ASSIGNMENT OF BACHELOR’S THESIS

Title:	FITCOIN: an Android light wallet
Student:	Minh Trieu Quang
Supervisor:	Mgr. Jan Starý, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2018/19

Instructions

Create the following parts of the FITCOIN crypto currency, as developed by Dvořák, Pajskr, Tománek in parallel works:

1. Descride how mobile wallets of cryptocurrencies obtain and verify transactions, not being full nodes.
2. Describe a simple protocol of such a communication for FITCOIN.
3. Implement an Android light wallet for FITCOIN that supports:
3a. maintaining balances on user's accounts.
3b. entering, signing and sending transactions.
4. Document your application.
5. Test you application.

References

Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. [https://bitcoin.org/bitcoin.pdf]

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 18, 2017

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

FITCOIN: an Android light wallet

Minh Trieu Quang

Department of Sotware Engineering
Supervisor: Mgr. Jan Starý, Ph.D.

May 13, 2018

Acknowledgements

I would like to thank my supervisor Mgr. Jan Starý, Ph.D. for his incredible guidance, patience and the support during the entire time on writing this thesis.

My thanks also goes to my dear friends Vojtěch Badalec and Tuan Do for introducing me to this amazingly interesting topic about cryptocurrencies and Laura Tichá and her mother for lending me an Android mobile phone to work with.

I would also like to give my special thanks to the *FITnam* organization which supported me by *de-stressing* me and making this thesis period more fun and enjoyable.

Last but not the least, I would like to give a big thanks to my family: my parents and my brother for supporting me throughout writing this thesis, not to mention through my studies on CTU FIT.

Declaration

In Prague on May 13, 2018 ………………….

Citation of this thesis

Trieu Quang, Minh. *FITCOIN: an Android light wallet.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

FITcoin je kryptoměna, která je paralelně vyvíjena dalšími studenty FIT ČVUT. Jedná se o triviální verzi Bitcoinu, která slouží ke studijním účelům pochopit její náležitosti a technologie, které tvoří základ každé kryptoměny.

Cílem této práce je popsat, jak obecně fungují mobilní peněženky, jakým způsobem získávají a ověřují transakce. Na základě této analýzy navrhnout a vytvořit mobilní peněženku v operačním systému Android a popsat komunikaci kryptoměny FITcoin. Zvolený problém byl vyřešený za pomocí Bloom filteru a metod kryptografie na bázi eliptických křivek. Výsledný prototyp je zdokumentovaný a důkladně otestovaný. V příloze práce lze nalézt aplikační balíček výsledného prototypu a snímky z jejího používání.

Klíčová slova mobilní peněženka v OS Android, analýza kryptoměny Bitcoin, simple verification payment, blockchain, lightweight client, transakce v kryptoměnách, komunikační protokol v peněženkách, Bloomův filtr, fitcoin

Abstract

FITcoin is a trivial cryptocurrency, which is developed by students of CTU FIT. It is a trivial form of Bitcoin which serves for study purposes to understand its requisites and technologies which form the basis of every cryptoccurency.

The goal of this thesis is to describe how a mobile wallet works, how it obtains and verifies the transactions. Based on this analysis, design and implement a mobile wallet in the operating system Android and describe a communication protocol for cryptocurrency FITcoin. The prototype is documented and thoroughly tested. The application package of the resulting prototype and screenshots of the usage can be found in the attachments.

Keywords mobile wallet in OS Android, general analysis of cryptocurrency Bitcoin, simple verification payment, blockchain, lightweight client, transactions in cryptocurrency, communication protocol of wallets, Bloom filter, fitcoin

Contents

List of Figures

List of Tables

Introduction

Nowadays, the number of smartphone users has been rapidly increasing. People use smartphones to communicate, search for information and mostly use them to simplify their life by having all the varied applications for their daily use. For instance, one of those application may be for financial organization which let them manipulate their bank account.

For the past few years, there has been a huge impact on our financial world in a form of cryptocurrencies. Mostly it is about *Bitcoin* but there are a lot of derived currencies. Since Bitcoin is open-source, many people are trying to improve this concept and find a balance between safety and speed.

A huge amount of people found their interest in these currencies and they started to invest in them and exchange them. It became so popular to the point where many services and shops are offering a possibility to pay with some of the most popular cryptocurrencies. Therefore these people will certainly start seeking an app to manage their assets. Since mobiles don't have such a huge storage as computers, it can only be used for sending and obtaining the transactions via a full node which is usually a computer. Thus, mobiles can only be a light node.

I have chosen this topic because in my humble opinion I think the concept of blockchain is a key to storing digital data which ensures safety. For example, Estonia has most of their government data in a form of blockchain.

In my thesis, I will analyze and describe the implementation of a mobile wallet for the trivial form of cryptocurrency *FITcoin* which is also being developed by CTU FIT students Karel Pajskr (*FITcoin*: peer-to-peer communication), Mikuláš Dvořák (a blockchain for *FITcoin*), Jan Tománek (*FITcoin*: trees of transactions) in parallel works and should aim to help people understand the current problematic.

My thesis is structured as follows. Firstly, I will describe what cryptocurrency is in general, its requisites and how it works. Secondly, how the mobile wallet communicates with the network, which will lead to its implementation and lastly the implementation of the mobile wallet itself.

Goal

The goal of the research part of the Bachelor thesis is to understand the current controversial problematic and grasp the basic terms of cryptocurrency, to analyze and design the mobile wallet for the trivial form of cryptocurrency *FITcoin* inspired by the *Bitcoin* itself, and how it communicates with the network.

The practical part will be concerned in creating a functional prototype of a mobile wallet in operating system *Android*, which will communicate with the *FITcoin* network. A user will be able to create, send, verify and receive transactions in a simple yet elegant *GUI*. The application will be tested and documented in the last chapter.

Chapter 1

Cryptocurrency

1.1 What is cryptocurrency?

Cryptocurrency (or *alternative coin*) is considered as a digital asset designed as a medium of exchange based on the usage of an *elliptic curve cryptography* to secure and control its transactions, creating them and verifying them. In this text, I will refer to these assets as *coins.*

Cryptocurrencies are decentralized using peer-to-peer technique so there is no central system controlling it, such as a central bank. It works completely independently through a distributed public transaction ledger called *blockchain.*

Cryptocurrencies are also known for their anonymity since these coins are tied rather to *keys* (or *addresses*) than people so the owners cannot be identified, but all transactions are being kept publicly [3].

1.2 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is a type of asymmetric cryptography over finite fields. Asymmetric cryptography, or *public key cryptography*, is a system that uses a pair of keys: a private and a public key and it is used to authenticate and encrypt messages. Primarily, it is deployed in a communication between parties, where everyone knows the public key but only the owner has his private key. Many algorithms of asymmetric cryptography have a set of mathematical constants, for example *domain parameters*, and require that everyone in the communication knows about them [4].

Mathematical operation in ECC is defined over an elliptic curve

$$y^2 = x^3 + ax + b.$$

With various a and b, the function gives a different elliptic curve. Every point (x, y), that solves this equation, lies on the elliptic curve.

ECC unlike the first-generation asymmetrical algorithms, has huge advantages. First, it uses shorter key. It means less computing power, generates less data and generates faster signatures and still get the same level of security [5]. Second, the public keys are generated from private keys and this is a fundamental feature in this problematic.

ECC can be used to generate *digital signatures.* Digital signature is a string that is produced by the private key applied to the data set we want to sign. The public key is then being used to *verify* this signature. This makes it useful, since everyone with the access to the public key can verify the transactions while only the owner can produce a valid signature with the private key. The digital signature algorithm based on elliptic curve is called *Elliptic Curve Digital Signature Algorithm* (ECDSA). ECDSA takes parameters $(curve, G, n)$, where *curve* is an elliptic curve, G is the base point of a prime order on the elliptic curve and n the order of G.

The randomness in signature is very important so one must put attention while generating the signatures. The signature algorithm uses a random key k and if the same value k is used twice, then the private key can be calculated [1].

The *curve* that is being used in Bitcoin is *secp256k1*, which is defined in [6]. This became popular thanks to its properties. Unlike many commonly-used curves, *secp256k1* has non-random structure which is very efficient for computation. The equation is

$$y^2 = x^3 + 7,$$

and is defined over finite field $\mathbb{F}_p$, where p is a prime order and is defined as $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

1.3 Keys, addresses

Ownership of coins is established through *digital keys* and *digital signatures.* The digital keys are stored in a *wallet* and generating them does not affect the cryptocurrency network. These keys come in a pair consisting of a public key and a private key.

There is an analogy to grasp this problem – *"Think of the public key as similar to a bank account number and the private key as similar to the secret PIN,... "* [1].

Private key is a picked number which we use in elliptic curve multiplication (one-way cryptographic function) to generate a public key. So the important part is to store private keys because public keys can be mathematically computed from them, however it does not work the other way.

