Mobile Payment using HCE and mPoint payment gateway based on NFC enabled phones.

AUTHOR : GRZEGORZ MILCARZ
S111040

DATE NOVEMBER 27, 2014

Summary

The goal of the thesis is to create a proof of concept that NFC payments can be performed using mPoint payment gateway and Android device.

The technology used in this project is mPoint payment gateway, mPoint SDK, together with NFC enabled Android devices that support HCE technology.

The final goal of the system is to cooperate with POS terminals in the shops, however for this thesis second Android phone was used to emulate the terminal behavior.

One of the devices is configured to act as a customer application emulating the credit card using HCE technology, second device is configured to mimic the behavior of POS terminal and read the credit card data passed via NFC radio.

Results

This thesis proved that mPoint payment gateway with mPoint SDK is capable of handling NFC payments.

Acknowledgements

I would like to thank both my supervisors Stig Høgh and Jonatan Evald Buus, for the supervision of this project. For all the help I received from them, patience and knowledge shared.

Abbreviations and Symbols

NFC – Near Field Communication

HCE – Host-based Card Emulation

NDEF – NFC Data Exchange Format

SE – Secure Element

API – Application Programming Interface

OS – Operating System

GNU – Gnu`s Not Unix

AID – Application ID , Application Identifier

SDK – Software Development Kit

ISO-DEP – ISO/IEC 14443-4 – standard defining near field contactless cards and protocols used during communication

XML – Extensible Markup Language

GUI – Graphical User Interface

Contents

Summary 2
Results 3
Acknowledgements 4
Abbreviations and Symbols 5
Introduction 8
Technology 11
NFC – Near Field Communication 11
NFC – Tag Dispatch System 12
NFC Implementation 14
HCE – Host-based Card Emulation 15
HCE Service 17
HCE Service Implementation 17
Android 19
mPoint Server 20
mPoint SDK 21
mPoint SDK - Implementation 21
Payment Initialization 21
Payment Authorisation 22
Status Handling 22
Project 23
Payment Flows 24
Stored Funds Payment 24
Credit Card Payment 25
Approach 1 25
Approach 2 26
mPoint SDK 28
Card Emulation 28
contactlessPayment() method 28
Card Detection 29
login() 29
transfer() 30
initialize() 31
authorize() 32
Utility methods 33
mPoint Server 33
Login 33
Transfer 34
Initialize 35
Authorize 35

Integration 36
processCommandApdu() 36
OnTagDiscovered() 36
handleSuccessfulLogin() 36
handleInitialization() 36
displayPaymentConfirmation() 36
handleStatus() 36
handleError() 37
Performance 37
User Application 38
Merchant Application 46
Discussion and Conclusions 48
Extras 49
List of Figures 49
List of Tables 49
List of Codes 49
References 51
Appendixes 52
Error Codes 52

Introduction

In Modern World mobile technology is accompanying us in every part of our lives. Mobile phones are slowly replacing a need for personal computer offering almost the same performance in normal usage and having advantage of being mobile and small. Phones are used to write e-mails, play games, monitor health or set up meetings. Thanks to technological advance mobile phones can now replace credit cards. Near Field Communication radio with Host-based Card Emulation technology can emulate user`s smart card. Now it is possible to completely replace users wallet using mobile phone.

This thesis goal is to create a proof of concept that mobile payments can be performed using Near Field Communication and Host-based Card Emulation technologies with help of mPoint server and mPoint SDK on Android devices. For the purpose of this thesis payment will be performed between two Android phones that support NFC technology. One phone will act as a Customer Device emulating the card, as second device will be used as an NFC reader to emulate the use of POS terminal.

This thesis focuses of key communication between customer – merchant device, as well as describing the work flow of the payment process that happens between merchant device using mPoint SDK and mPoint Server. Simplified overview of this communication can be seen below. Green arrows highlight focus of this thesis.

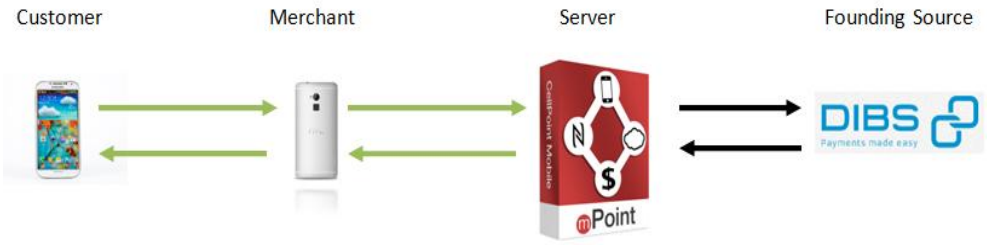


Figure 1. Simplified Communication Schema

This project will showcase the ability of mPoint to easily integrate into already existing application and seamlessly perform payments. Wallet application will be used as a basis and will be modified to handle NFC payments. The idea is to present the real life scenario of merchant application that can handle payments. Most of shops already develop their own application that handles Loyalty programs or notify about promotions. This system can be easily integrated into

those applications and create one centralized system. The complete workflow of the system is illustrated on Figure 2. System Overview. Application workflow is defined by following steps:

1. User Account Registration
2. Card Emulation
3. Payment using mPoint
4. Card authorization
5. Payment Confirmation

Thesis main focus are steps number 2 and 3 highlighted with green arrows on the picture. Step 2 handles card emulation and detection, while step 3 is responsible for verification and transfer of money.

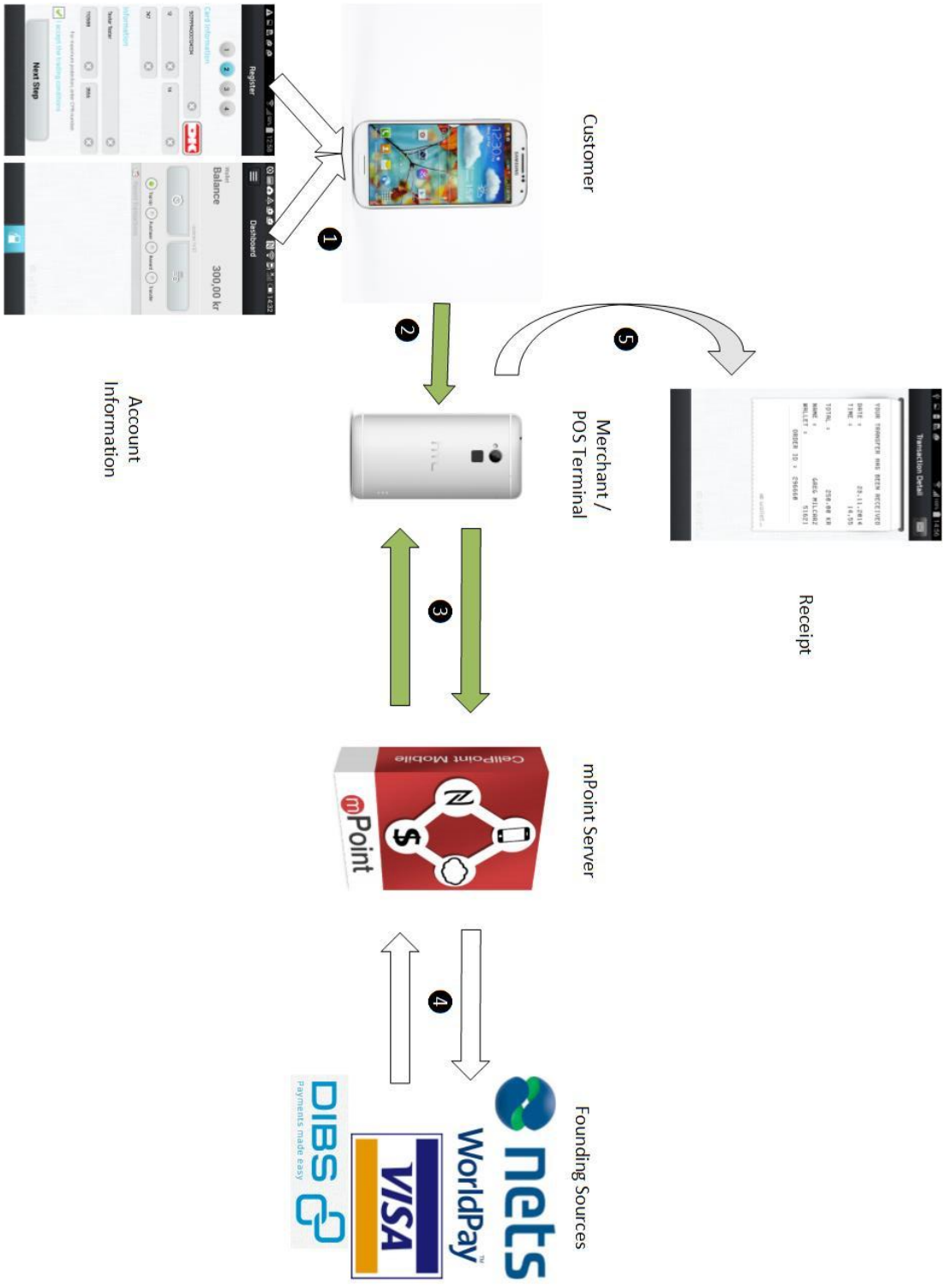


Figure 2. System Overview

Technology

NFC – Near Field Communication

Near Field Communication (NFC) is a short-range, high frequency radio standard allowing for wireless data transmission. This technology is a simple extension of the ISO/IEC 14443 (proximity card) standard that connects the interface of a smart card and reader in one device. NFC devices can communicate both with existing ISO/IEC 14443 standard card and readers, as well as with other NFC devices, making it compatible with already existing public transport and payment infrastructure. NFC is mainly focused for usage in mobile devices.

Near Field Communication dates back to 1983 when the first patent was associated with the abbreviation RFID and granted to Charles Walton. Later on in year 2002 Sony and Phillips on establishing a new technology specification and created a technical outline. In 2004 Nokia, Sony and Phillips joined Near Field Communication (NFC) Forum - a non-profit producers association established to develop contactless communication technology, later in year 2006 Nokia 6131 become the first NFC enabled phone. In year 2010 Samsung`s Nexus S became the first Android NFC enabled phone and NFC started to gain more attention.

Near Field Communication operates on 13.56 MHz on ISO/IEC 18000-3 and at rates ranging from 106 kbit/s to 424 kbit/s. Theoretical working distance is up to 20 centimeters, however practical working distance is about 4 centimeters.

NFC has two communication modes:

- Passive communication mode: The initiator device provides the carrier field and the target device answers by modulating the existing field.
- Active communication mode: Both initiator and target devices communicate by alternately generating their own fields. A device deactivates its RF (radio frequency) field while it is waiting for data.

However there are 3 main operating modes that Android devices support:

- Read/Write mode – android devices can read from or write to passive NFC tags.
- P2P mode – android devices can exchange data with other NFC peers. Used by Android Beam.
- Card emulation mode – android device can act as NFC card.

Basic functionality of NFC on Android devices it reading NDEF data from NFC tags or beaming NDEF messages between devices with Android Beam. This

thesis interest is not in NDEF message exchange, but it is important to understand how tad dispatch system works on android devices.

NFC – Tag Dispatch System

NFC enabled android devices is looking for NFC tags when device screen is turned on. When NFC tag is discovered, desired behavior is to start the correct application without having to ask user which one it is. Since NFC works on a small range it is possible that selecting correct application by user will break the connection by moving the device away from the reader. That is why ones application should be designed to only handle specific NFC tags that are relevant and can be handled appropriately. Android have its own tag dispatch system to help with this issue. Tag dispatch system works in three simple steps:

1. Parse the NFC tag and figure out MIME type or URI to identify payload in the tag.
2. Encapsulate MIME type or URI and the payload into an intent.
3. Start an activity based on intent.

There are 3 intents defined in tag dispatch system that works on the priority basis shown on (PICTURE NUMBER) and listed below:

1. ACTION_NDEF_DISCOVERED – used when scanned tag contains an NDEF payload and is of recognized type.
2. ACTION_TECH_DISCOVERED – used when tag contains NDEF data that cannot be mapped to MIME type or URI or does not contain NDEF data but is of known tag technology.
3. ACTION_TAG_DISCOVERED – started when there is no activity handlers defined for two previous ones.

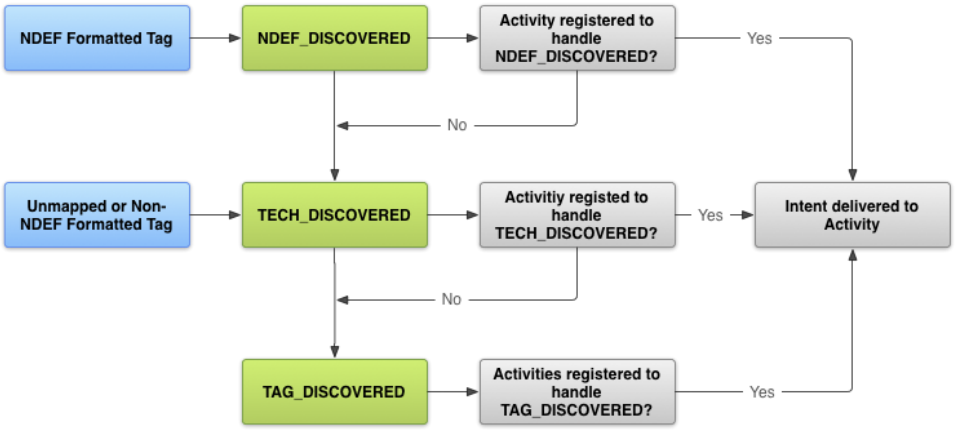


Figure 3. Tag Dispatch System **[2]**

Work in this thesis will be dealing with ACTION_TECH_DISCOVERED intents. There are several supported technologies in the *android.nfc.tech* package. They are listed in the (TABLE NUMBER). Technology used this project is IsoDep, that allows for sending raw ISO-DEP data to the tag and receive response, using *byte[] transceiver(byte[] data)* method.

