-customizations/oy-getfile.php?u=wp-content/uploads/documents/EMV-Contactless-Book-A-Architecture-and-General-Rqmts-v2.9-April-2_logoupdate.pdf

[14] (2004) ISO 7816 - Smart Card Standards Overview. [Online]. Available: https://smartcardsupply.com/Content/Cards/7816standard.htm

[15] (2013, Nov) SIM card forensics: An introduction. [Online]. Available: https://resources.infosecinstitute.com/topic/sim-card-forensics-introduction/

[16] (2001, September) Smart Card HOWTO. [Online]. Available: https://tldp.org/HOWTO/pdf/Smart-Card-HOWTO.pdf

[17] (2020, Dec) Complete list of Application Identifiers (AID). [Online]. Available: https://emv.cool/2020/12/23/Complete-list-of-application-identifiers-AID/

[18] A. Schwier. (2010) Static Data Authentication (SDA). [Online]. Available: https://www.openscdp.org/scripts/tutorial/emv/SDA.html

[19] (2016, Mar) Book A - Architecture and General Requirements. [Online]. Available: https://www.emvco.com/wp-content/uploads/2017/05/Book_A_Architecture_and_General_Rqmts_v2_6_Final_20160422011856105.pdf

[20] (2021) EMV MasterCard Contactless Transaction Flow. [Online]. Available: https://www.level2kernel.com/emv-mastercard-contactless-transaction.html

[21] (2022) Google Pay Safety & Security Features - Google Safety Center. [Online]. Available: https://safety.google/pay/

[22] (2022) Apple Pay security and privacy overview. [Online]. Available: https://support.apple.com/en-us/HT203027

[23] (2022, Apr) A Guide to Use Cases. [Online]. Available: https://www.emvco.com/wp-content/plugins/pmpro-customizations/oy-getfile.php?u=wp-content/uploads/documents/EMVCo-Payment-Tokenisation-A-Guide-To-Use-Cases-v2.2.pdf

[24] M. Fehr. (2018, Apr) Apple Pay: How different is it from other Pay solutions, what role does tokenisation play, and to what degree can Card not Present payment benefit from Apple Pay in future. [Online]. Available: https://www.royalholloway.ac.uk/media/5612/rhul-isg-2018-3-techreport-marcelfehr.pdf

[25] (2014) Apple Announces Apple Pay. [Online]. Available: https://www.apple.com/newsroom/2014/09/09Apple-Announces-Apple-Pay/

[26] (2018, May) Introduction to Secure Elements. [Online]. Available: https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf

[27] (2021) Where you can travel on public transport using Apple Pay. [Online]. Available: https://support.apple.com/en-gb/HT207958

[28] A. Radu, T. Chothia, C. J. Newton, I. Boreanu, and L. Chen. (2021, Aug) Practical EMV Relay Protection. [Online]. Available: https://practical_emv.gitlab.io/assets/practical_emv_rp.pdf

[29] G. Wong, "Google Wallet officially announced," May 2011. [Online]. Available: https://www.ubergizmo.com/2011/05/google-wallet-officially-announced/

[30] (2012) Google Wallet Security: PIN Exposure Vulnerability. [Online]. Available: https://zvelo.com/google-wallet-security-pin-exposure-vulnerability/

[31] B. Popper. (2015, May) Google introduces Android Pay a replacement for its wallet app on mobile. [Online]. Available: https://www.theverge.com/2015/5/28/8661867/google-introduces-android-pay-replace-wallet-app

[32] (2021) How payments work - Google Pay Merchant Help. [Online]. Available: https://support.google.com/pay/merchants/answer/6345242?hl=en#zippy=%2Cdetailed-google-pay-transaction-process-in-stores

[33] (2021) Host-based card emulation overview | Android Developers. [Online]. Available: https://developer.android.com/guide/topics/connectivity/nfc/hce.html

[34] G. Marwaha. (2014) Apple Pay vs Google Wallet : The Secure Element. [Online]. Available: http://www.gmarwaha.com/blog/2014/10/02/apple-pay-vs-google-wallet-the-secure-element/

[35] (2020, June) Software-based PIN Entry on COTS (SPoC)™ - Security Requirements. [Online]. Available: https://www.pcisecuritystandards.org/documents/SPoC_SecurityRequirements-v1.1.pdf

[36] (2019, December) Contactless Payments on COTS (CPoC™) - Security and Test Requirements. [Online]. Available: https://www.pcisecuritystandards.org/documents/Contactless_Payments_on_COTS-Security_and_Test_Requirements-v1.0.pdf

[37] D. Curry. (2022, Jan) Mobile payments app revenue and usage statistics (2022). [Online]. Available: https://www.businessofapps.com/data/mobile-payments-app-market/

[38] R. Hayton, B. Gourdin, D. Keating, and A. Lindsay. (2021, May) What is CPOC + PIN? Preparing for the PCI's next evolution in contactless payments. [Online]. Available: https://www.trustonic.com/whitepapers/cpoc-plus-pin-contactless-payments/

[39] (2022) Visa Ready Tap to Phone Program. [Online]. Available: https://partner.visa.com/site/programs/visa-ready.html

[40] (2022, Jan) Tap on Phone Implementation Guide. [Online]. Available: https://www.mastercard.com/content/dam/public/mastercardcom/na/global-site/documents/tap-on-phone-implementation-guide-jan2022.pdf

[41] A. Jamieson. (2020) Commercial Off-the-Shelf (COTS) Devices - Implementation Challenges . [Online]. Available: https://www.ul.com/resources/commercial-shelf-cots-devices-implementation-challenges