Public key is cryptographically calculated point from the private key where the function is *irreversible.* The reverse problem is called "finding the discrete logarithm" which can be very difficult as trying all the possible values is very time-consuming.

Address is a derived form of public key made from a hash function. It can be shared with anyone and allows recipient to obtain coins.

Hash function is a one-way cryptographic function that produces a string called a *hash*. You can put any data in the hash function and it will output a hash that identifies those data. By changing a slightest bit in the data, the hash would also change. The hash is commonly used to verify the integrity of the data set.

1.4 Blockchain

Blockchain is a modern technology for digital assets that cryptographically secures the data, as the name says, in a chain of *blocks*. It is commonly visualized as a stack, hence the distance between the first block and the most recent one is referred as a *height* [1]. A container data structure *block* consists of two parts, a block header, which contains the hash of the previous block, a timestamp and information connected to mining, which will be discussed later. The second part of the block is a list of the transactions.

In the digital currency scheme, there is a huge potential flaw called *double spending*. It is a huge risk, where the digital asset can be easily falsified or reproduced and therefore can be spent twice. Physical currencies do not have these issues since they cannot be replicated so easily and are more easily verified [7].

This was a concern initially with Bitcoin since it doesn't use any central authority to be able to verify it. However, it was solved by conceptualizing the blockchain into Bitcoin by a person or group of people Satoshi Nakamoto in 2008. Bitcoin requires that all the valid transactions are included in the blockchain [8].

The concept of chaining has a security advantage against attackers. Let's have a look at blocks A, B chained in that order, each having a unique hash. If we go back to block A and alter it, the hash would change for block A even for B, since the block B is hashed with its data and the hash of the block A.

Blockchain, for the use as a public ledger, is managed by a peer-to-peer network following the rules of consensus. Any system, application or wallet that participates in the peer-to-peer network using the currency's protocol is called a *node*. Every node starts with a blockchain at least with one block. The first one is always statically encoded and is called *Genesis block*. The process of generating the blocks is called *mining* and the nodes doing such thing are *miners*.

1.5 Mining

Mining is a decentralized computational process that proposes a block by hashing it with the header of the most recent block and a *nonce* (an incremental number), and comparing it with the *target value.* Target value is being determined by the *mining difficulty*, which means if the blocks are mined too fast, the difficulty is increased. That is, as more *miners* join the network, the block creation increases and therefore the mining difficulty increases.

For the block to be accepted, the SHA-256 hash of the block must begin with a number of zero bits higher or equal to the target value. The probability of generating such hash is very low, so many attempts must be made. In each attempt, a *nonce* is increased, which is a number hashed with the block data. This is based on the *hashcash Proof-of-Work* and these variables, specifically a nonce and target difficulty, are also included in the block header [8].

After the hash is found, it must be verified by the other nodes. On top of that, the miner has to validate all the transactions before it is included into the blockchain. If the block is invalid, the network would reject the block and the miner would have wasted his time computing the hash for it.

It is intentionally designed to be difficult and therefore a *block reward* is given to the miner who finds it. It is created as the first transaction in the block by the miner and is known as a *coinbase transaction.* This creates an incentive for incoming miners. As miners are joining the network, more security to the network is provided and this is what makes the cryptocurrency decentralized [9].

The block reward has also an another meaning. It is a constant rate of generating new coins and it is the only way how the network generates new coins.

There is a flaw named *51% attack*, where an attacker controls more than 50% of the network's computing power. Currently, to be capable of performing such an attack, enormously expensive equipment is needed and that is economically impractical and might be even impossible [10].

1.6 Network

As previously mentioned, the network of cryptocurrency is structured as a peer-to-peer (P2P) network, which means that the nodes participating in the network are peers to each other, and are all equal. There is no "special" node, every node shares the same burden of providing the network services. The network is connected in a *flat* topology, that means there is no central server within the hierarchy of the network.

Despite every node being equal, each of them can have different roles depending on the functionality they support. A node can have these functions: *routing*, *the blockchain database*, *mining* and *wallet services* [1]. Every node includes *routing* function, which establishes the participation in the network. The type of nodes described in [1] are distinguished as:

Full node is a node that maintains a complete and up-to-date copy of the blockchain. It can autonomously verify any transaction without any external help.

Mining node is a node that does *mining* as described in the section *Mining*. Typically, they would also maintain the full blockchain otherwise they would put themselves at disadvantages of finding a block. They would need to rely on the external node to validate the incoming transactions that would be added in the block, they have just found.

Lightweight node is mostly run on space and power constrained devices, such as smartphones, tablets or embedded systems and therefore don't have the ability to store the full blockchain. These nodes are becoming the most common form of a node, especially for wallets.

1.7 Lightweight nodes

As already outlined in the previous chapter, lightweight nodes don't keep a track of a full blockchain, therefore they download only the block headers without the transactions. The resulting chain of blocks is much smaller than the full blockchain. Because of this, they cannot construct a full picture of any transaction and need to rely on an external node to get the partial view of them [1]. This method is called *simplified payment verification*, or *SPV*. These nodes are also called *SPV nodes*.

Let's have a look at what a *Merkle tree* and *Bloom filter* are, before describing the verifying part of an SPV node.

1.7.1 Merkle tree, Merkle path

A *Merkle tree* is a data structure used to efficiently verify the integrity of large sets of data. It is represented as a binary tree and it is constructed recursively from bottom-up by hashing the pair of nodes until the root, which is called *Merkle root*. The Merkle root is also included in the block header and is used to summarize all the transactions. As seen in the figure 1.1, the *leaves* are hashed transactions and its parent is a hash constructed by two children hashes concatenated together and then hashed [1].

A *Merkle path* is a set of hashes that are necessary to prove that a specific transaction is included in a block. For example, in order to verify a transaction T_xC (figure 1.1), the path would consists of the hashes H_{AB} and H_D. The rest would be computed by the hashes provided by the Merkle path and then compared with the root.

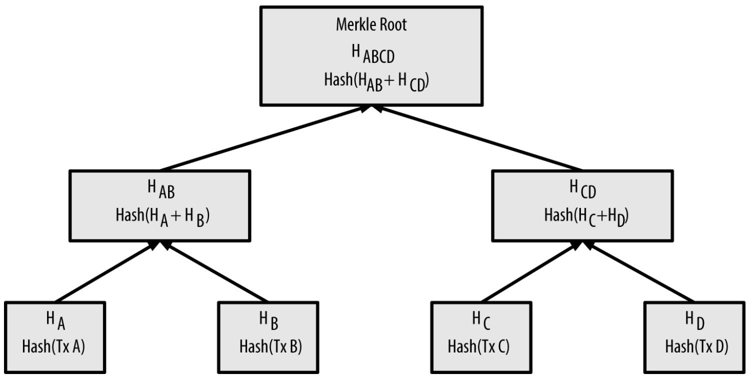

Figure 1.1: The merkle tree taken from [1].

1.7.2 Bloom filter

Asking selectively to verify the transactions creates a privacy risk leading to exposing the addresses and public keys associated with a user, thus completely destroying the user's privacy. This problem is resolved thanks to the feature called *Bloom filters*.

A Bloom filter is a probabilistic data structure, that creates a space-efficient filter describing a search pattern and is used to test membership of an element. Bloom filter allows to specify a search for transactions that can be balanced either towards precision or privacy. A more specific filter will produce more accurate results, but at the expense of revealing the patterns the node is interested in. On the other hand, a less specific filter will produce more irrelevant transactions but maintain a better privacy. Despite the structure being probabilistic, it generates only "false positives" but never "false negatives" – in other words, it returns either a "possibly a match" or "definitely not a match" [11].

Bloom filter is implemented as a bit array of N bits along with K hash functions. The hash function will produce such an output that would be matched to the 1–N bits and the bit is set to 1. The hash functions have to be deterministic, so that every node can get the same results for a specific input.

To test for the existence of data, we hash it and check if the output numbers are all set to 1 in the array. That means, the result is positive and it *probably* is in the set. On the contrary, if the data is tested and any one of the bits is set to 0, then it proves that the data was not recorded in a filter and it *certainly* is not in the set.

1.7.3 SPV node verifying

SPV nodes verify a transaction by establishing a link with a node containing a full blockchain. The SPV node creates a Bloom filter, adds and hashes the data the node is interested in, and then it will broadcast it. The peer will respond with the transactions matching the filter and block headers, in which the transaction lie, and with *Merkle paths*. The SPV nodes will afterwards discard all the redundant transactions and use the Merkle path to connect the transaction to the block and verify that it is included in the block. The SPV node also uses the block header to link the block to the rest of the blockchain. Together with these linkings, the transaction can be proved that it is recorded in the blockchain.

1.8 Wallets

A wallet is an application that servers mainly as a user interface. It gives an access to user's coins, tracks the balance, manages keys and addresses, creates and verifies transactions.