Class	Description
`TagTechnology`	The interface that all tag technology classes must implement.
`NfcA`	Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
`NfcB`	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
`NfcF`	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
`NfcV`	Provides access to NFC-V (ISO 15693) properties and I/O operations.
`IsoDep`	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
`Ndef`	Provides access to NDEF data and operations on NFC tags that have been formatted as NDEF.
`NdefFormatable`	Provides a format operations for tags that may be NDEF formatable.

Table 1. Supported tag technology

NFC Implementation

NFC support implementation begins with declaring access to NFC features in *AndroidManifest.xml* file:

```
<uses-permission android:name="android.permission.NFC" />
<uses-sdk android:minSdkVersion="10"/>
<uses-feature android:name="android.hardware.nfc" android:required="true" />
```

Code 1. NFC features declarations **[2]**

Element *<uses-permission>* is used to access NFC hardware. Element *<uses-sdk>* specifies minimum required SDK version. API lower than level 10 only support limited tad dispatch via ACTION_TAG_DISCOVERED. Element *<uses-feature>* lists application in Google Play store only for devices that support NFC hardware. Second step is to create an XML resource file listing the technologies that activity supports with tech-list set. Following sample specifies all of the technologies for ACTION_TECH_DISCOVERED intent.

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.IsoDep</tech>
        <tech>android.nfc.tech.NfcA</tech>
        <tech>android.nfc.tech.NfcB</tech>
        <tech>android.nfc.tech.NfcF</tech>
        <tech>android.nfc.tech.NfcV</tech>
        <tech>android.nfc.tech.Ndef</tech>
        <tech>android.nfc.tech.NdefFormatable</tech>
        <tech>android.nfc.tech.MifareClassic</tech>
        <tech>android.nfc.tech.MifareUltralight</tech>
    </tech-list>
</resources>
```

Code 2. XML file listing nfc.tech technologies **[2]**

Similar file listing only supported technologies should be created and saved under *<project-root>/res/xml* folder.

Last step is to specify XML resource file in *AndroidManifest.xml* file in the *<meta-data>* element under the *<activity>* element, similar to the example shown on Code 3. <meta-data> element declaration in Manifest.xml

[2]

```
<activity>
...
<intent-filter>
    <action android:name="android.nfc.action.TECH_DISCOVERED"/>
</intent-filter>

<meta-data android:name="android.nfc.action.TECH_DISCOVERED"
```

```
    android:resource="@xml/nfc_tech_filter" />
...
</activity>
```

Code 3. <meta-data> element declaration in Manifest.xml **[2]**

HCE – Host-based Card Emulation

Host Card Emulation (HCE) is a method of representing a smart card on a mobile device using only software. Before HCE, NFC payments were relying on using the Secure Element, that stored the card credentials on a piece of hardware inside the users phone.

Term Host Card Emulation was created in 2011 by founders of SimplyTapp, Doug Yeager and Ted Fifelski. A year later in August 2012 the first payment wallet using NFC technology was released. However HCE technology needed direct access to the Android OS that was not officially supported by Google at the point. Meaning only technological enthusiasts with rooted (modified) versions of Android were able to use the application.

In October 31, 2013 Android adopted the HCE technology. Google hoped that by including HCE within the Android OS, the largest mobile OS will gain a rapid growth in mobile payments ecosystem. In year 2014 Visa and MasterCard announced coming support for HCE.

HCE allows for secure element to be outside the mobile device, bypassing Secure Element Issuers and passing control directly to the developers, without third-party involvement.

It allows NFC data transfer between Near Field Communication terminal or NFC enabled device and smart card or mobile device configured to emulate the smart card. Host Card Emulations requirement is access to main Android Operating System (Android OS) to bypass the need to access local hardware-based Secure Element (SE) chip configured to act like a card.

Secure Element (SE) is a hardware. It is a protected memory, granting access only to applications having valid key, that can be issued from the chipset manufacturer. SE security is based on invalid key entry attempts and is disabling access completely.

When NFC card emulation is settled up to work with secure element all the communication between NFC reader and device is rooted to the secure element. As pictured on Figure 4. Provisioning using physical Secure Element. This concept puts whole communication with NFC reader in hand of Secure Element manufacturer, no application on the device can interrupt that transaction. After

the transaction is made, desired application can call secure element directly for transaction status and display the status to the user.

When NFC card is emulated using HCE Figure 5. Provisioning using Secure Element stored in the cloud - HCE, communication from NFC reader is rooted directly to device CPU on which all the applications are working.

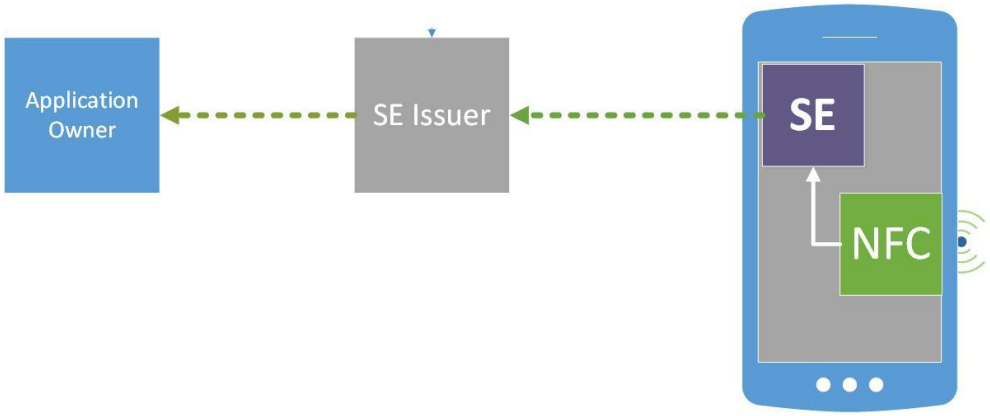


Figure 4. Provisioning using physical Secure Element

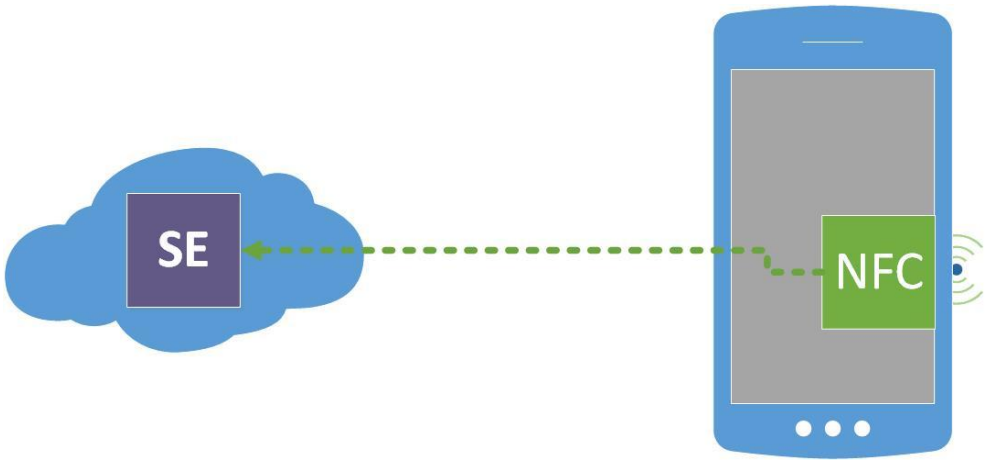


Figure 5. Provisioning using Secure Element stored in the cloud - HCE

HCE Service

The best advantage of using HCE is the usage of Android Service components. Android Service components are designed to work in background not to interfere with normal usage of the phone. In case of mobile payments, it gives a great advantage of being able to emulate the smart card while doing anything else on the phone. One does not have to start the appropriate application to make a payment, the only requirement is to unlock the phone and put it in the proximity of NFC reader.

Service selection is very important since phone needs to know which HCE service should be started for particular NFC reader. Service selection is based on Application ID (AID) and specified in ISO/IEC 7816-4 document. AIDs of different payment systems are a public knowledge and registered accordingly to ISO/IEC 7816-5 specification to avoid collisions and multiple systems trying to handle different requests. As this projects is a proof of concept and is not being deployed to the market yet, AID is not being officially registered and choose just to work between two selected phones without interfering with the system.

Android distinguishes two categories CATEGORY_PAYMENT and CATEGORY_OTHER. This separation is designed to allow multiple HCE services in the system. CATEGORY_PAYMENT is storing AID groups from application handling all or most of credit card protocols. However only one AID group can be enabled by default in the system, meaning only one payment application can be enabled on the users phone. CATEGORY_OTHER keeps all the other AID groups that are to be used in one merchant only situations. This category allows, that all AID groups defined in it, to be always active in background and take priority while NFC reader AID selection.

HCE Service Implementation

Host-based Card Emulation implementations begins with check if device supports HCE feature. This is done by use of *<uses-feature>* tag in the manifest of the app. FETURE_NFC_HOST_CARD_EMULATION needs to be set to true.

Android 4.4 comes with the service class that can be used as a basis of Service component implementation, the *HostApduService* class that need to be extended. Thos class contains two abstract methods that developer has to implement to handle the process.

```
public class MyHostApduService extends HostApduService {
    @Override
    public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
        ...
    }
```

```
    @Override
    public void onDeactivated(int reason) {
        ...
    }
}
```

Code 4. HostAdpuService Example **[3]**

processCommandApdu() method is being called each time NFC reader sends an Application Protocol Data Unit (APDU) to the service. APDUs are packets exchanged between NFC reader and HCE service, that are being send as a request and response. Normally the first APDU sent by NFC reader is "SELECT AID" request, containing Application ID that it wants to communicate with. Android isolates this AID and selects appropriate HCE service.

onDeactivated() method is called in one of two cases. First when "SELECT AID" request was resolved to correct service, or when the link between device and reader had split.

Implemented service needs to be declared in the manifest with some modifications. Example of such declaration is shown in Code 5. HostApduService declaration in Manifest.xml file example.

```
<service android:name=".MyHostApduService" android:exported="true"
         android:permission="android.permission.BIND_NFC_SERVICE">
    <intent-filter>
        <action
android:name="android.nfc.cardemulation.action.HOST APDU SERVICE"/>
    </intent-filter>
    <meta-data android:name="android.nfc.cardemulation.host_apdu_service"
               android:resource="@xml/apduservice"/>
</service>
```

Code 5. HostApduService declaration in Manifest.xml file example **[3]**

Service declaration requires *android.permission.BIND_NFC_SERVICE* permission and *android:exported* attribute set to true to be able to work with external applications. Additionally *<intent-filter>* declaring the Intent that is handled by that service. Finally the *<meta-data>* tag with resource file that specifies which AID groups are requested by the service. Example of resource file containing AID group is shown in Code 6. Resource file defining AID group.

```
<host-apdu-service
xmlns:android="http://schemas.android.com/apk/res/android"
           android:description="@string/servicedesc"
           android:requireDeviceUnlock="false">
    <aid-group android:description="@string/aiddescription"
               android:category="other">
        <aid-filter android:name="F0010203040506"/>
```

```
        <aid-filter android:name="F0394148148100"/>
    </aid-group>
</host-apdu-service>
```

Code 6. Resource file defining AID group **[3]**

The last thing to do is to give the application permissions to use NFC and HCE features.