[42] (2021) Core NFC | Apple Developer Documentation. [Online]. Available: https://developer.apple.com/documentation/corenfc

[43] (2022) Dagger. [Online]. Available: https://dagger.dev/

[44] (2022) What is Koin? [Online]. Available: https://insert-koin.io/docs/reference/introduction

[45] I. O. for Standardization, *Cards and security devices for personal identification — Contactless proximity objects — Part 4: Transmission protoco*, iso/iec 14443-4:2018 ed. Vernier, Geneva, Switzerland: International Organization for Standardization, 2018. [Online]. Available: https://www.iso.org/standard/73599.html

[46] (2021) IsoDep | Android Developers. [Online]. Available: https://developer.android.com/reference/android/nfc/tech/IsoDep

[47] (2021) Tag | Android Developers. [Online]. Available: https://developer.android.com/reference/android/nfc/Tag

[48] (2020, July) EMV Certification Process in 3 Easy Steps. [Online]. Available: https:
//idtechproducts.com/blog/emv-certification-process-in-3-easy-steps/

[49] (2019, December) PCI DSS Certification. [Online]. Available: https://www.imperva.com/learn/data-security/pci-dss-certification/

[50] (2019, Aug) UL Brand Test Tool. [Online]. Available: https://www.ul.com/sites/g/files/qbfpbp251/files/2021-08/Fact-sheet-UL-Brand-Test-Tool_201905.pdf

[51] V. Hyman. (2021, Aug) Swiping left on magnetic stripes. [Online]. Available: https://www.mastercard.com/news/perspectives/2021/magnetic-stripe/

A Source Code Listing

The following text lists source files of the Dotypay application that have been added or updated during this master's thesis. However, minor or non-significant changes are omitted there.

A.1 Files Specific to Nexgo CardReader Implementation

All files that were added to extend the application with the support for Nexgo devices are located in the `com.stpos.a8pos.devicesdk.impl.nexgo` package. The following files from that package are of the most importance:

- `NexgoCardReader.kt` – implementation of the `CardReader` interface, where the most important method is the `searchForCard` method, which is used as the starting entry-point of a card transaction on Nexgo devices,
- `NexgoRxPinPad.kt` – implementation of the `RxPinPad` interface specific to Nexgo devices, where methods required for injection of PIN working keys, calculation of a KCV value of a key and encryption or decryption using injected keys are implemented. The method that allows to encrypt or decrypt data using the injected keys is used when encrypting messages sent to the acquirer and without it, online transaction authorizations would not be possible;
- `card/NexgoCardReaderHelper.kt` – contains a helper method that wraps the call of the method from the Nexgo SDK used to initialize the searching for cards,
- `emv/NexgoEmvDataConverter.kt` – defines methods that are used to convert the list of objects containing certificates of supported card issuers and properties of supported card applications from Dotypay class instances to instances of classes from the Nexgo SDK,
- `emv/NexgoEmvInitializer.kt` – defines a method used to initialize the device's EMV component with the list of supported card applications and card scheme certificates,

- `emv/NexgoEmvProcessHelperImpl.kt` – contains a helper method that initializes the EMV transaction and processes it according to the device's and card's capabilities (e.g., prompts for PIN or creates an online authorization request).

A.2 Files Specific to COTS CardReader Implementation

All files that were added to extend the application with the support for COTS devices are located in the `com.stpos.a8pos.devicesdk.impl.cots` package.

The `COTSCardReader.kt` file contains the implementation of the `CardReader` interface. Inside the `searchForCardInternal` method the device's NFC interface is activated and the process of application selection is initiated using the `ppse.PPSEManager` class extracted from the Visa TTP SDK. When a supported Visa card application is found, the `VisaTTPKernelHelper` class (located in the `visa/VisaTTPKernelHelper.kt` file) is used to call the Visa TTP kernel. This class is responsible for correctly initializing the kernel as well as setting all necessary TLV tags. Online authorization results are not validated there because the Visa TTP kernel does not currently support it.

B User Guide

B.1 Building the Application

Before being able to compile and assemble Android applications, one must have the Android SDK installed and configured in his development environment. The official Android documentation[1] contains a guide that involves installing the Android Studio IDE and setting up all the necessary tools (including the Android SDK).

The Dotypay application can then be built to operate in one of 3 different modes (flavors):

- `mock` – transactions are not sent to the acquirer and online authorization results are rather mocked,
- `stTest` – transactions are routed to the testing environment of the Monet+ acquirer,
- `production` – transactions are routed to the production environment of the Monet+ acquirer.

Additionally, it can also be selected whether the application is built to be run in debug or release mode (`debug` and `release`).

An APK that can be installed on supported devices (Android SDK Level 22 – Android 5.1 or higher) is assembled by selecting the build mode and flavor and executing the following command in the root of the application's project's directory:

```
./gradlew assemble{flavor}{Debug|Release}
```

This produces an APK archive in the `./app/build/outputs/apk` folder.

For example, the debug variant of the application, where transactions are routed to the testing environment of the acquirer, can be built by executing the following command:

```
./gradlew assembleStTestDebug
```

[1] `https://developer.android.com/studio/install`

C Operating the Application

The complete user guide to the Dotypay application can be found on the following website: `https://manual.dotypay.com/1/cz/topic/uvod` .

Credentials to be used to log in to the application[1] are listed in the Table C.1.

Role	**PIN**
Cashier	`0000`
Manager	`1111`

Table C.1: Default login credentials.

[1]These credentials are present only in the default configuration that is preloaded when the application is installed.