A common misconception about wallets is that wallets keep and contain the coins. The wallets keep only the keys that prove the ownership of the coins thus a wallet is basically just a *keychain*. The coins alone are recorded in the blockchain on the network.

The article [12] defines two types of wallets describing how the keys are created.

1.8.1 Non-deterministic wallets

A non-deterministic wallet generates a random number which represents a key. Initially, this was the only method and is very hugely discouraged. Every key would need to be backed up. Once the wallet is lost without a backup, there is no way to restore the keys and therefore the funds would be lost. This conflicts with the principle of *key re-usage*. Key re-usage decreases the privacy because it creates a link between transactions and addresses. To avoid this problem in this type of wallet, many keys would have to be created which leads to frequent backup. Ideally, every address should be used for one transaction.

1.8.2 Deterministic wallets

The deterministic wallet is a standard for every cryptocurrency. The keys are derived from a *seed*, which is randomly generated with other data and is defined in the BIP32 (bitcoin proposal number 32) [12]. The seed is sufficient to recover all the keys that are derived from it, hence the backup is only needed for the seed. The seed is usually a bunch of letters, mostly a hash and it is very hard to remember so the BIP39 solves this problem.

BIP32

The BIP32 describes how the keys are created. The form of the wallets is called *Hierarchical Deterministic Wallet* (HD wallet). The keys are derived from a tree structure, where a parent key can derive a sequence of children keys and each of them can derive a sequence of grandchildren etc. The seed (the root of the tree), which is randomly generated, is an input to an algorithm *HMAC-SHA512* producing a hash that is split into *master private key* (m) and *master chain code* (c). The corresponding *master public key* is generated by the elliptic curve multiplication process $m \cdot G$ (G is a base point of a prime order on the EC). The master chain code is used as an entropy to the function that creates the children keys from the parent. That function is called *child key derivation* or ckd.

The child key derivation consists of parent private and public key, the chain code, which is basically a seed and an index number. The chain code is used as a seemingly random data into the process of derivation, so the index number itself is not sufficient to derive the children keys.

Parent public key, chain code and an index are concatenated together and serve as an input to the hash algortihm HMAC-SHA512. The output is a hash of 512 bits and is split into two halves. The left half with the index are added to the parent private key to create a child private key. The right half is used as the child chain code.

By changing the index, the derivation would produce the child in the sequence, meaning index 0 would produce child-0, index 1 is child-1 etc. Every parent key can have up to 2 billion children keys.

The child key derivation by public key exposes a potential risk where with a knowledge of a child private key and the parent *extended public key* (an encoded hash with the public key and chain code for the export to make a public derivation tree) can be used to derive the rest of children private keys. Not only that, the parent private key could be deduced from it. To prevent this risk, the alternative form of derivation function is presented. It is called *hardened child key derivation*, where the parent private would be used instead of the parent public key. Using this method, the result of the derivation would be different than using normal derivation function. The resulted *branch* of keys can be used to produce public keys, which are not vulnerable, since the chain code cannot be used to deduce the parent private key [1]. This method is combined together with the normal ckd and can be used to create an effective HD wallet tree structure, which is defined in BIP44. However, in my simplified version, there is no need to use such a proposal.

BIP39

The BIP39 (defined in [13]) introduces the mnemonic code words – a sequence of words, which represents the seed. This sequence is enough to re-create the wallet with all the derived keys. Mnemonic code words enable for the user an easier backup of a wallet, since the words are easier to read and correctly transcribed, compared to the random sequence number or hash. The process of generating mnemonic code words works in a few steps as defined in the BIP39:

1. Create a random sequence (entropy) from 128 to 256 bits.
2. Create a checksum of the entropy by taking the number of bits, depending on the size of the entropy.
3. Add it to the end of the entropy.
4. Divide the entropy into the section of 11 bits, which serves as an index to the dictionary with pre-defined 2048 words.

The seed is then created by using the entropy as an input to a function *PBKDF2* using HMAC-SHA512 to stretch the hash (512 bits). It is then split into two halves and those are used to build a deterministic wallet.

This along with BIP32 became a standard for the cryptocurrency but each can have a different dictionary.

1.9 Transactions

Transactions are the most important part of the cryptocurrency. Everything that is designed in the cryptocurrency is to ensure, that transactions can be made, broadcast into the network, verified and added to the blockchain [1].

The life cycle of a transaction starts with the creation. Then it is signed by the user, which proves that the coins are his. After it is broadcast to the network where every node will validate it and broadcast further. Eventually, the transaction is verified and validated by a mining node and included into a block, which is added to the blockchain.

The transaction is a data structure, which encodes the exchange of coins from the source called *inputs* to the target *outputs*. These transaction inputs and outputs are not associated with users (identities), but rather to the amount of the coins locked by the secret (key signature) known only to the owner. By providing the right signature, the owner can unlock it and spend it.

1.9.1 Transaction outputs and inputs

The fundamental building block of a transaction is *transaction output*. Transaction outputs are indivisible data recorded on the blockchain and are recognized as valid by the entire network. The one that are spendable are called *unspent transaction output*, or *UTXO* and are tracked by the full nodes and kept in a collection called *UTXO set* [1].

Whenever a user “receives” the coins, it is recorded on the blockchain as UTXO. Thus, the coins of the user can be scattered as UTXO between hundreds of transactions and blocks. In fact, there is nothing such as a“balance”. The concept of a “balance” is created by the wallet and it is a sum of all UTXO belonging to the user by searching through the whole blockchain. This however is a very difficult operation for lightweight nodes, and that’s why they depend on the external full node.

Although an UTXO can have any amount when it is created, it is not possible to split it. That means, if I have an UTXO with 20 coins, and want to spend only half of it, I will have to create a transaction that has two outputs. First to the target and second to myself. This output is referred as a *change* [8].

This way, the transaction always consumes an UTXO and produces another UTXO that can be used in the future.

The exception from the chain of outputs and inputs is a special type transaction called coinbase transaction, which was previously mentioned in the section Mining. This transaction is always in the first place of the block and does not consume any UTXO, instead it has a special type of input called *coinbase* and has a slightly different information in the *transaction input* [14].

Transaction input is a pointer to the UTXO referenced by a hash of the transaction where the UTXO is recorded as an output in the transaction. It can be spent after providing a proof of ownership.

1.9.2 Transaction fee

Many transactions include a transaction fee, which are the reward for the miners. The transaction does not include these automatically and affects the priority of the processing. That means the greater the fee is, the faster it is included in the next block. On the other hand, it could take many blocks or it might not even be included. The fees are not mandatory and the transactions without the fees might be eventually processed, although the process can be encouraged by including a transaction fee to get a higher priority.

The transaction fee is set by the market strength which is influenced by the capacity of the network and the amount of transactions and is calculated based on the size of transaction in kilobytes rather than the coins in the transactions [15].

The reason the fee is calculated in kilobytes is because the miners are validating by looking through the transaction and the difficulty rises as the transaction data is bigger. Therefore the fee is set to the size of the transaction and not the amount.

The transaction as a data structure does not have a field for the transaction fee. Instead of it, it is calculated as the difference between the sum of inputs and the sum of outputs.

Chapter 2

Analysis and Design

In this chapter I will introduce the analysis of the current state of mobile wallets, explain the FITcoin as a cryptocurrency, specify the requirements of the wallet application, and then present the similar applications and what they support.

Based on the analysis from the first part, I will then describe the design and design choices of the application needed to fulfill the requirements and the communication protocol.

2.1 FITcoin as a cryptocurrency

FITcoin is a trivial form of cryptocurrency based on Bitcoin. It uses the same curve and domain parameters. Specifically the *secp256k1*, whose parameters are defined in [6].

2.1.1 Hash functions

The hash functions used in FITcoin are `SHA-256` and `RIPEMD-160`. Similarly to Bitcoin, every hash is calculated twice. It means, the hash function is applied twice on a data set. Specifically, `hash_256 = sha256(sha256(data))`. In case the shorter hash is needed, it is hashed in this way: `hash_160 = ripemd160(sha256(data))`. The only reason for the shorter hash is to have a shorter hash (160 bits instead of 256 bits).

2.1.2 Keys, addresses

Keys are generated by the algorithm ECDSA and addresses are a derived form of public keys by hashing with `hash_160`.

The Bitcoin address contains extra information such as checksum and a version, which is then encoded in the Base58. This format has its advantages for users. The encoding removes the visually identical looking characters and with the combination of the checksum it is durable against mistakes when transcribing the address.

The default FITcoin format for users is hexadecimal version of the hash – without the version and checksum. The length of such format is 40 bytes with the leading zeros. However, the mobile wallet application will be supporting the Base58 format.

2.1.3 Transactions

The structure of transactions in FITcoin is again very similar to Bitcoin transaction. Unlike Bitcoin transaction, the FITcoin one contains less fields, specifically the transaction script language, instead the signature will replace it (almost identical to the `scriptSig` in Bitcoin, which is used to verify the ownership).