```
<uses-permission android:name="android.permission.NFC" />
<uses-feature android:name="android.hardware.nfc" android:required="true" />
```

Code 7. NFC & HCE permissions **[3]**

Android

Android Operating System (Android OS) is the most common mobile operating system in the world. Google started developing this open source operating system in year 2006, later in year 2007 Open Handset Alliance was established by Google, HTC, Motorola, Intel, Qualcomm, T-Mobile, Sprint Nextel and NVIDIA with an aspiration to develop open standards for mobile phones. First Android OS was presented. However the first public release of Android OS happened in year 2008.

Android OS is a Linux system and it is based on Linux kernel and GNU licensed software. Android OS was design primary for mobile phones with touch displays, however through the years of development and numerous iterations it grew to the position of operating system for mobile phones, tablets, wearable devices, television or even cars. Android applications are commonly written using Java programming language using Android application programming interface (API).

With latest Android 4.4 Kitkat Google integrated HCE within the operating system itself allowing developers to use this new technology. Together with most of new devices being NFC enabled, it create a enormous opportunity to increase the mobile payment market and make it easy accessible and effortless for millions of users.

mPoint Server

mPoint is a powerful payment gateway through which clients have access to multiple funding sources for securely processing payments. mPoint has been architected to provide cloud based access to available sources across merchant channel. The diagram below illustrates how mPoint provides access to multiple funding sources for merchants and customers.

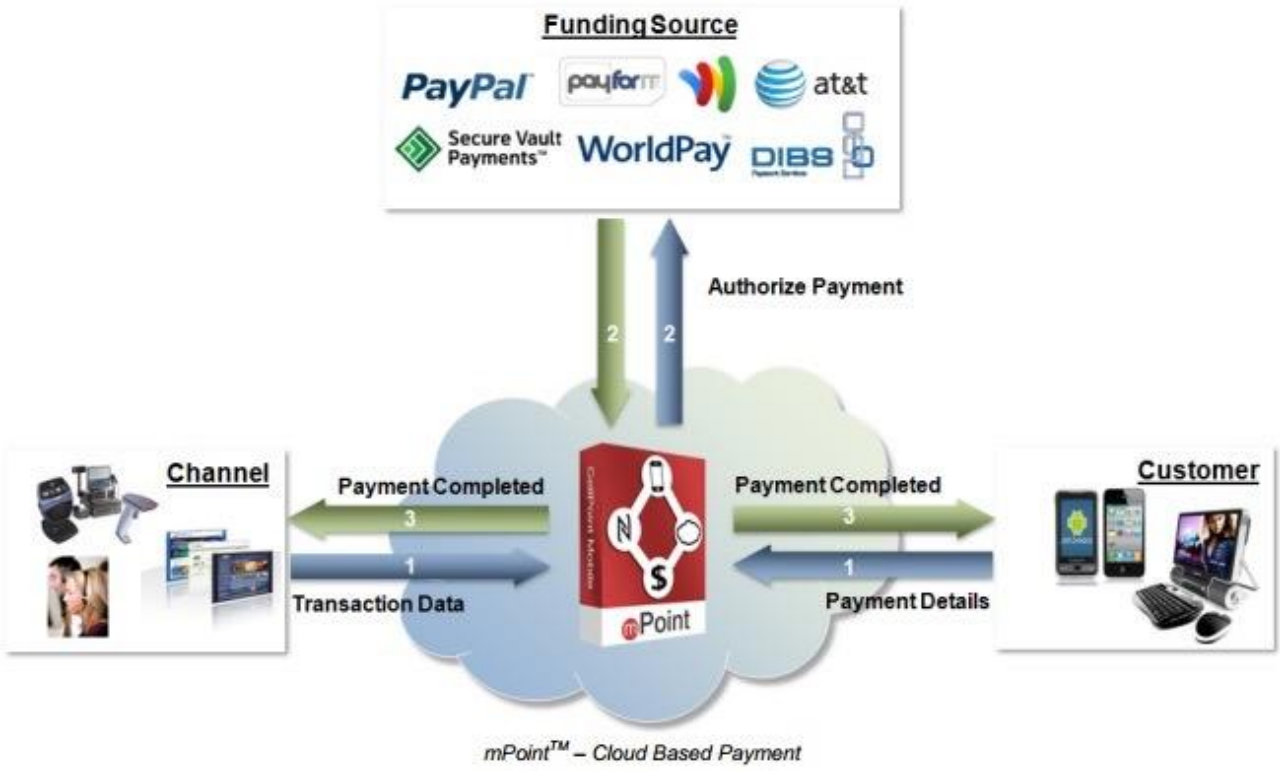


Figure 6. mPoint system schema **[1]**

mPoint comes with and SDK for iOS and Android to simplify integration into the merchant`s mobile application. mPoint Solution is very well equipped to process mobile payments, however it was not designed for Near Field Communication payments and requires two steps. NFC payments needs to be executed immediately, that is why mPoint Server will be expanded with new methods specifically design for that purpose.

mPoint SDK

As illustrated by **Błąd! Nie można odnaleźć źródła odwołania.**, the mPoint SDKs have been designed for asynchronous communication using the delegate pattern which requires the Merchant application to implement the methods defined by the provided mPoint interfaces:

The mPointDelegate interface for merchants using basic mPoint features set to process card based payments through one of the supported Payment Service Providers (PSP).

The mPointWalletDelegate interface for merchants using full feature set of mPoint to provide enhanced security and 1-click payment.

Diagram can be broken down into following logical parts:

- Payment Initialisation
- Payment Authorisation
- Save Card

mPoint SDK - Implementation

To use mPoint SDK Android developer needs to place the mpointsdk.jar file in the */libs* directory and add reference to the JAR file to the Java build path.

Payment Initialization

Initializing payment through mPoint SDK involves the following high-level tasks:

1. Invoke the initialize method on the SDK instance.
2. Handle the SDK callback through the *handleInitialize()* method.
3. Construct the GUI for enabling customers to enter their card details or select sorted card.

Payment Authorisation

Authorizing the payment using the mPoint SDK invokes the following high-level tasks:

1. Invoke one of the authorize methods on the SDK instance
2. Handle the SDK callback through the *displayPaymentConfirmation* method.
3. Construct the GUI for presenting customers with a payment confirmation screen.

Status Handling

Handling return status from mPoint server, will result in callbacks from the SDK to one of two methods that must be implemented by the application.

handleStatus is called by the SDK for “high-level” errors when the mPoint server rejects the request and returns one or more status codes giving the reason or for operations that does not return any data where mPoint server returns the operation status.

handleError is called for “low-level” errors such as network connectivity problems.

As one can see mPoint SDK is a very powerful library that speeds up process of creating wallet or mobile payment application. However it does not include methods that could speed up development of the Near Field Communication payment using Host-based Card Emulation that is possible since the release of Android 4.4 platform. This thesis will expand capabilities of mPoint SDK for Android to allow rapid development or fast and easy integration of NFC payments using HCE technology on supported devices.

Project

The goal of the project is to allow Wallet application to conduct NFC payments using HCE by extending mPoint SDK and mPoint server with needed methods.

Current capabilities of mPoint SDK and server allow for seamless mobile payments and are used in several mobile applications like Mobileperiodekort or DSB App. However current implementation have to be changed to process NFC payments. NFC payments have to be fast and secure to be competitive with Smart Cards payment.