The smallest value of the coin is one fitcoin (FTC) and is principally the same as 1 satoshi in Bitcoin (the smallest unit in Bitcoin corresponding to the value of $1 \cdot 10^{-8}$ bitcoin).

2.2 Target base

The target base of the application contains everyone who is interested in cryptocurrencies and wants to handle them. This should be accounted in order to design the application as simple as possible yet containing all the functionality required for a mobile wallet.

2.3 State of the Art

This section serves as a key factor when developing the application, pinpointing the flaws which should be considered in one's application. In my case, I will examine and focus mainly on graphical side and what they do support since my wallet will be focused on the FITcoin instead of the existing currencies such as Bitcoin, Ethereum, etc...

2.3.1 "Bitcoin Wallet"

The application was chosen for the analysis because it is the most downloaded wallet for Bitcoin with over one million downloads. It supports the basic stuff such as sending and receiving transactions and backup. The restoring process is slightly difficult. It requires the user to put the backup file into the specific folder which could be hard for a person without technical knowledge.

The GUI is slightly bugged, the first page is blank with no title and no content. When turning the phone horizontally, the GUI changes significantly. This phenomenom can be seen in the figure 2.1.

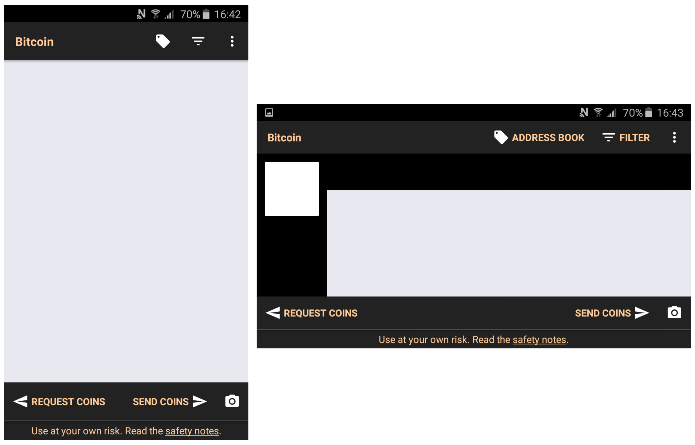

Figure 2.1: Bitcoin wallet screenshots.

2.3.2 Mycelium

Mycelium is the most advanced wallet there is. It offers many functionality mostly for the advanced users. The UI is very nice as seen in the figure 2.2. It supports 100% control over private keys, HD wallet manager, key exporting, connection through Mycelium super nodes and much more. Mycelium supports multi-accounts and random key generator. The only disadvantage I found is that the application does not show or give a possibility to view the keys in the HD account.

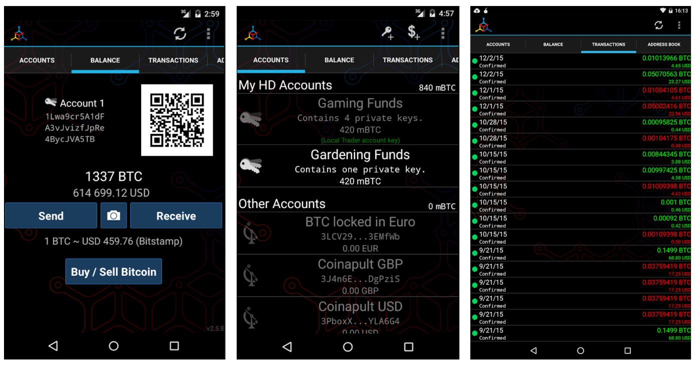

Figure 2.2: Mycelium screenshots from [2].

.

Functional Requirements
F1: Create an account
F2: Restore an account
F3: Key manager
F4: Track the balance
F5: Send and receive transactions

Non-functional Requirements
NF1: Security
NF2: Availability
NF3: Simple UI
NF4: Reliability
NF5: Data integrity

Figure 2.3: Application requirements

2.3.3 Coinomi

Coinomi is a very simple application and very easy to use. It supports like every wallet the basic functions with transactions. The only thing I found hard to use is key manager. Coinomi does not let users choose the keys to manipulate with. As far as UI is concerned, it is very simple and contains all the information needed for a user to find.

2.4 Application requirements

The application has a set of requirements it needs to fulfill. It can be divided into two types of requirements – functional and non-functional requirements. The functional requirements describe the behavior of the application, while non-functional specifies properties or restrictive conditions of the application. The requirements can be seen in the 2.3.

2.4.1 Functional requirements

The functional requirements are described as follows:

F1: Create an account The application will create an account without any registration on an initial start-up of the application and afterwards will work only on one account.

F2: Restore an account The application will be able to restore the account thanks to the BIP32 and BIP39, which will be implemented in the application.

F3: Key manager The application will manage keys for the account, there will be an option to create keys, delete keys and label the keys so the user knows what the key is used for.

F4: Track the balance The application will track the balance of each key by storing the database keeping the updated UTXOs.

F5: Send and receive transactions The application will be able to create transactions and send them to other nodes and receive them.

2.4.2 Non-functional requirements

The non-functional requirements are described as follows:

NF1: Security The application should be safeguarded against the faults such as inserting the wrong restore words or sending invalid transactions.

NF2: Availability The application should have an access to the internet to send and receive transactions. The history of transactions can be viewed offline.

NF3: Simple UI The application should have a simple UI in order for everyone to tell where is what and use it comfortably.

NF4: Reliability The application should have minimal failure rate and the respond time of the apllication should be fast.

NF5: Data integrity The application will save all the monetary info accurately in an integer. The DB should be safeguarded against mistakes or changes in them.

2.5 Architecture

The implementation will try to follow the MVC (Model-View-Controller) pattern. The application is split into three layers: models, views and controllers. Thanks to the Android syntax, the view part is split by default as the `XML` files in the folder `app/res/*`. The controller ensures the communication between the view part and the model part, which represents the data layer to get the saved and downloaded data.

The reason I have chosen this architecture is because it provides easier maintenance of the application in the late developing phase and changing any of the layer would project minimal changes to the other.

2.6 Communication protocol

The communication protocol is realized via message exchange. In the decentralized network, the messages are broadcast to the other nodes and based on the trust, the node accepts back the transmitted data. The available commands with the header structure can be found in the chapter Documentation.

Furthermore, as SPV node, the Bloom filter plays a huge role in the communication, which prevents and protects the user's privacy while getting the current transactions of the specific key owned by the user. The Bloom filter is implemented as [16], where the author analyzed the math behind it. Worth mentioning is the math equations for k, the number of hash functions and m, the size of the bloom filter. The equation takes in the parameters n, which is the number of elements and p as probability of false-positive rate. In my application, I use $p = 0.0005$, which is 0.05% and the number of elements derive from the amount of keys and addresses the user owns. The equations are as follows:

$$m = \frac{n\log(\frac{1}{p})}{\log^2(2)}, \ k = \frac{m\log(2)}{n}$$

2.7 Transaction design

The transaction in the FITcoin is much more trivial than Bitcoin and consists of `version`, `vin` (a list of inputs) and `vout` (a list of outputs). The current version is 1 and it gives us an opportunity to change the format in the future and keeps the validity of the older transactions.

Input consists of `txid` in a form of `Outpoint`, which is the hash of a previous transaction to which the input is referencing, `vout_idx` – an index to the list of output referenced to in the `txid` and lastly the `tx_sig`, which is a transaction signature. The transaction signature consists of two parts. The signature alone and the the public key used to verify the signature.

The output consists of `value` and `address` and indicates how much FTC the `address` will "receive".

The domain model of the transaction in FITcoin can be seen in figure 2.4.

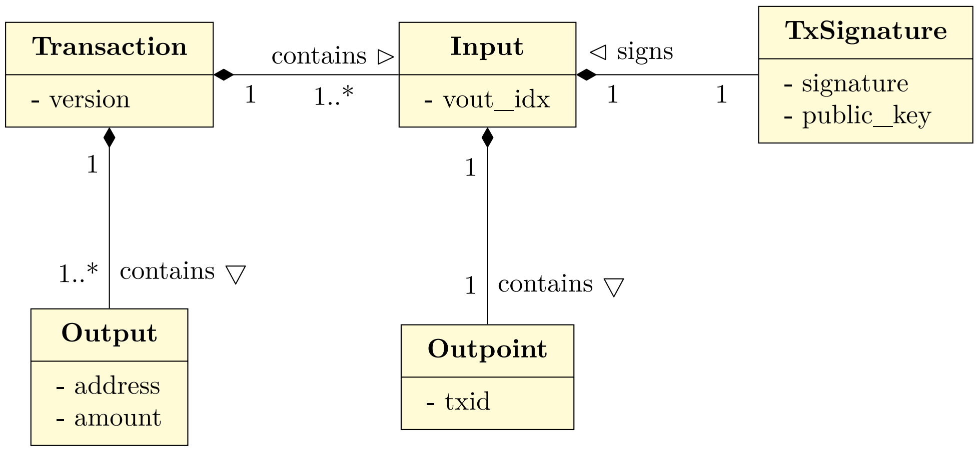

Figure 2.4: Domain model describing the transaction

2.8 Storage design

For the data storing I have chosen SQLite, which provides a relational database management system. According to [17], the SQLite is "*the most widely deployed database in the world...*"

SQLite is known for its properties and features, it is self-contained, serverless, needs zero configuration and is transactional. That makes the SQLite usable in environments such as mobile phones and does not need a separate server to operate with.

As far as security is concerned, the database is encrypted with the algorithm 256-bit AES in CBC mode and each database page is encrypted individually with its own randomly generated initialization vector. This is realised thanks to the *SQLCipher* [18].

To prevent data manipulation outside of the application, the tables contain an extra field for checksum, to test if the data is not corrupted.

The relational database can be seen in figure 2.5

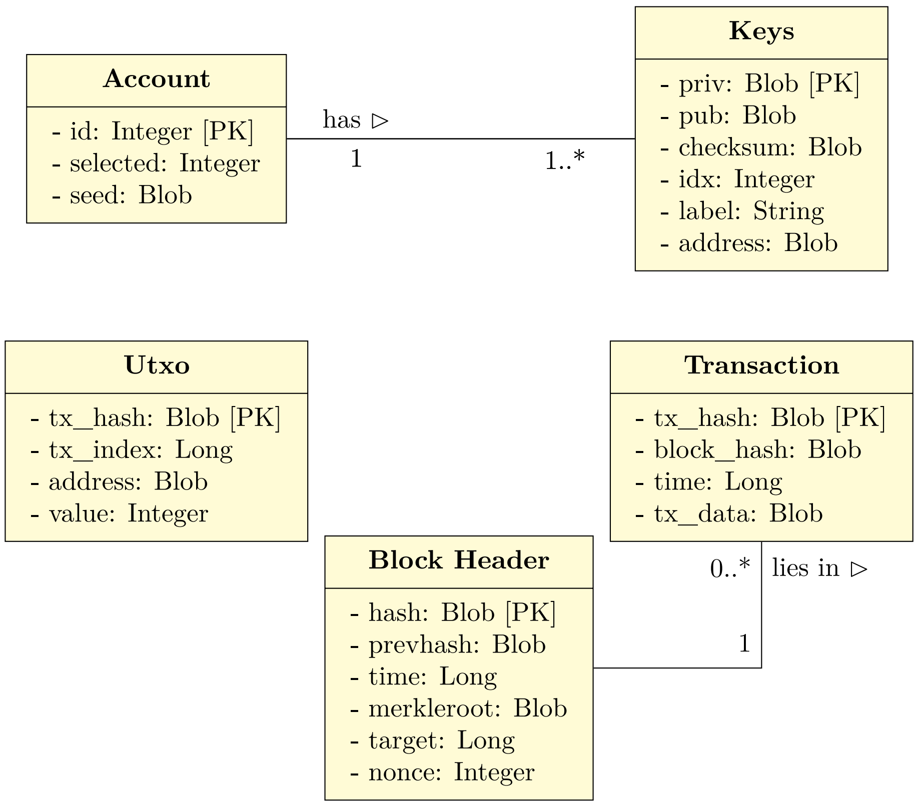

Figure 2.5: Relational model

2.9 Wireframes

For the design of the UI, I used the tool called *Mockplus* [19]. It is a powerful tool which helps quickly build the wireframes for the application. The interaction is visualized with a simple drag-and-drop components, basically *WYSIWYG* (What You See Is What You Get). The wireframes help the developers to adapt to the design and additionally it allows for the second party to know how the application will look.

As seen in figure 2.6, the main window consists of three fragments – accounts, balance and transactions. The account fragment gives an opportunity to create more deterministic keys, the balance fragment tracks all the utxo for the chosen key and the transaction fragment shows the list of the executed transactions. The fragment switching can be toggled either by swiping to the left or right side, or by clicking on the bottom navigation menu.

The wireframes of the actions can be seen in figure 2.7. The first screen describes the receive activity with a single input for the amount and a QR code which updates upon changing the amount. The second screen is transaction information showing its `ID`, the date and the amounts. The last shows the send activity, upon clicking the camera icon, the person can scan the QR code made from the main page or the receive.

Lastly, the wireframes of the setup page can be seen in figure 2.8. It gives the user an opportunity to create or restore the wallet. Upon clicking on create button, the twelve word list is generated and instructs the user to write it down, while clicking on the restore button will show the user a keyboard and enables inserting the twelve word list. If the user inserts more than two letters, the three suggestions will always show up.

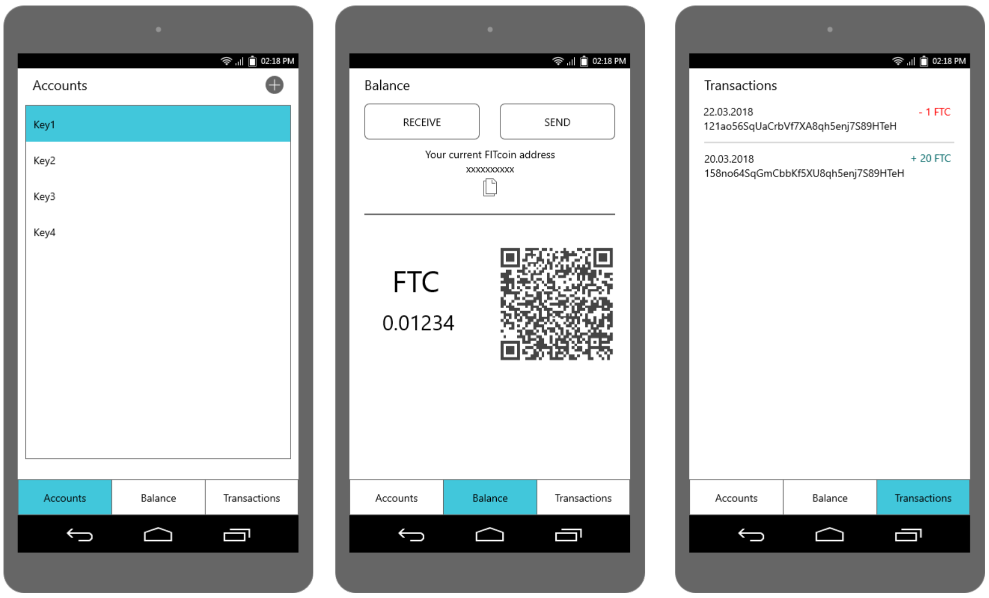

Figure 2.6: Wireframes of the main page.

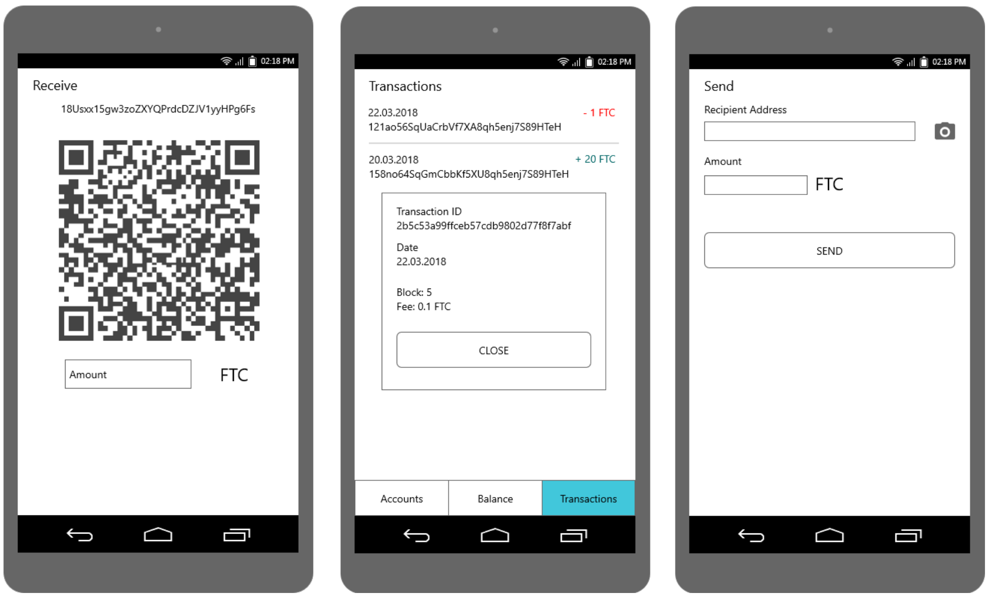

Figure 2.7: Wireframes of the actions.