mPoint has a big advantage over Smart Cards that merchants, which is stored funds payments. Smart Cards issuers like Visa or MasterCard get a small fee from each card usage. Card users are not aware of that since it is a merchant that has to handle that fee. Fee is usually a small percentage of transaction amount, but also has “not less than” amount. This “not less than” amount is really visible with small purchases. Huge stores all around the world are trying to find a way to get rid of that fees with different approaches. Most talked about is approach to make direct bank transfers from users to merchants that is being developed by CurrentC with support of big retailers. That, however requires each user to give access to bank account, that causes a lot of security risks. With Stored Funds merchant can get rid of that “not less than” fees using direct mPoint account to mPoint account money transfer. The fee is paid only when user uses credit card, that means that system can be modified to be most profitable for merchant and offer minimum amount of cash to be transferred to stored funds to minimize fees.

Stored Funds approach gives us 2 possible payment flows to be considered. Payment using Stored Funds and payment using stored Credit Card.

Payment Flows

Stored Funds Payment

Stored Funds Payment flow is the fastest possible scenario, since there is no need for authorization of the credit card information. This allows for seamless experience for user. Payment flows is illustrated on Figure 7. Payment Flow using Stored Funds

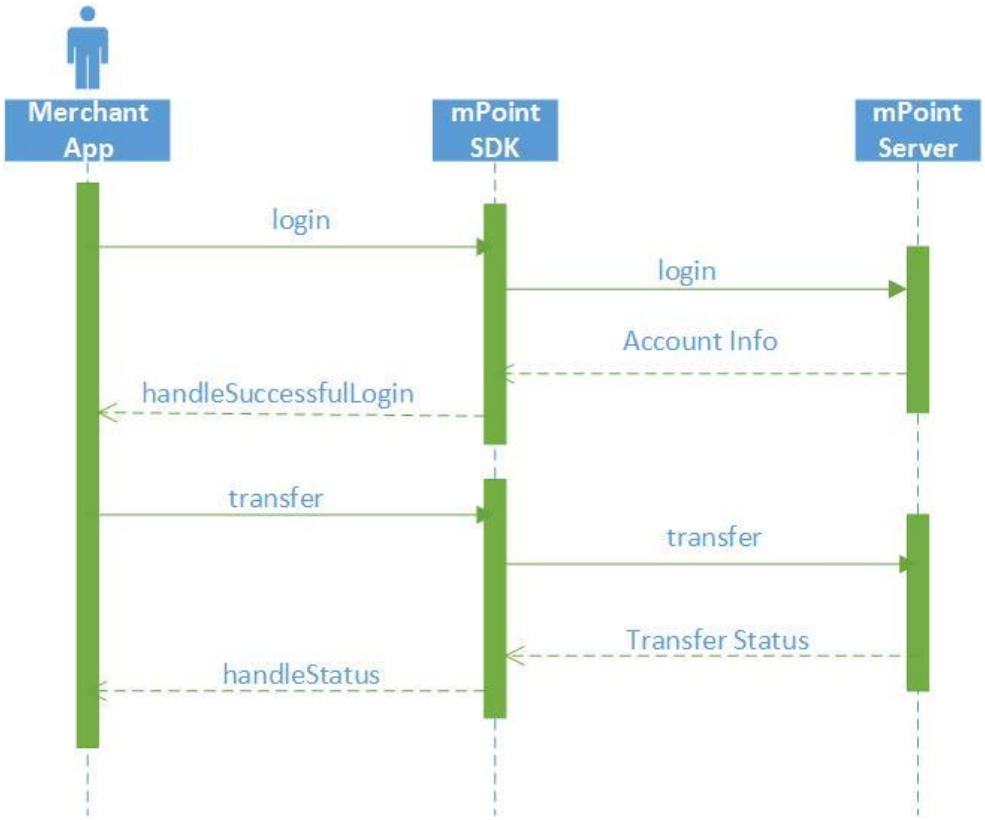


Figure 7. Payment Flow using Stored Funds

Credit Card Payment

Payment using stored credit card is a little slower and involves request to Payment Service Provider like DIBS, WorldPay or PayPal for card authorization, adding one step to the flow. This payment flow is described on Figure 8. Payment Flow using stored Credit Card

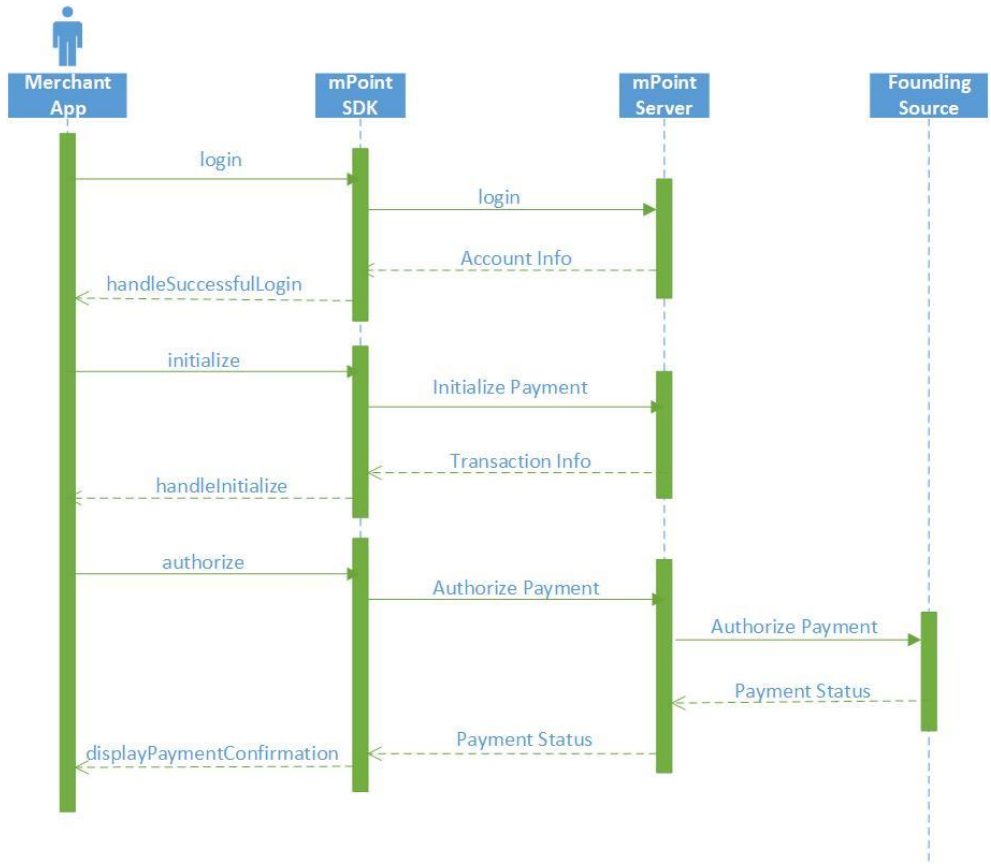


Figure 8. Payment Flow using stored Credit Card

Approach 1

This paragraph describes first approach that was taken to solve contactless mobile payments using mPoint payment gateway. The main idea was to secure the user that their card/mPoint account data will be guarded at all time. Android with HCE gives the first step of security, only transmitting data when the screen of device is activated. Second step of security is possibility to force user to unlock the screen before the payment is made by setting *android:requireDeviceUnlock* to true in the resource files that holds list of recognizable AIDs.

The third step of security is designed to guard the data while it is being transmitted to the NFC receiver. The OneTimePassword that can only be used

once for payment authorization and prevent skimming of the complete user info that could potentially lead to duplicating the user account and performing payments.

The idea is to introduce OneTimePasswords that are generated on the server and pushed to users device. There are two OneTimePasswords generated for a user and each one can be used to authorize one payment. OneTimePassword is verified on the server and if matching par is found payment can be processed. Next step is generation of new OneTimePassword and pushing it to user device.

OneTimePassword requires changes to whole payment channel starting from the server, through database and the SDKs. Server needs expansion with new methods that can generate, validate, save and push OneTimePasswords. Also DataBase needs extension with new table for storing the passwords. Next all the SDKs would need to implement new payment schema and extensions to handle saving and retrieving OneTimePasswords needs to be added.

Another security measure is assured by only verified merchants being able to use contactless payment approach. That means that even if you card/account data would be stolen, the “copied” account would only be useful in verified sell points and cannot be used to make money transfer to any other account.

Last security measure and suggestion from CellPointMobile CTO Jonatan Evald Buus lead to leaving OneTimePassword implementation half way. It was pointed out that introduction of OneTimePassword is just a minor security upgrade and does not offer greater security than transferring user password. Meaning that on successful skimming attack user`s password or user`s OneTimePassword is stolen and can be used to make transaction. Main point is that you can empty user`s credit card only once, meaning that it does not matter which password would be used to do that. That lead to changing the payment approach to Approach 2.

Approach 2

As discussed in previous paragraph there are several security measures starting from Android`s device screen on, to only verified merchants being able to complete transfers. That lead to changing approach and putting the password security measure in hands of the user. People are used to making payments using credit cards, which requires entering PIN code or signing the receipt. That is why new flow was introduced based on the user input. Change in a new flow requires user to manually input password on the merchants phone or decide that they trust that system is secure and allow for transferring it via NFC during payment.

This change, adds a new (optional) step to the payment flow. Complete payment flow is illustrated on Figure 9. Complete Payment Flow .

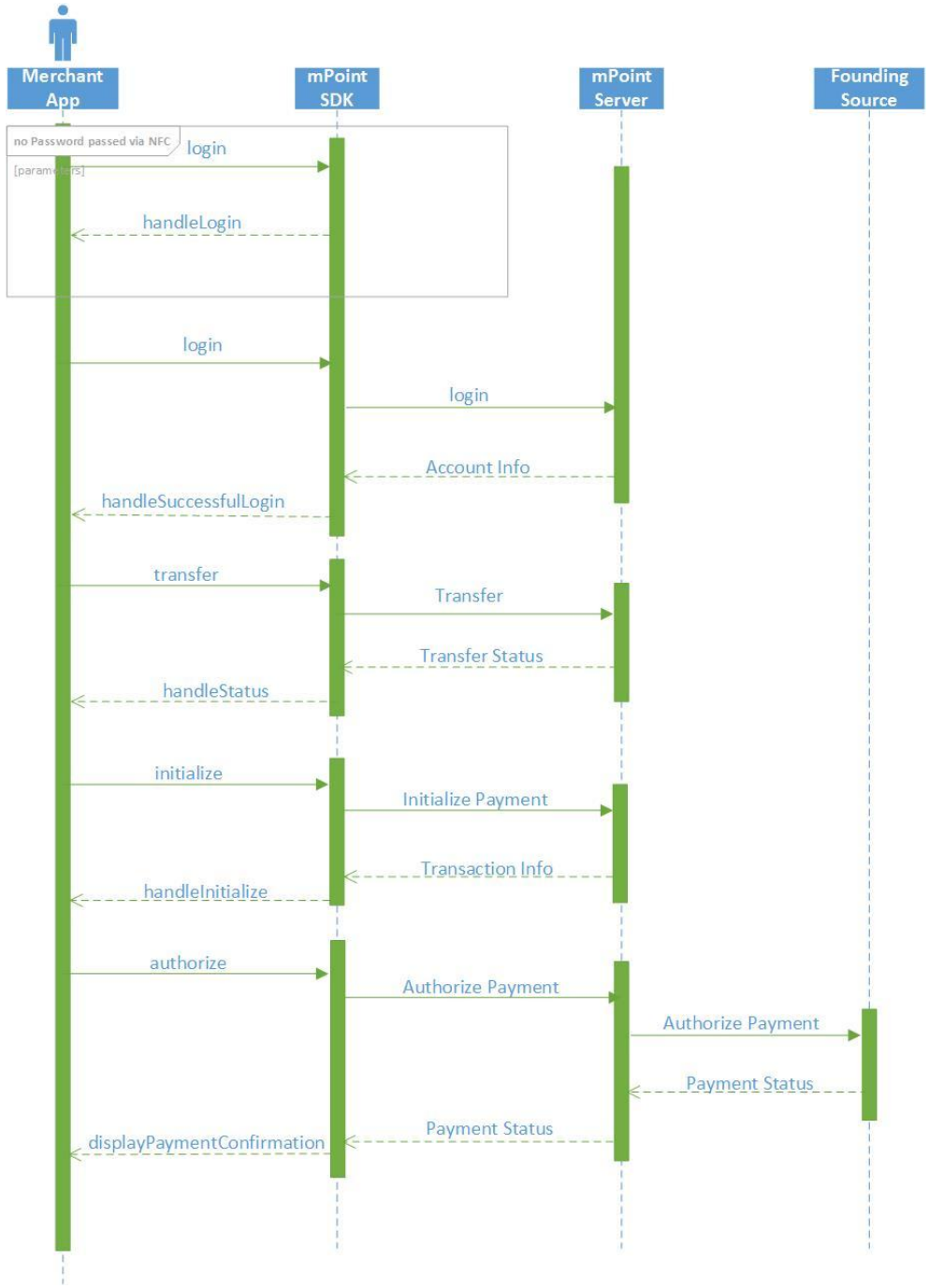


Figure 9. Complete Payment Flow

mPoint SDK

Despite mPoint SDK`s powerful and robust set of methods it lacks ability to support NFC payments. Work done for mPoint SDK includes creating methods to allow developers to easily implement NFC payments into their application.

Work can be separated into two main categories :

- card emulation - methods supporting host card emulation
- card detection - methods supporting detection of specific card