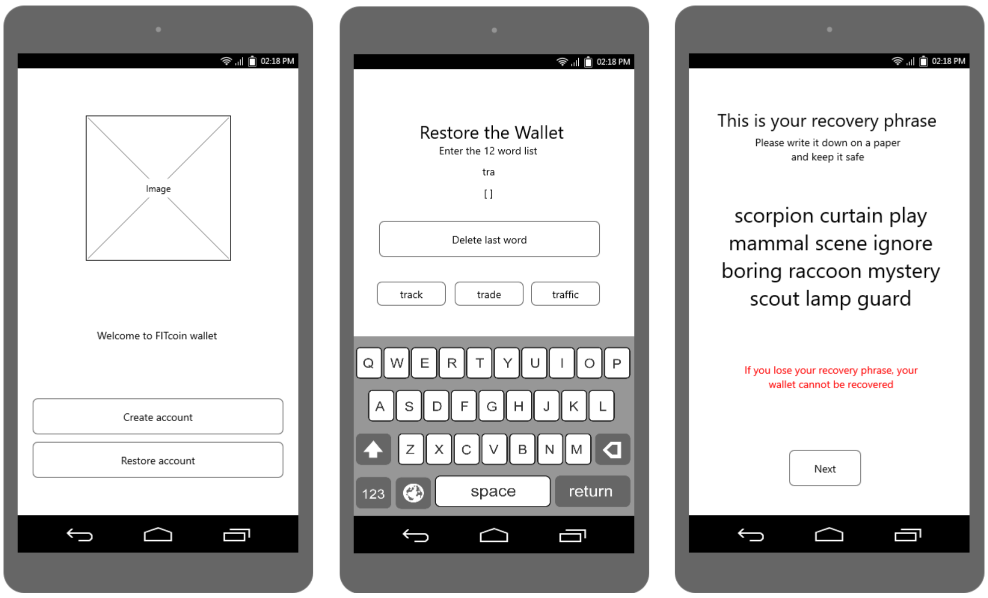
Image
Welcome to FITcoin wallet
Create account
Restore account
Restore the Wallet
Enter the 12 word list
tra
[ ]
Delete last word
track
trade
traffic
space
return
This is your recovery phrase
Please write it down on a paper and keep it safe
scorpion curtain play mammal scene ignore boring raccoon mystery scout lamp guard
If you lose your recovery phrase, your wallet cannot be recovered
Next

Figure 2.8: Wireframes of the set-up page.

2.10 Models

This section will describe the models of the final application.

2.10.1 Sending the transaction

Sending a transaction is one of the main processes of the wallet. The main goal is to enable the user to send the transaction comfortably. If the user won't have enough coins, it will warn the user about insufficient coins. In other case, the transaction will then be broadcast to the network and accepted if the transaction data is valid. The network will write it into the transaction pool, where it will await for the miner to find the block, and send back the message about its success. The activity diagram can be seen in figure 2.9.

2.10.2 Use cases

This part of section is about use cases which describe how the user can work with the application. These use cases can be seen in figure 2.10.

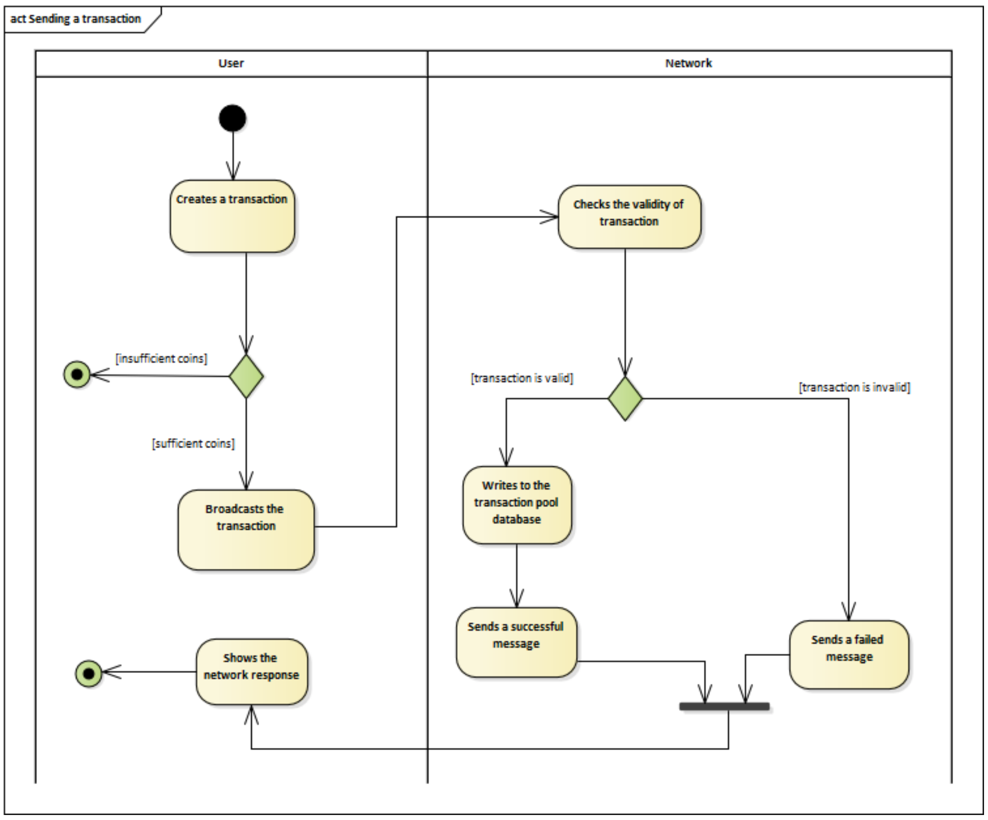
act Sending a transaction
User
Network
Creates a transaction
[insufficient coins]
[sufficient coins]
Broadcasts the transaction
Shows the network response
Checks the validity of transaction
[transaction is valid]
[transaction is invalid]
Writes to the transaction pool database
Sends a successful message
Sends a failed message

Figure 2.9: Activity diagram of sending the transaciton.

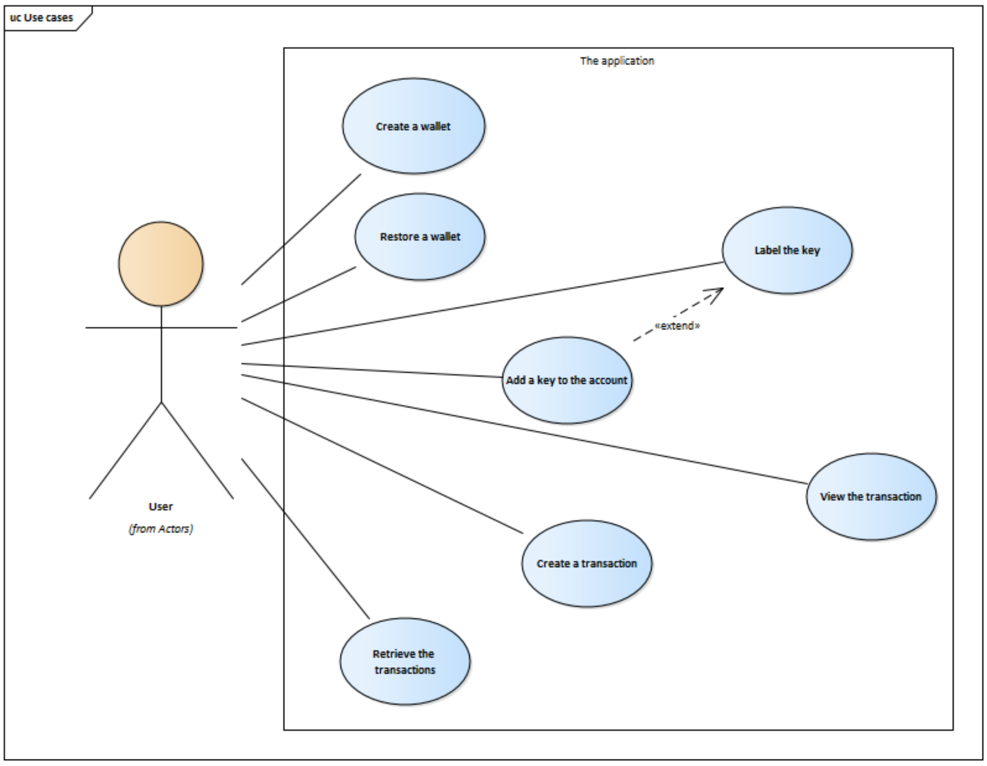

Figure 2.10: The use cases of the final application.

Chapter 3

Implementation

3.1 Language

For the development I have chosen Java which is the oldest and also the most used language for Android development. The alternative is *Kotlin* developed by JetBrains [20] and is relatively new so the community isn't so expanded as Java's and that's the primary reason why I have chosen Java. Most of the problems are already solved in Java and to find such a solution is much easier.