- utility methods - methods that are considered helpers

Card Emulation

As described in HCE Service - Implementation there are several steps that developer needs to implement to support HCE. This part describes all the methods that were incorporated into mPoint SDK to support HCE and take control of Application Protocol Data Units that can be recognised by system.

New method *byte[] contactlessPayment(byte[] commandApdu, Bundle extras, mPointClientInfo clientInfo, String oneTimePassword)* was introduced to process the received APDU and return appropriate response upon verification of received data. This method also handles all the data that is being returned together with response APDU. Implemented method also takes instance of mPointClientInfo and client one time password for authentication on server.

Due to change in approach OneTimePassword is change with String password field to allow for seamless payments. The new method is byte[] *contactlessPayment(byte[] commandApdu, Bundle extras, mPointClientInfo clientInfo, String Password)* and can be called using mPoint instance.

This method is supposed to be called from the class that is extending HostApduService and implements *processCommandApdu()* and *onDeactivated()* methods. *contactlessPayment()* methods processes all the information needed by *processCommandApdu()* methods and returns byte[] with prepared response.

contactlessPayment() method

This method is designed to handle card emulation. Calling this method requires following inputs:

- byte[] commandApdu
- Bundle extras
- mPointClientInfo clientInfo
- String Password

This method handles creation of *byte[]* holding information needed to perform contactless payment. First thing that happens in the method is check if the appropriate APDU received matches SELECT_APDU command for the service by comparing arrays:

Arrays.equals(SELECT_APDU, commandApdu) == true

If that statement is correct, clientInfo and password received are combined to create one string that is than translated to *byte[]* and concatenate with *byte[]* holding "OK" status word. Such created *byte[]* is then returned to the application as a response. Application sends *byte[]* as response to NFC reader (another mobile phone in this case).

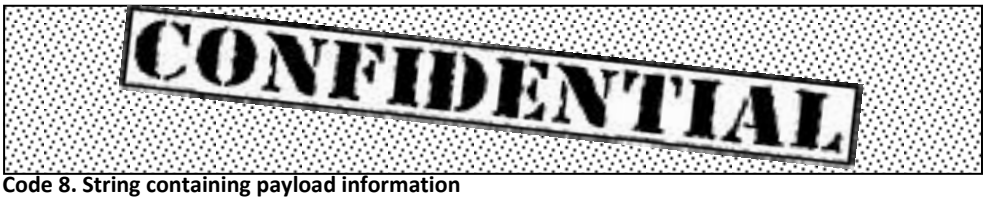
Card Detection

As mentioned in HCE Service - Implementation there are certain methods that needs to be implemented to detect NFC card. This paragraph describes methods incorporated into mPoint SDK that allows easy usage of HCE in the project.

New method *onTagDiscovered(Tag tag)* was implemented to handle discovery of another phone trying to communicate via NFC. This method uses *isoDep.connect()* method to connect with remote NFC device. Next the *byte[]* consisting of ISO DEP command HEADER (00A40400) and CARD AID is constructed using utility method *BuildSelectApdu(String aid).* Such prepared command is send to connected device and waits for response. If AID was successfully selected 0x9000 is sent as last two bytes of the response. Everything before that is payload used to transmit data needed to perform payments. In this project data transmitted via NFC consists of mPointClientInfo and String containing OneTimePassword. After approach change OneTimePassword is changed to user password. When data is received merchant application combines received data with merchant specific information and performs a call to transfer() method implemented in mPoint SDK.

login()

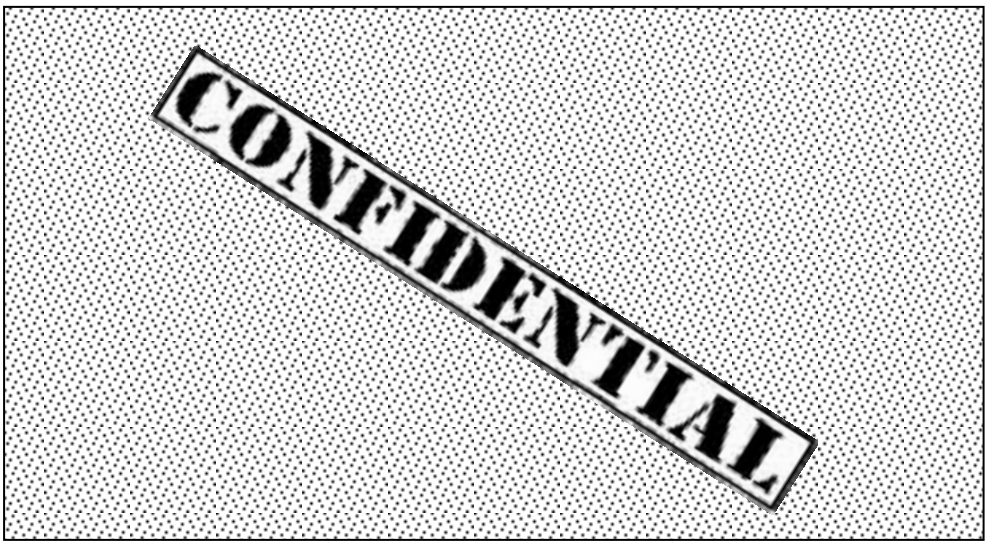
Login method was extended to handle data in *byte[]* format. New *login(byte[] data)* method is added to work with data transferred via NFC directly. Method takes last 2 bytes form the array and performs a check is new *byte[]* statusWord consisting of last 2 bytes from transferred data is equal to SELECT_OK_SW command. SELECT_OK_SW is a *byte[]* consisting of 2 elements (byte)0x90 and (byte)0x00, the "OK" status word. Next the utility method *ByteArrayToHexString(byte[] bytes)* is called and remaining *byte[] data* is translated to String value. *byte[] data* translated to String value is shown on Code 8. String containing payload information



Code 8. String containing payload information

Information received from payload can be used to perform a *login()* request to verify, that account exist, and receive security token for transaction as well as stored funds amount.

login() method creates XML combining passed information and performs a request to mPoint Server. Example of such XML is shown below.



Code 9. Login XML

Login response XML contains information about the account that is trying to perform a payment. mPoint method *processResponse()* is called and redirects data to *handleSuccessfulLogin()* method that has to be implemented by developer in the application delegate.

transfer()

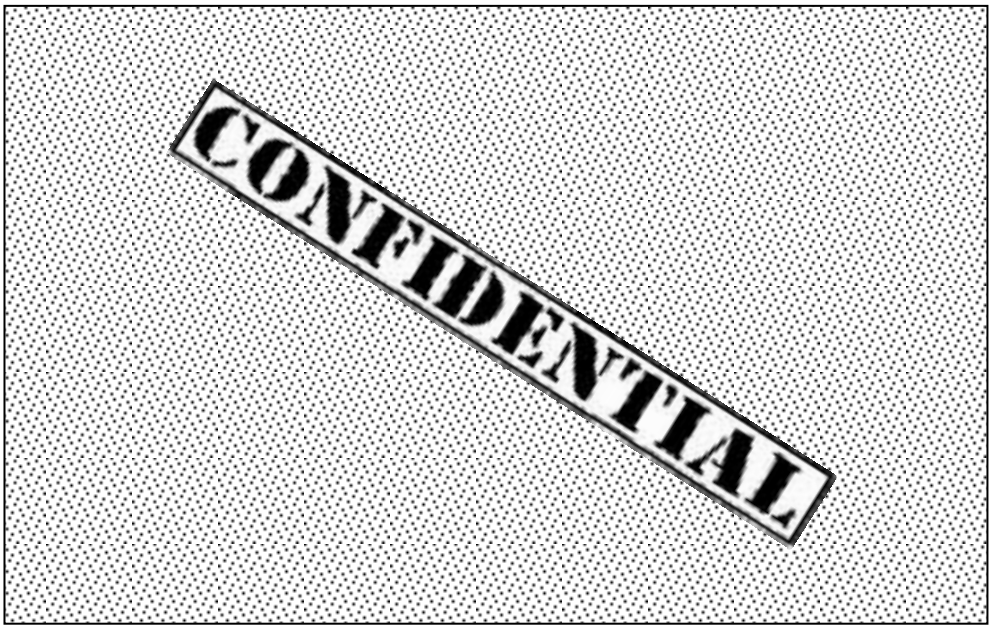
transfer() was extended to handle NFC transfer seamlessly. Calling this method requires following input information:

- *byte[] payload* – byte[] containing client information and password passed via NFC
- *int amount* – amount to be paid
- *int countryId* – id of the country for currency recognition
- *int operator* – network operator
- *long mobile* – merchant`s phone number
- *String password* – merchant`s password
- *String message* – message to be displayed

- *mPointClientInfo merchantInfo* – merchant information

This method takes all the information about merchant account like countryId, operatorId, mobile number, merchant password and merchantInfo (containing merchant deviceId) together with *byte[] payload* containing data transferred from user`s device, and also amount that needs to be transferred. The *byte[] payload* is translated to String value using utility methods.