3.2 Development tools

In this section, I will describe the tools used while developing the application.

3.2.1 IDE

Android Studio is an official IDE created for the development for Android [21]. I have chosen this IDE because it is a good starting point for a beginner in Android development and this IDE is very easy to use.

3.2.2 Version control

For the version control I have chosen the web-based Git repository manager FIT Gitlab which tracks all of the changes in the development and all the commits were controlled by the desktop program *GitHub Desktop* [22].

3.2.3 UML diagrams

The diagrams were created in the *Enterprise Architect* in the UML language except for the relational model and domain model, which were created in the TikZ-UML. The program *Enterprise Architect* is used for creating diagrams describing the developing process, which leads to the generation of analytic documentation.

3.3 Libraries

The application relies on a few external libraries which helps and makes the development easier and safer.

3.3.1 Bouncy Castle

Bouncy castle is a lightweight crypto library. All the cryptographic operations and functions are realized via it – the hashing, operations over keys and signatures.

3.3.2 Zxing

Zxing supports the manipulation with QR codes, reading them and creating them. In my application it is used to generate the address and reading the QR code of the address to create and send a transaction.

3.3.3 SQLCipher

SQLCipher as mentioned in the previous chapter Analysis and Design section Storage design on page 21, is a library that cryptographically secures the SQLite data by encrypting it with AES-256 in CBC mode.

3.4 Developing process

As I am a beginner in Android development, I had to try the functionality of the Android so I started on small applications with individual features of the final application.

3.4.1 First steps

In the first application I tried to establish a connection between a client and a server. The first problem I came across right after creating a socket was that the Android didn't support opening a socket on the main thread and therefore the side thread had to be created for it. After the connection was established, the server and client had to communicate in some way. I created few data on the server-side program and communicated with it by simply exchanging the plain text messages to retrieve the data and sent back.

The second application was about the encryption and cryptography. The native cryptography library from Android is not sufficient and that's why I had to look for an alternative solution. I tried *JNI* (Java Native Interface), which is a framework that enables Java to call functions from other libraries written in another languages, to import the OpenSSL but I couldn't make it work so after a few attempts, I have chosen the Bouncy Castle.

After deciding over the crypto library, the key creation had to be studied. The Android supports its own version of `PrivateKey` and `PublicKey` which could be generated only together and randomly from `KeyPair`. Thankfully, the Bouncy Castle gives an opportunity to create a public key from a specific private key by using direct operations over the elliptic curve with the predefined variables.

However, this created another problem – a signature. The Android has its own `Signature` and requires the native `PrivateKey` and `PublicKey` as the input so I had to find a Bouncy Castle solution.

After a few trials, I found the first solution but it required the conversion from the `BigInteger` to `ECPublicKey` and `ECPrivateKey`. This was very ineffective so after a little research I found out about `ECDSASigner`. Thanks to this method, the problem with conversion was solved including the randomness in signature by presenting a deterministic k calculator, which processes the k with the data itself, therefore for each transaction the algorithm produces a different k. The standard algorithm for deterministic k calculator is described in RFC 6979 [23].

The next step was to find a way to store the data information, specifically the keys. Again, the Android offers `KeyStore` but works only for their native keys and furthermore I needed to save multiple keys in my application and this would be very ineffective. That's why I decided to use the SQLite.

3.4.2 The final realisation

After a few attempts on smaller applications, the features were put together and created into a new project. Many things were modified and furthermore added and currently, the application's structure looks like this:

```
│ Activity ........................ the package containing the activities
├─ Core ............................ the cryptocurrency model and settings
│  ├─ Bip39 ........................ the BIP39 implementation and models
│  ├─ Model ................. contains the data structure of cryptocurrency
│  └─ Wallet ..................... contains the data structure of the wallet
├─ Fragments ...................... activity fragments of the application
├─ Network ..................... network communication and its settings
│  ├─ Commands ........ the available commands to use in the application
│  ├─ Messages ............... the interface and messages available to send
│  └─ Responses ........... the interface and responses available to retrieve
├─ Random .......... the package with random interface and implementation
├─ Storage ..................... SQLite data storage and table definitions
└─ Utils .............................. the package with the useful classes
```

3.5 Code preview

In this section, I will introduce some code previews of the application concerning the transaction creation and network communication.

3.5.1 Messages and responses

The messages are realized via simple socket TCP/IP implementation and is established in the `AsyncTask` connection handler, where the message is executed. The messages are an implementation of the interface `IMessage` seen in the listing 1. The same applies to the responses. The responses are then processed in the `onPostExecute(IResponse response)` of the connection handler. This can be seen in the listing 2.

```java
public interface IMessage {

    /**
     * Executes the message and returns the response
     * @param out the outputstream of the socket
     * @param socket the connection socket with the other node
     * @return response
     */
    IResponse execute(DataOutputStream out, Socket socket);
}
```

Listing 1: Interface of the message

3.5.2 Transaction creation

The code below in the listing 3 describes the transaction creation, which is called when a user triggers the send action. It looks through the local `utxo` database if the user has sufficient coins to send. After the construction of `List<TransactionInput>` and `List<TransactionOutput>`, it is used to construct a `Transaction`, where the `selectedKeys` is used to sign the inputs.

```java
/**
 * An interface of the response
 * @param <T> The generic data type which will be processed and returned
 */
public interface IResponse<T> {

    /**
     * Gets of the data that was sent
     * @return the data
     */
    T getData();

    /**
     * Checks if the communication failed
     * @return true if the communication failed
     */
    boolean failed();

    /**
     * Returns a human readable message
     * @return a message
     */
    String message();

    /**
     * Process the data
     * @param context context
     */
    void process(Context context);
}
```

Listing 2: Interface of the response

```
public Transaction constructTransaction(){

    if(amount.getText().length() == 0 ||
    address.getText().length() == 0) {
        showErrorDialog("The address or amount can't be empty.");
        return null;
    }
    int am = Integer.valueOf(amount.getText().toString());
    int sum = 0;
    List<TransactionInput> inputs = new ArrayList<>();

    // construct the inputs
    for(UnspentTransactionOutput utxo: this.utxo){
        if(selectedKeys.getAddress().getEncoded()
           .equals(utxo.address.getEncoded()))
        {
            inputs.add(new TransactionInput(utxo.outPoint,
            utxo.tx_id));
            sum += utxo.amount;
            if(sum > am + Parameters.fee) break;
        }
    }
    if(sum < am+Parameters.fee) {
        showErrorDialog("You have unsufficient assets.
        The transaction couldn't be created.");
        return null;
    }

    // construct the outputs
    List<TransactionOutput> outputs = new ArrayList<>();
   outputs.add(new TransactionOutput(am,
   new Address(address.getText().toString().getBytes())));

    // construct change
    if(sum-am-Parameters.fee > 0){
       outputs.add(new TransactionOutput(sum-am-Parameters.fee,
       selectedKeys.getAddress()));
    }
    return new Transaction(inputs,outputs,selectedKeys);

}
```

Listing 3: Code preview of transaction creation

Chapter 4

Documentation

4.1 Javadoc

Javadoc is a documentation system to generate the documentation in `HTML` format from Java source. It is useful to comment on the classes, interfaces and methods to ensure people understand what they are used for. The technical java documentation was generated through the IDE Android Studio and can be found in the attachments.

4.2 Available network messages

The network messages are developed by the student Bc. Karel Pajskr. The list of available commands is described below.

Message structure

Field name	Data type	Field size	Comments
magic value	`uint32_t`	4	{0xf9, 0xbe, 0xb4, 0xd9}
command	`char[12]`	12	the type of command
length	`int`	4	the length of the data

version

This command is sent by the node that establishes the connection.

Field name	Data type	Field size	Comments
version	`int_t`	4	the version of the client

getaddr

The command to get the known peers (nodes in the network).

addr

The message with the list of IP of the peers.

Field name	Data type	Field size	Comments
address count	`int`	4	
IP addresses	{`uint32_t` IP, `uint16_t` port}		

inv

The response to the command `getblocks` or broadcasting the hash of the new transaction or block to the other peers.

Field name	Data type	Field size	Comments
count	`uint32_t`	4	the size of the inventory
inventory	`inv_vec [count]`		

The structure of the `inv_vec` looks like this:

Field name	Data type	Field size	Comments
type	`uint8_t`	1	which type of the hash it is
hash	`uint8_t[32]`	32	the hash

notfound

The response to the command `getdata`. It is sent if no hash is found.

Field name	Data type	Field size	Comments
count	`uint32_t`	4	the size of the inventory
inventory	`inv_vec [count]`		hashes that couldn't be found

tx

The response to the `getdata`. It contains only one serialized transaction.

4.3 User manual

As this is a very simple mobile wallet application, there is only a few actions that can be triggered. The list of available actions is described below in the table 4.1. The screenshots of the usage can be found in the attachments.

Table 4.1: User manual

Item	Description
[Initial wallet creation]	When turning on the application for the first time, there will be a choice to create a wallet. When you click on it, the 12 word list phrase will show up with a button to continue. Upon clicking on the button, the wallet is created.
[Restore the wallet]	When turning on the application for the first time, there will be a choice to restore the wallet. When you click on it, the application will ask you to fill the 12 word list phrase to restore the wallet. The 12 word list is afterwards checked if it is correct.
[Add key to the account]	To create a key, you have to be at the *Accounts* tab and press on the image button which represents the key creation. When you click on it, there will be a possibility to label the key.
[Delete a key]	To delete a key, you have to be at the *Accounts* tab and press on the key you want to delete. Afterwards the menu will pop up and simply click on the delete button, which is represented by the trash bin logo. However, the balance of the key must be zero, otherwise there won't be a possibility to delete it.
[Rename the key]	To rename a key, you have to be at the *Accounts* tab and press on the key you want to rename. The menu will pop up with a icon of the pencil for renaming.
[Send a transaction]	To send a transaction, you have to be at the *Balance* tab and press on the button *Send*. The new activity will pop up asking for the address of the recipient and the amount you want to send. There is a choice to scan the QR code of the prepared address with the amount or simply only the address.
[View a transaction]	To view a transaction, browse to the *Transactions* tab and click on the transaction that you want to view.

Chapter 5

Testing

This chapter will be concerned about testing, which is one of the most important process while developing an application. The purpose of this is to ensure that the application works correctly with the least errors. For the testing, I have prepared the unit testing and user testing.

5.1 Unit testing

The Unit testing is focused on the functionality of the classes and does not need the access to the database or application context. The Unit testing is realized via JUnit framework which is written in Java [24].

The most important is to ensure that the cryptocurrency core classes work always correctly, therefore for the Unit testing, I have created 5 test suites representing the parts of the core with 19 test cases. The test suites can be found in the attachments.

5.1.1 Results

The tests helped me solve multiple problems, mostly it were the serialisation of the transaction which is a very important part of the wallet. The serialisation needs to be exact and fulfill the form so the consensus over the network would work.

5.2 User testing

The user testing's purpose is to test the practical use of the application. For this testing, the test application is prepared with the test scenario and the outcome of the scenario. The testers will follow this scenario and if the application's behaviour is the same as the outcome, the test is fulfilled. In the other case, it is not and the application behaviour is reported back.

5.2.1 Test scenarios

The first scenario is to create a key and label it with the string “mykey”. The outcome of such scenario is the added key with labeled string showing in the *Accounts* tab.

The second scenario is to switch between keys in the account. This change will affect the *Balance* tab, where the sum and the key should be the one that a user have chosen.

The third and fourth scenario is creating and sending the transaction. One is with the sufficient amount, the other test is not. When inserting the sufficient amount, the transaction should be created and broadcast to the network. In such case, either the connection can’t be established or it will successfully be sent and the balance should change. In the other case the users should be warned by the dialog saying there is not sufficient assets to spend.

The fifth scenario is about receiving the transactions by clicking on the refresh icon in the top menu. The user should receive a message about the success of fetching the data and on the *Transactions* tab, a new transaction should pop up.

5.2.2 Results

The results were all successful and no unexpected nor wrong behaviour was found while testing it. The group of testers were mostly students of CTU FIT for which I thank them for participating in the testing. There were small issues but all of them were fixed immediately.

5.2.3 Important note

The tests were tested with the dummy implementation of the network since Bc. Karel Pajskr did not manage to finish his Bachelor’s thesis in time.

Conclusion

The goal of the thesis was to create a functional prototype of mobile wallet on operating system Android, which will support the basic functions such as creating and sending the transactions, retrieving them and maintaining the balance on the account.

The prototype was created in Java language with additional features such as restoring the account or creating more keys into the account. The prototype was documented, thoroughly tested in the JUnit environment and by the users.

Unfortunately, the network communication, which was developed by the student Bc. Karel Pajskr, was not finished in time and therefore the dummy implementation had to be created for the application.

The future plans of the application could be support of more languages, work over more than one account and the possibility to restore more than one wallets (a trezor). As far as security is concerned, the application could be secured by the password and the database with the user entered password.

Bibliography

[1] ANTONOPOULOS, A. *Mastering Bitcoin: Programming the open blockchain.* O'Reilly Media, Inc, USA, second edition, 2017.

[2] Mycelium Developers. *Mycelium [online].* [cit. 2018-04-07]. Available from: `https://play.google.com/store/apps/details?id=com.mycelium.wallet`

[3] Einstein.Exchange. *Bitcoin Basics Lesson 2: Essentials of Bitcoin [online].* [cit. 2018-03-11]. Available from: `https://medium.com/einstein-exchange/bitcoin-basics-lesson-2-5727b9591a78`

[4] DJEMILEVA, E. *Aplikace eliptických křivek v kryptografii.* Master's thesis, Bankovní institut vysoká škola Praha, Praha, 2014.

[5] Techwalla. *What Are the Advantages & Disadvantages of Elliptic Curve Cryptography for Wireless Security? [online].* [cit. 2018-03-11]. Available from: `https://www.techwalla.com/articles/what-are-the-advantages-disadvantages-of-elliptic-curve-cryptography-for-wireless-security`

[6] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters.* [cit. 2018-03-11]. Available from: `http://www.secg.org/sec2-v2.pdf`

[7] CHOHAN, U. *The Double Spending Problem and Cryptocurrencies.* University of New South Wales (UNSW), UNSW Business School, discussion Paper.

[8] NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash system [online].* [cit. 2018-03-13]. Available from: `https://bitcoin.org/bitcoin.pdf`

[9] Anon. *Everything you need to know about Bitcoin mining [online]*. [cit. 2018-03-13]. Available from: `https://www.bitcoinmining.com/`

[10] Investopedia. *51% attack [online]*. [cit. 2018-03-13]. Available from: `https://www.investopedia.com/terms/1/51-attack.asp`

[11] HASSAN, A. S. *Probabilistic Data structures: Bloom filter [online]*. [cit. 2018-03-15]. Available from: `https://hackernoon.com/probabilistic-data-structures-bloom-filter-5374112a7832`

[12] WUILLE, P. *Hierarchical Deterministic Wallets [online]*. 2012, [cit. 2018-03-17]. Available from: `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`

[13] PALATINUS, M.; RUSNAK, P.; VOISINE, A.; et al. *Mnemonic code for generating deterministic keys [online]*. 2012, [cit. 2018-03-17]. Available from: `https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`

[14] MORROW, J. *What is coinbase transaction? [online]*. [cit. 2018-03-18]. Available from: `https://blog.cex.io/bitcoin-dictionary/coinbase-transaction-12088`

[15] COHEN, B. *How wallets can handle transaction fees [online]*. [cit. 2018-03-18]. Available from: `https://medium.com/@bramcohen/how-wallets-can-handle-transaction-fees-ff5d020d14fb`

[16] CORTESI, A. *3 Rules of thumb for Bloom Filters [online]*. [cit. 2018-04-16]. Available from: `https://corte.si/posts/code/bloom-filter-rules-of-thumb/index.html`

[17] SQLite developers. *SQLite – SQL database engine [online]*. [cit. 2018-04-14]. Available from: `http://sqlite.org/index.html`

[18] Zetetic, LLC. *SQLite Cipher [library]*. [cit. 2018-04-17]. Available from: `https://www.zetetic.net/sqlcipher/`

[19] Jongde Software LLC. *Mockplus [software]*. [avail. 2018-04-25]. Available from: `https://www.mockplus.com/`

[20] JetBrains. *Kotlin Programming Language [online]*. [avail. 2018-04-27]. Available from: `https://www.kotlinlang.com/`

[21] Google Inc. *Android Studio – The Official IDE for Android [online]*. [cit. 2018-04-10]. Available from: `https://developer.android.com/index.html`

[22] GitHub, Inc. *GitHub Desktop [software]*. [avail. 2018-04-25]. Available from: `https://desktop.github.com/`

[23] PORNIN, T. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) [online]*. [avail. 2018-04-27]. Available from: `https://tools.ietf.org/html/rfc6979`

[24] The JUnit Team. *JUnit 5 [library]*. [avail. 2018-05-07]. Available from: `https://junit.org/junit5/`

Appendix A

Acronyms

GUI	Graphical user interface
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
P2P	Peer-to-peer
UTXO	Unspent transaction output
IDE	Integrated Development Environment

Appendix B

Contents of enclosed CD

- `fitcoin.apk` .... the application package of the final prototype
- `documentation` .... the directory with the generated documentation
- `screenshots` .... the directory with the screenshots
- `src` .... the directory of source codes
  - `app` .... implementation sources
  - `thesis` .... the directory of LaTeX source codes of the thesis
- `text` .... the thesis text directory
  - `thesis.pdf` .... the thesis text in PDF format