transfer() method creates XML to be sent to the mPoint Server. XML example is shown on Code 10. XML created by transfer() method



Code 10. XML created by transfer() method

mPoint *processResponse(Object response, Client client)* method handles redirection of all responses from mPoint Server. If response is received from transfer path than it is redirected to delegate method *_handleStatus(ArrayMap<RecordMap<String, Object> >statuses, Client client, mPoint mpoint)* that has to be implemented in the merchant application to display successful payment confirmation.

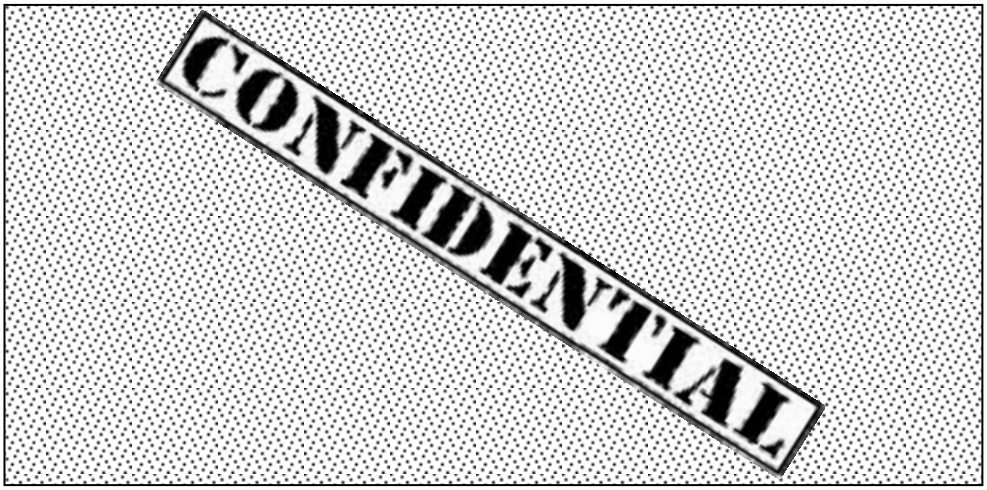
initialize()

initialize() method is called immediately after determining that stored funds are not sufficient to perform a transfer. This method begins the payment process, creating a trace record in the *transaction_tbl* in database. *initialize()* method takes following data as input:

- *int amount* – amount to be paid
- *int country* – unique country ID to define currency of the payment

- *int reward* – number of loyalty point to be rewarded for the payment
- *int operator* – unique mobile network operator ID
- *long mobile* – user`s mobile number
- *String email* – user`s e-mail address
- *LANGUAGE language* – language to be used by mPoint in any following communication

Method transforms input data into nice XML form that is sent to mPoint server as a request. Example of such request is displayed below :



Code 11. Initialize XML

Data returned by the server are passed to *handleInitialize()* method that has to be implemented in the delegate class by developer.

authorize()

This method is called directly after receiving response data from initialize() call. This method takes following data as input :

- *mPointStoredCardInfo card* – instance of the stored card
- *String pwd* – user`s password

Authorize method creates the XML format data to be send to mPoint server. XML example is shown below.

Code 12. Authorize XML

Utility methods

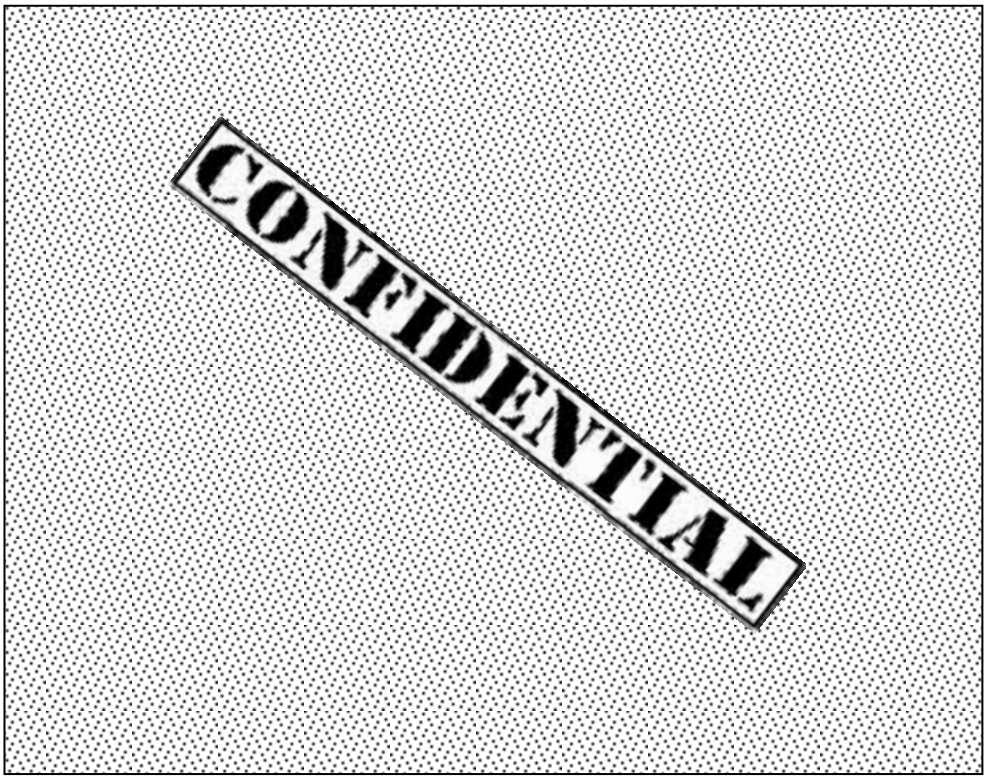
There are several utility methods that are use by both card emulation and detection:

- *byte[] ConcatArrays(byte[] first, byte[]... rest)* - method to concatenate two byte arrays. As processComandApdu() returns byte[] method that concatenate byte[] containing message with byte[] containing "OK" message (0x9000) is needed.
- *byte[] BuildSelectApdu(String aid)* - Build APDU for SELECT AID command. This command indicates which service a reader is interested in communicating with. This method converts String containing AID to byte[] using HexStringToByteArray(String s).
- *String ByteArrayToHexString(byte[] bytes)* - method to convert a byte array to a hexadecimal string
- *byte[] HexStringToByteArray(String s*) - method to convert a hexadecimal string to a byte string

mPoint Server

Login

Login function performs numerous checks and authorizations to verify that all received data are correct. It begins with authorization of the merchant. If merchant was found in the database servers looks for user data for authorization. Method then issues the security token and creates XML containing User Account Details. Example of the XML containing User Account Details is shown below.



Code 13. Response from successful login

There are several status codes that can be returned. The complete list of status codes can be found in Error Codes Paragraph.

Transfer

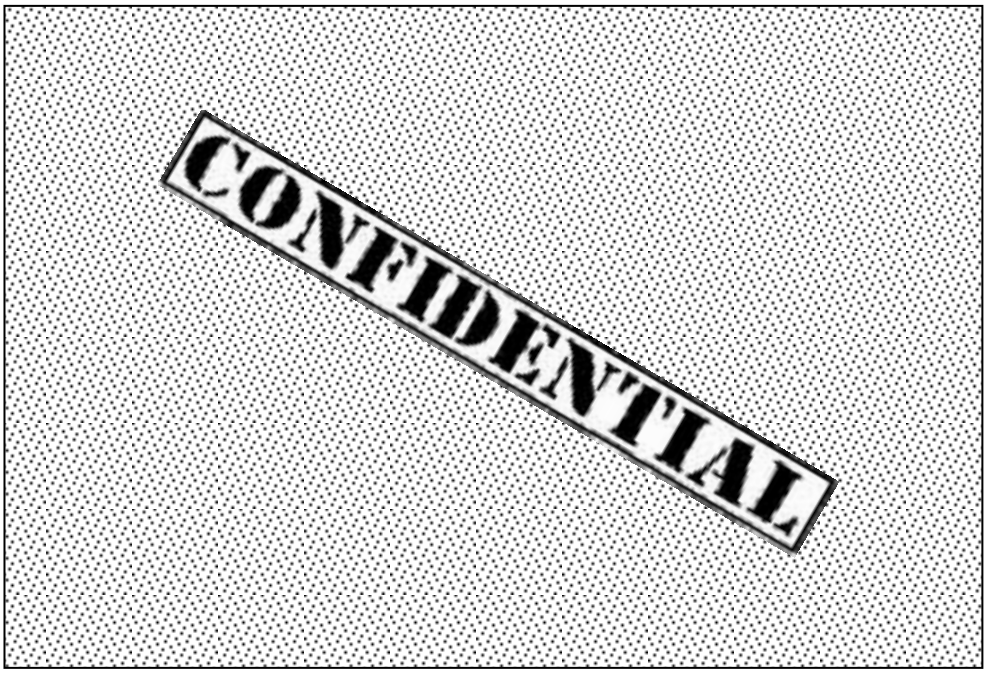
Transfer server method begins with authorizing the merchant account. If account was found in database new transfer is initialized. It performs numerous checks to verify existence of user and merchant information in the system. At the end it compares the security token for validation. There are additional checks performed to compare currency of the money being transferred and merchant account currency settings for money conversion purposes. When every check succeeds – successful response is sent. Example of such response is presented below:

Code 14. Successful response from transfer

There are several status codes that can be returned. The complete list of status codes can be found in Error Codes Paragraph.

Initialize

This method is used to initialize transaction. After receiving XML data from initialize() request it performs usual merchant and user validation against the database and creates the transaction trace in transaction_log table. Method returns XML formatted data to the delegate. Example of initialize return data can be seen below.



Code 15. Response from initialize

There are several status codes that can be returned. The complete list of status codes can be found in Error Codes Paragraph.

Authorize

Authorize method is called on successful initialize response received. This method receives request for credit card payment authorization. It uses transaction identification number (transaction id) from initialize response to verify which transaction should be performed. It verifies that transaction status is pending and changes it to complete after payment authorization is received from external system. Example of XML data returned by this method is displayed below.

Code 16. Succesful Authorize Status Response

There are several status codes that can be returned. The complete list of status codes can be found in Error Codes Paragraph.

Integration

This paragraph describes methods that developer need to implement to add HCE payments using mPointSDK.

processCommandApdu()

This method has to be implemented in the user application to send data of the emulated credit card. *processCommandApdu()* is called when user device is placed in proximity of reader device. This method should make a call to newly implemented *contaclessPayment()* method to handle card emulation.

OnTagDiscovered()

This method is implemented in merchant application and is called by the OS whenever appropriate NFC Tag is discovered. It is used to create a connection between two NFC enabled devices with *isoDep.connect()* method. On successful connection *isoDep().transceive(command)* method is called and *byte[] return* from the user`s device is used to perform *mpoint.login(byte[] result).*

handleSuccessfulLogin()

Method called when successful login request was performed. User specific information are returned from the server. Data from Login Response are used to determine if user has sufficient Stored Funds to perform a transfer or should Credit Card be used. Two different calls can be performed form this method:

- *transfer()* – payment with Stored Funds
- *initialize()* – payment with Credit Card

Stored Funds amount is taken out from Login Response and compared to amount to be paid. If there are enough Stored Funds – *transfer()* method is called immediately, else *initialize()* method is called to perform credit card payment.

handleInitialization()

Method called on successful payment initialization. As soon as this method is called developer should perform *authorize()* request to perform a payment.

displayPaymentConfirmation()

Method called on successful payment authorization. Merchant application has to implement this method and display confirmation screen with receipt.

handleStatus()

Method has to be implemented in merchant application whenever any of the methods responds with a status message only. Following code snippet shows how to differentiate between different operations.

Code 17. Code snippet to differentiate between operations

handleError()

This method has to be implemented by merchant application to handle error messages, that can occur throughout the payment flow. This method has display appropriate information depending on error code received. List of Error codes can be found in Error Codes paragraph.

Performance

One of the issues encountered during project was the NFC chip placement. During project Samsung Galaxy S4 and HTC One devices were used and at the beginning they did not want to send data between each other. It turned out that one of the phones has NFC chip at the top of the device, while the other one has it at the bottom of the device. That problem should not occur while using the system with POS terminals.

Performance of the system is very promising. The average times for receiving response for specific requests are listed below.

- Login response – 0.2 sec
- Transfer – 0.15 sec
- Initialize – 1 sec
- Authorize – 2.4 sec

From above data one can observe that total time of performing a transfer of Stored Funds averages around 0.35 sec, while performing a payment with stored card takes more time and averages around 3.6 sec. That is caused by the need to request the authorization in the third-party company.

User Application

The goal of user application is to replace the credit/debit cards and loyalty cards from the wallet. It will be able to store multiple credit cards and loyalty cards from stores associated with the program. That will allow to decrease the overall number of plastic cards kept in the wallet and automatically associate the loyalty cards with the customer without the need to display extra card during payment process, but what`s most important it will increase the speed of transaction and users comfort.

Wallet app is really simple from the user perspective. After one time setup process user does not even have to open the app to proceed with the payment, the only requirement will be to unlock the mobile phone screen, so the system knows that it`s a legitimate payment attempt not a skimming attempt. Payment information will always be accessible, while the screen is unlocked, even during usage of different app or phone call.

User is able to conduct fast, contactless payments and check the history of transactions on the spot. User is also able to define the default card that he wants to use for contactless transactions

This paragraph will present the steps needed to set up user application to be able to conduct contactless payments.

During the first opening of the application customer is presented with the welcome screen (PICTURE NUMBER) displaying some key information about the security and key features of the app.

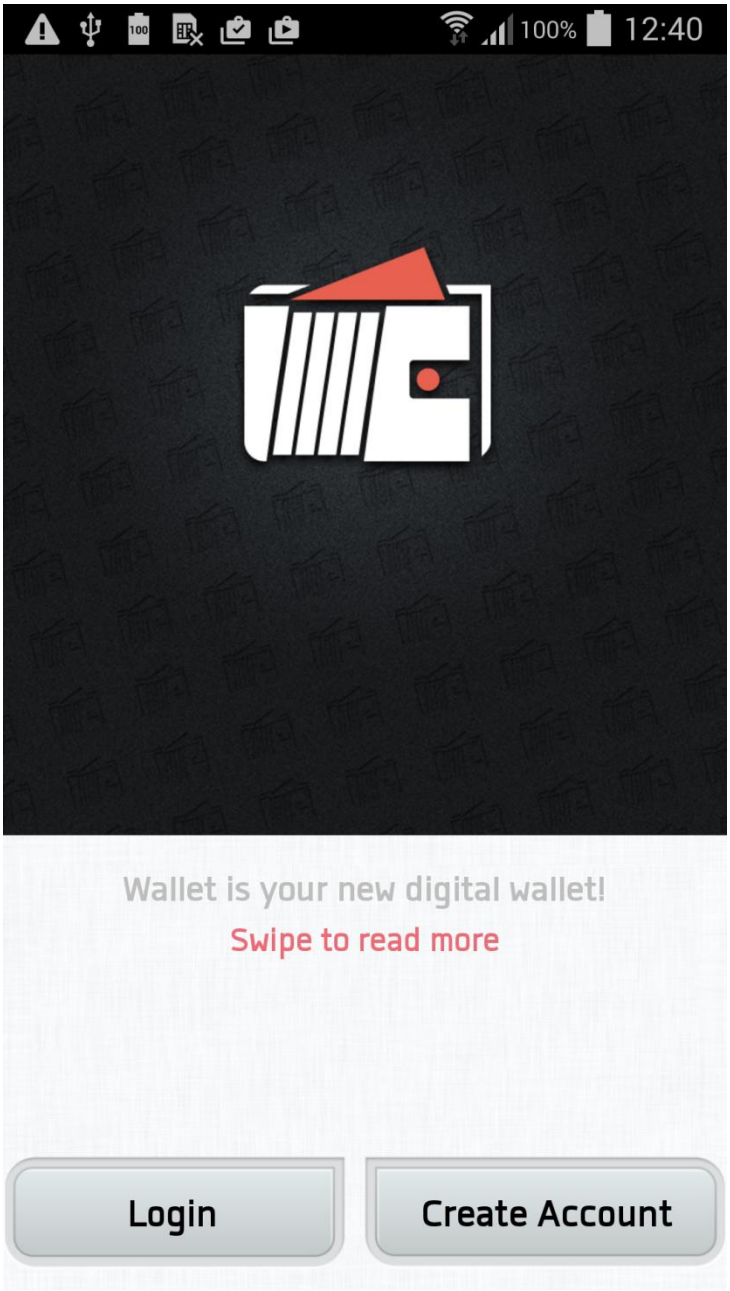


Figure 10. Welcome Screen

Here user can choose to login with existing account or create a new account if it is his first usage of the application. To create a new account user will need to enter key information on the following screens.

Pressing “Create Account” button takes user to the mobile details screen (PICTURE NUMBER). User is required to enter his phone number and e-mail address for verification.

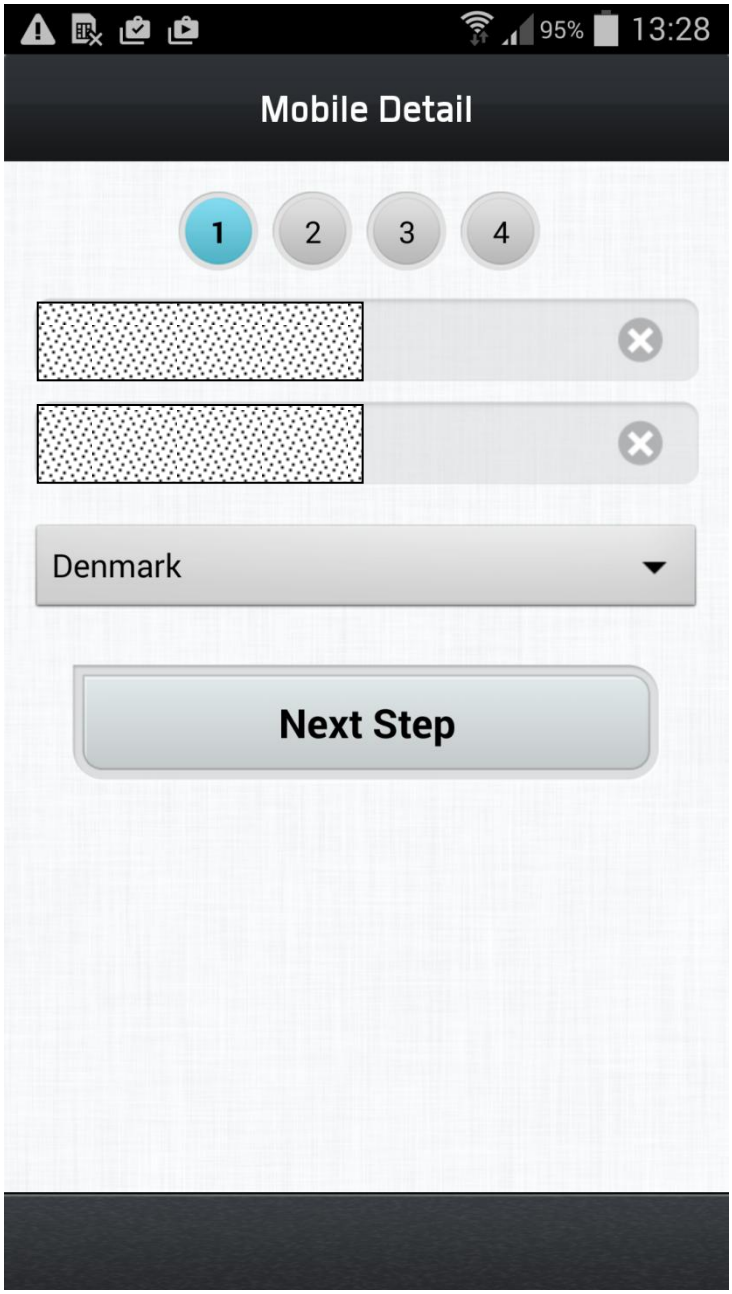


Figure 11. Phone Number/ Email input screen

On the next screen user is asked to fill the registration form (PICTURE NUMBER). It consist of two sections. First section contains the credit card information used to conduct payments. Second section contains personal information.

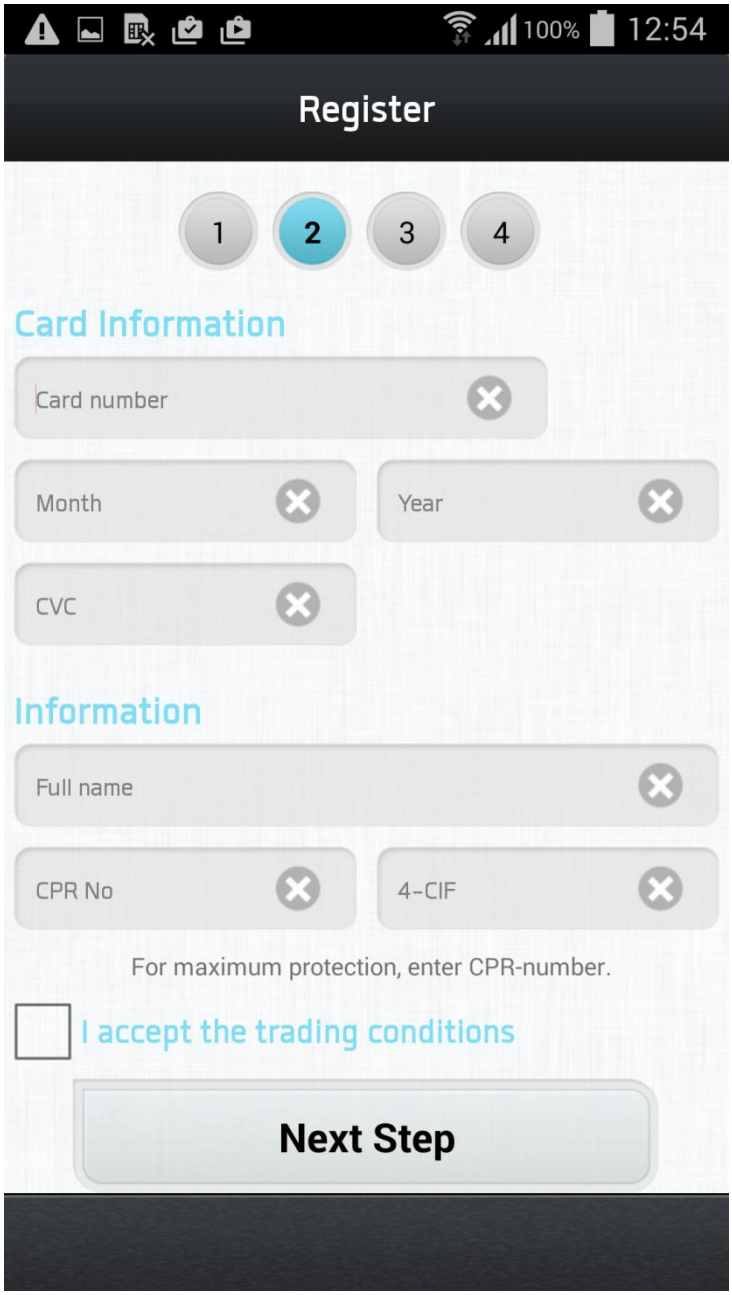


Figure 12. Account Information Registration Screen

After entering the required information (PICTURE NUMBER) the card is identified displaying correct (SMART WORD FOR VISA,DANKORT,MASTERCARD). User have to accept the trade conditions and “Next Step” button unlocks.

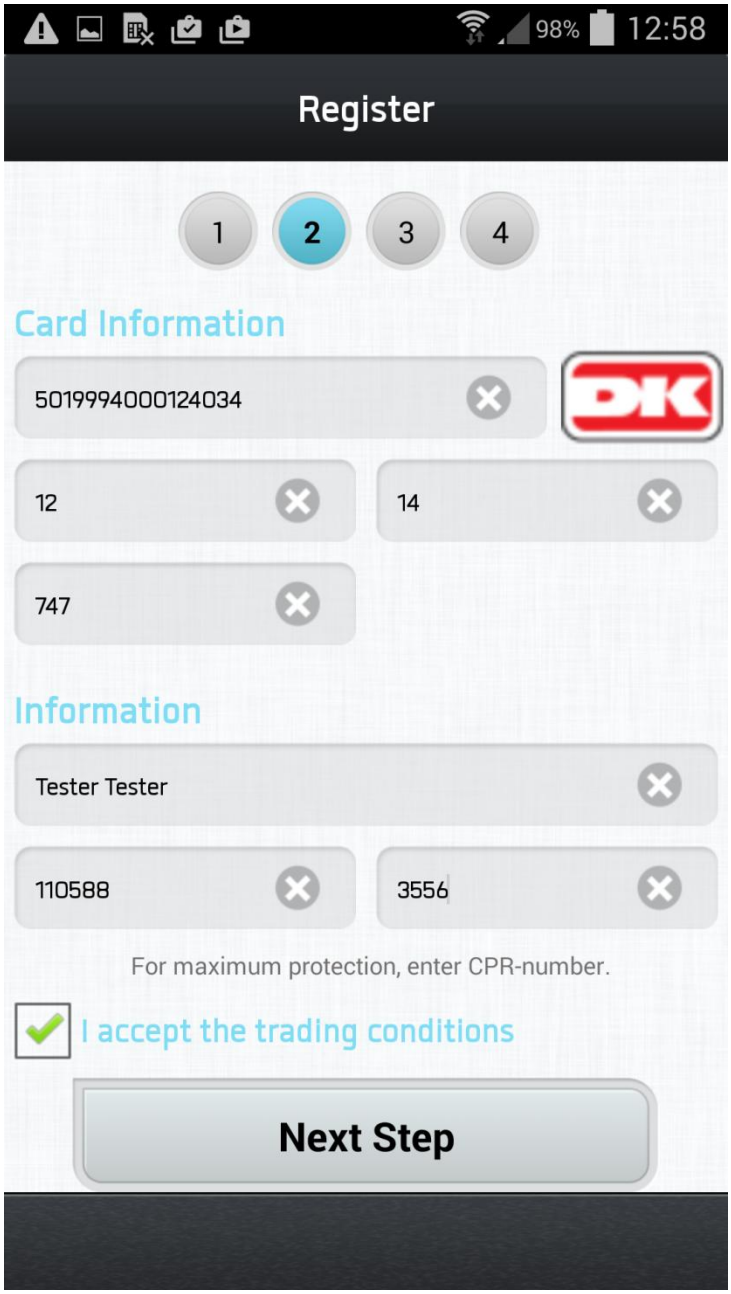


Figure 13.Example of filled Registration Screen

Third screen requires user to set up password that will be user to access the application and add the name for credit card.

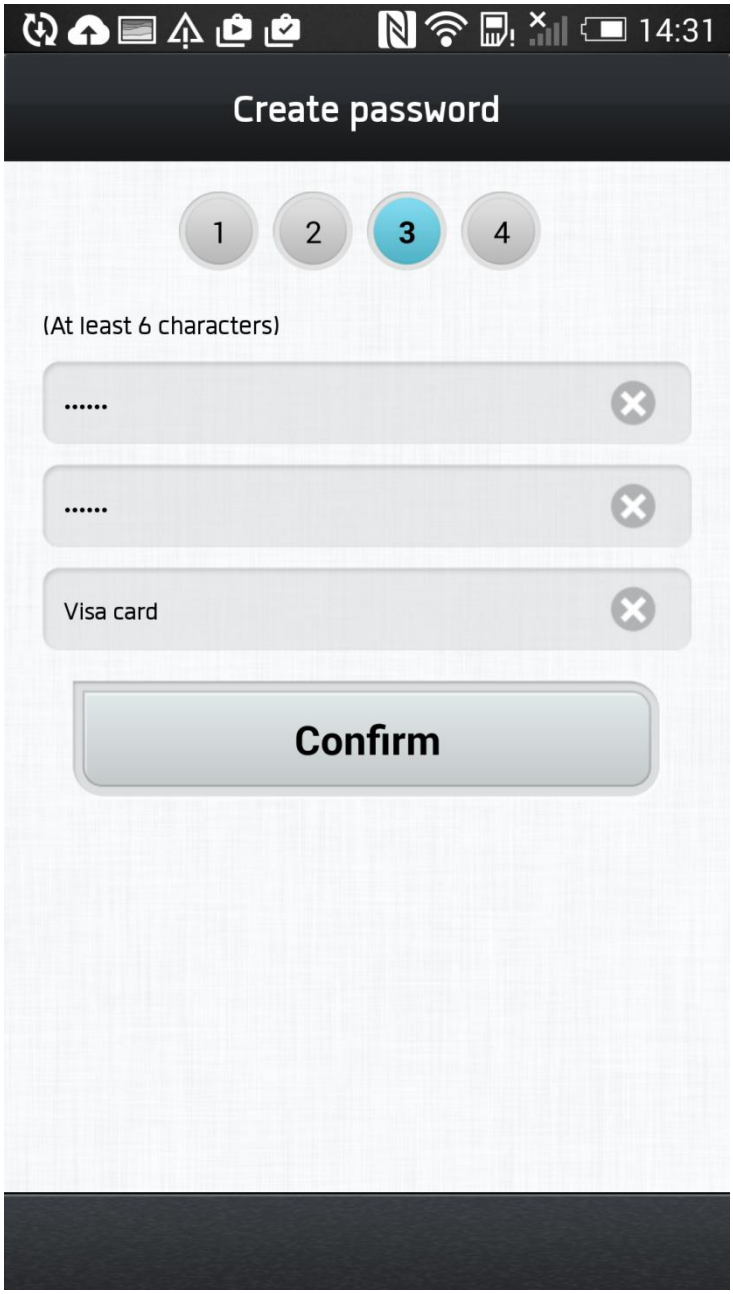


Figure 14. Password / Card Name Screen

Last screen requires user to verify validity of entered phone number. One Time Password is sent to the phone number specified by the user and is used to authenticate user`s phone number.

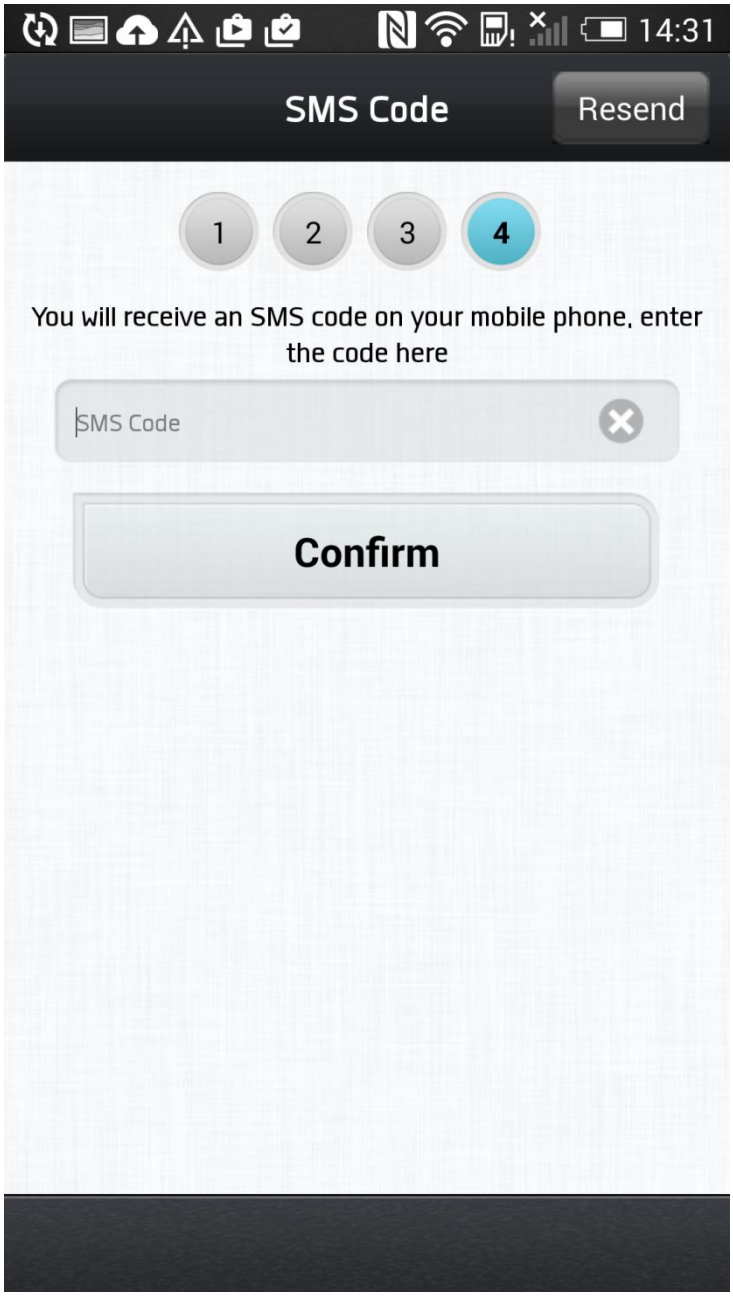


Figure 15. SMS Verification Screen

After successful registration process, user is displayed with virtual account balance and transaction history.

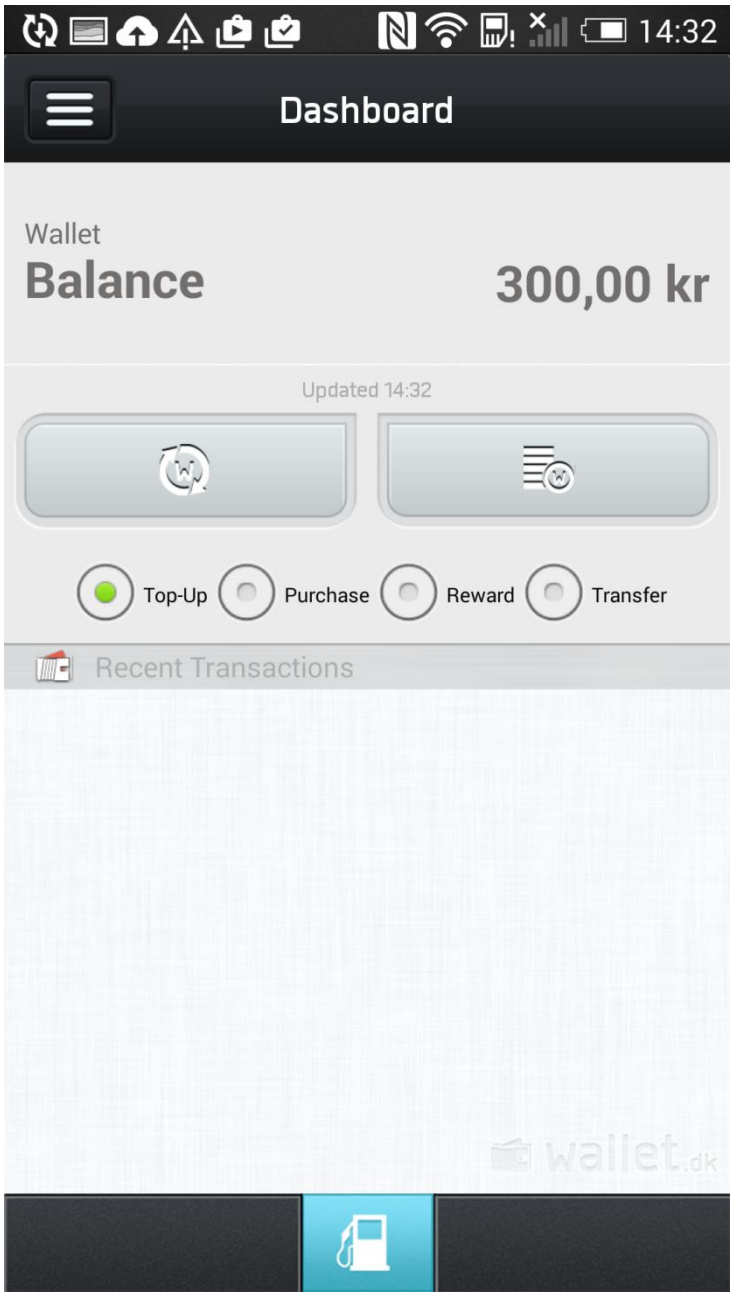


Figure 16. Balance Screen

Merchant Application

Merchant Application looks similar to the user application. After initial set up merchant application has one main display screen, showing input fields for transfer amount and transfer description.

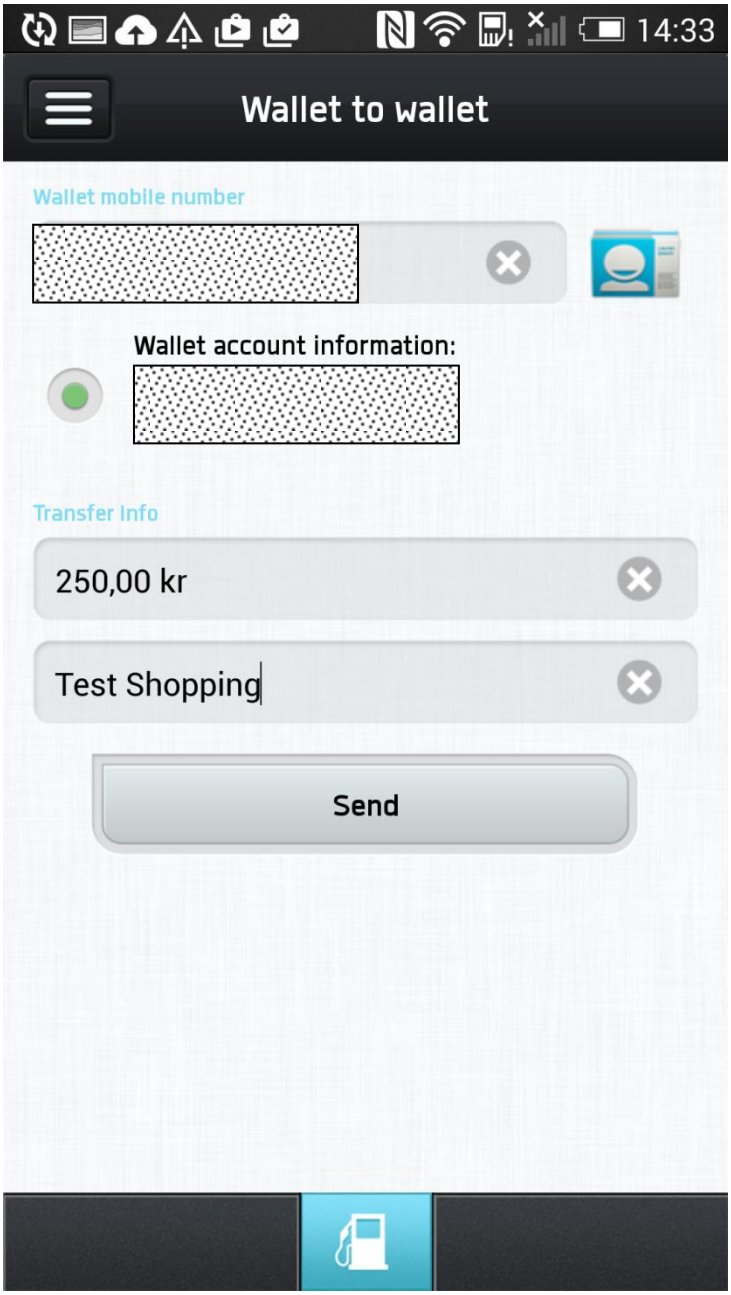


Figure 17. Transfer Screen

After successful transfer payment confirmation is displayed.

Figure 18. Payment Confirmation Screen

Discussion and Conclusions

In the thesis usage of mPoint payment gateway in NFC payments was discussed. It was used together with mPoint SDK to develop a virtual credit card payment system that could be implemented into any application. Wallet application was modified to work as a customer application (emulate the credit card usage) on one device and as merchant POS terminal on second device. This approach was chosen to showcase that mPoint can be integrated into any existing application on the market and work as a personalized retailer wallet ex. IRMA wallet, Coop wallet.

Android was chosen for an operating system of the devices for several reasons. Android is the most common mobile operating system in the world, working on more than 1 billion devices. Google support HCE technology from Android 4.4 (API 19) release, that allowed for performing this project. mPoint system comes with mPoint SDK for Android that helps with integration.

Problem approach was changed during the development, motivated by creating a complex system without increasing safety significantly. The additional safety – One Time Password – was abandoned, because it was proven not to offer enough security. Device security together with possibility of making payments only to registered merchants and transaction tracing offers enough surveillance over the data.

The main result of this thesis is that mPoint system can serve the purpose of NFC payments with modifications to handle NFC transferred data seamlessly. It proved to be fast during performing a payment and offer additional advantages for both potential users (merchant customers) as well as clients (merchants). It also allows for complete replacement of the physical credit card.

Extras

List of Figures

Figure 1. Simplified Communication Schema 8
Figure 2. System Overview 10
Figure 3. Tag Dispatch System [2] 12
Figure 4. Provisioning using physical Secure Element 16
Figure 5. Provisioning using Secure Element stored in the cloud - HCE 16
Figure 6. mPoint system schema [1] 20
Figure 8. Payment Flow using Stored Funds 24
Figure 9. Payment Flow using stored Credit Card 25
Figure 10. Complete Payment Flow 27
Figure 11. Welcome Screen 39
Figure 12. Phone Number/ Email input screen 40
Figure 13. Account Information Registration Screen 41
Figure 14.Example of filled Registration Screen 42
Figure 15. Password / Card Name Screen 43
Figure 16. SMS Verification Screen 44
Figure 17. Balance Screen 45
Figure 18. Transfer Screen 46
Figure 19. Payment Confirmation Screen 47

List of Tables

Table 1. Supported tag technology 13
Table 2. Login Status Codes 52
Table 3. Transfer Status Codes 53
Table 4. Initialize Status Codes 54
Table 5. Authorize Status Codes 55

List of Codes

Code 1. NFC features declarations [2] 14
Code 2. XML file listing nfc.tech technologies [2] 14
Code 3. <meta-data> element declaration in Manifest.xml [2] 15
Code 4. HostAdpuService Example [3] 18
Code 5. HostApduService declaration in Manifest.xml file example [3] 18
Code 6. Resource file defining AID group [3] 19
Code 7. NFC & HCE permissions [3] 19

Code 8. String containing payload information 30
Code 9. Login XML 30
Code 10. XML created by transfer() method 31
Code 11. Initialize XML 32
Code 12. Authorize XML 33
Code 13. Response from successful login 34
Code 14. Successful response from transfer 34
Code 15. Response from initialize 35
Code 16. Succesful Authorize Status Response 35
Code 17. Code snippet to differentiate between operations 37

References

[1] CellPoint Mobile SDK Documentation mPoint

[2] *NFC* - https://developer.android.com/guide/topics/connectivity/nfc/hce.html

[3] *HCE* - http://developer.android.com/guide/topics/connectivity/nfc/hce.html

[4] Tag Technologies - http://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.html

[5] http://www.smartcardalliance.org/publications-proximity-mobile-payments/

[6] http://www.smartcardalliance.org/wp-content/uploads/HCE-101-WP-FINAL-081114-clean.pdf

Appendixes

Error Codes

Login Status Codes

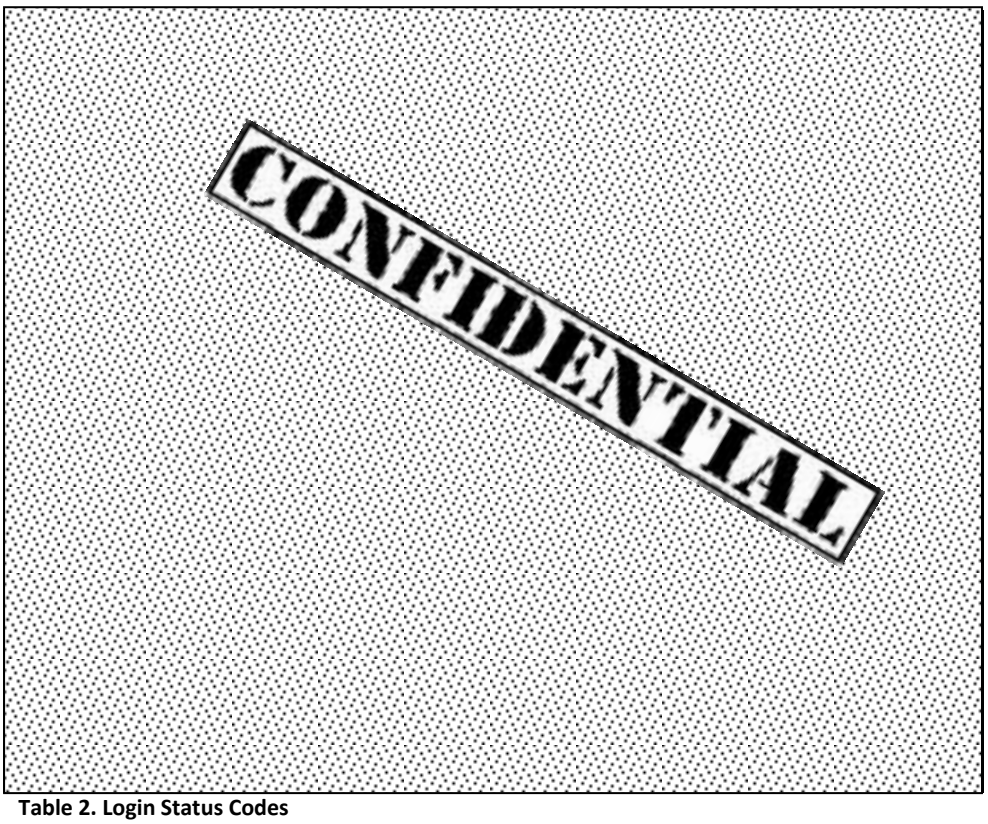
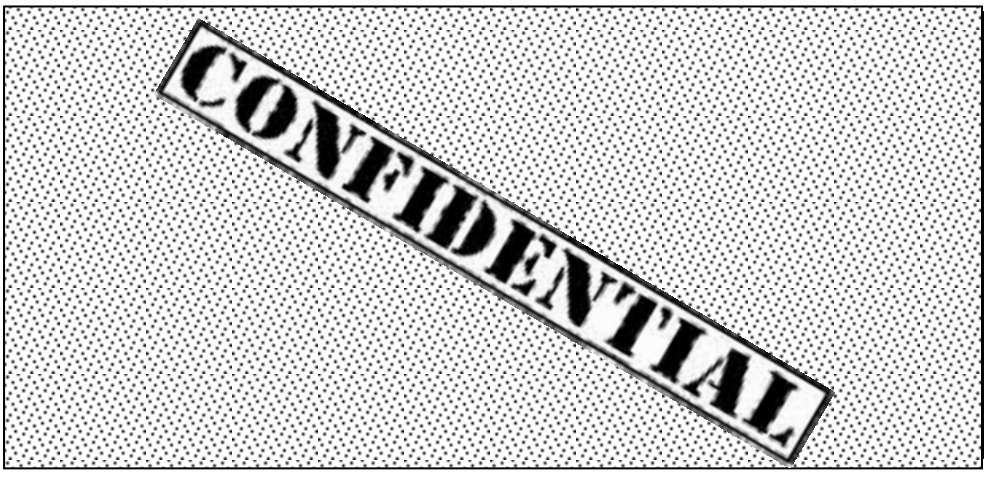


Table 2. Login Status Codes

Transfer Status Codes



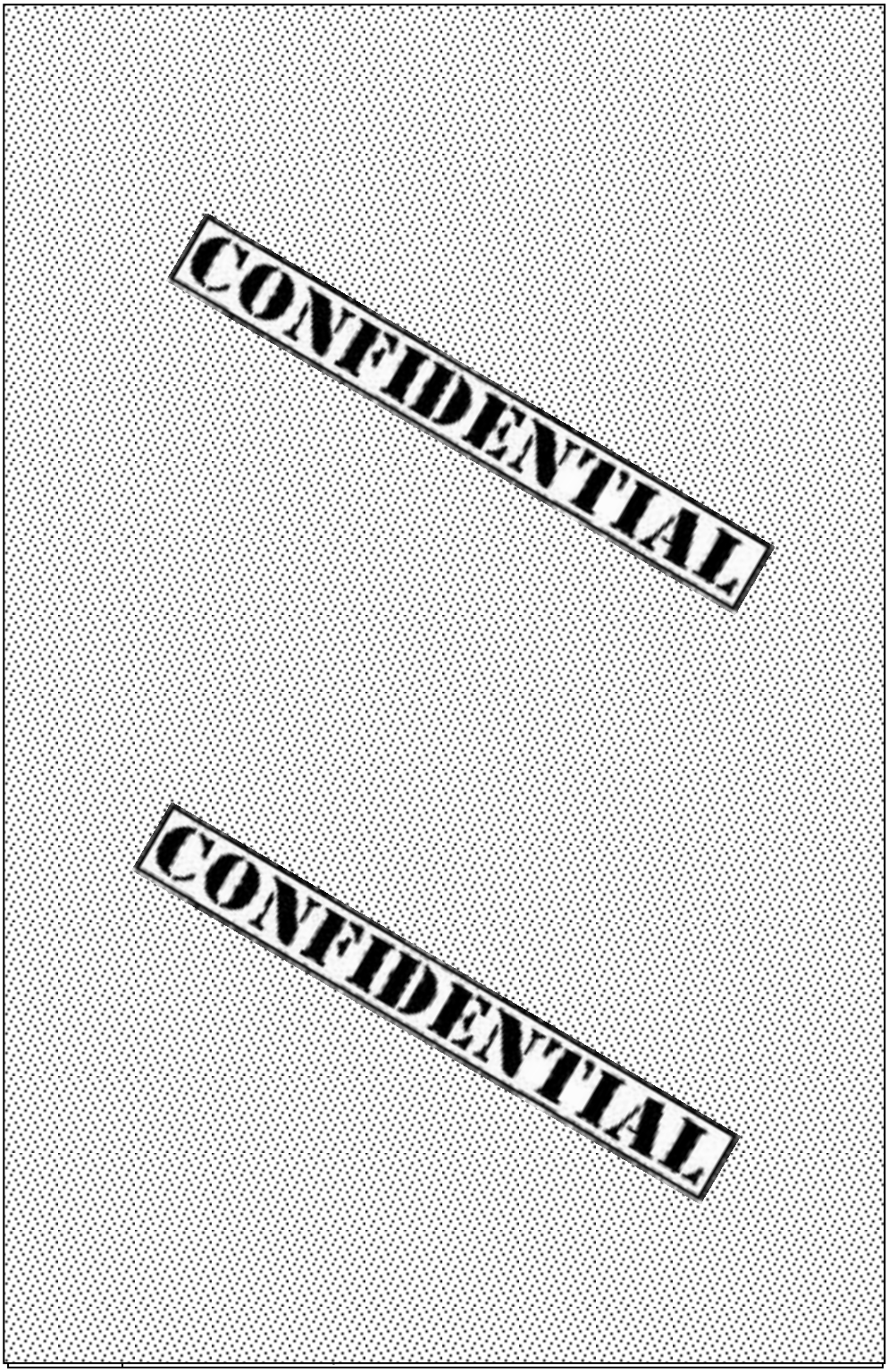


Table 3. Transfer Status Codes

Initialize Status Codes

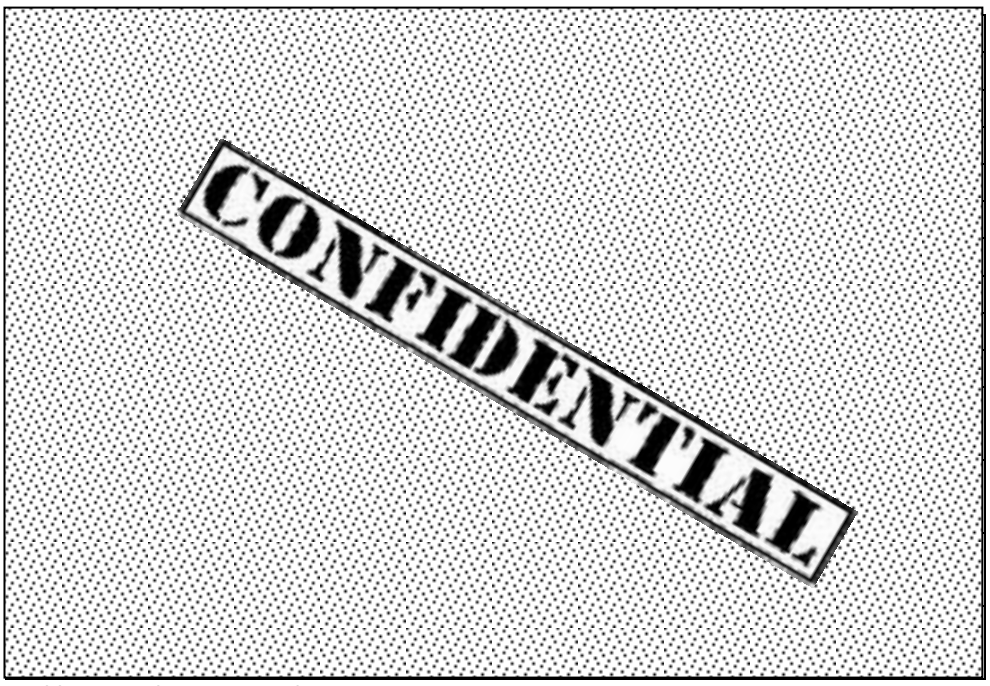


Table 4. Initialize Status Codes

Authorize Status Codes

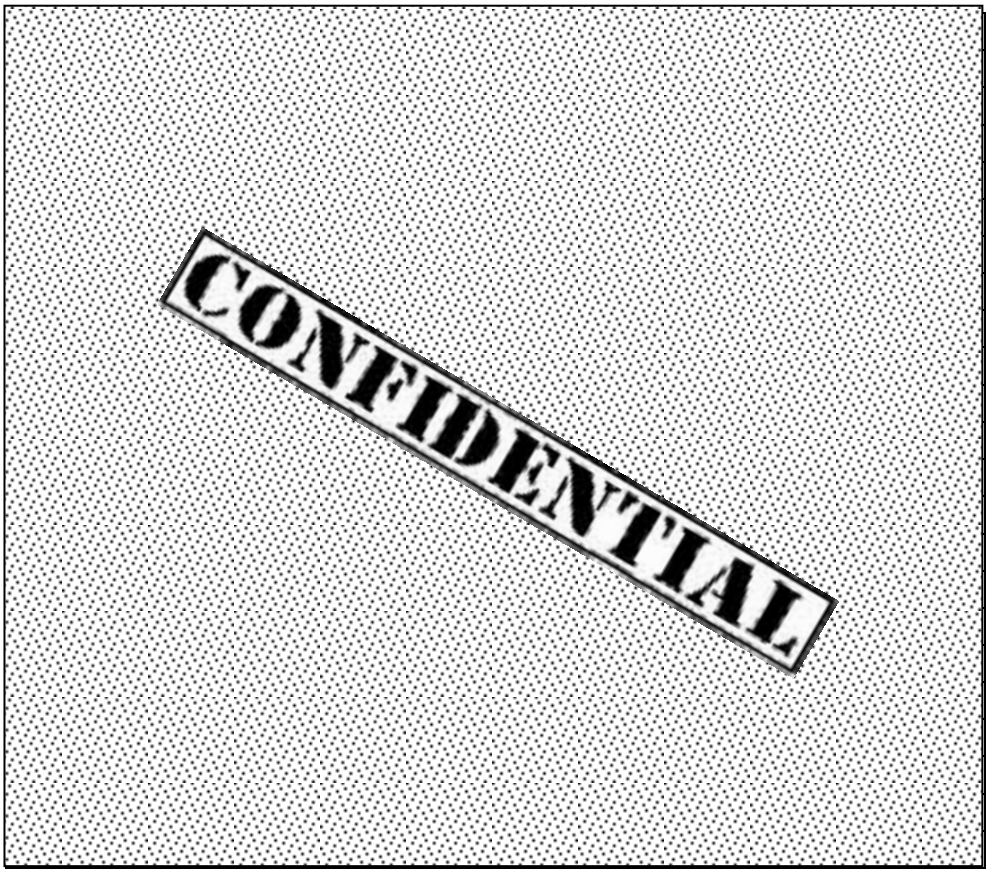


Table 5. Authorize Status Codes