BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

VIRTUAL WALLET COMPATIBLE WITH CRYPTOCURRENCY

VIRTUÁLNÍ PENĚŽENKA KOMPATIBILNÍ S KRYPTOMĚNOU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MAREK MARUŠIN**
AUTOR PRÁCE

SUPERVISOR **Mgr. Ing. PAVEL OČENÁŠEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Marušin Marek**
Program: Informační technologie
Název: **Virtuální peněženka kompatibilní s kryptoměnou**
Virtual Wallet Compatible with Cryptocurrency
Kategorie: Elektronický obchod

Zadání:

1. Seznamte se s fungováním kryptoměn a s možností implementace virtuální peněženky.
2. Analyzujte požadavky na jednoduchou virtuální peněženku, která bude podporovat existující kryptoměnu. Peněženka bude podporovat ukládání privátních klíčů, správu transakcí, správu tokenů, uživatelský přístup.
3. Na základě analýzy navrhněte takovou virtuální peněženku.
4. Navržené řešení implementujte dle instrukcí vedoucího práce.
5. Implementované řešení otestujte v reálném prostředí.

Literatura:

- Raval, Siraj: Decentralized Applications, O'Reilly Media, 2016.
- Antonopoulus, Andreas: Mastering Bitcoin, O'Reilly Media, 2017.
- Antonopoulus, Andreas: Mastering Ethereum, O'Reilly Media, 2018.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 - 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Očenášek Pavel, Mgr. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 31. října 2018

Abstract

The purpose of this thesis is to identify differences between classical payment systems and cryptocurrencies. A paper also compares the most well-known cryptocurrencies and explains how they work. By analyzing the principles of cryptocurrencies focused mainly on the Ethereum, we can see how virtual wallets for these digital tokens can be implemented and how we can work with custom tokens. A demonstration of created virtual wallet for Ethereum and custom ERC20 tokens is shown. Implemented practical example using Python programming language covers principles as generating private and public keys with appropriate addresses, manipulation with tokens and transactions or providing tests.

Abstrakt

Účelom tejto práce je analyzovať rozdiely medzi klasickými platobnými systémami a kryptomenami. Štúdia tiež porovnáva najznámejšie kryptomeny a vysvetluje ako pracujú. Analýzov prinípov ktyptomien zamerané hlavne na Ethereum, ale aj Bitcoin, budeme vidiet ako je možné naimplementovať virtuálnu peňaženku určenú na ukladanie takýchto digitálnych tokenov ci spracoavnie vlastnych tokenov. Na koniec je popísaná implementáciu virtuálnej peňaženky pre Ethereum a ERC20 tokeny. Praktická ukážka naprogramovaná v jazyku Python zahrnie väčšinu popísaných princípov vrátane generovania prívatných a verejných kľúčov s príslušnými adresami, manipuláciu s tranzakciami či tokenmi.

Keywords

wallet, payment systems, cryptocurrencies, Ethereum, ERC20, Python, blockchain

Kľúčové slová

peňaženka, platobné systémy, kryptomeny, Ethereum, ERC20, Python, blockchain

Reference

MARUŠIN, Marek. *Virtual Wallet Compatible with Cryptocurrency.* Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Ing. Pavel Očenášek, Ph.D.

Rozšírený abstrakt

Sú kryptomeny len digitálne peniaze? Na prelome rokov 2017 a 2018 sa nafúkla bublina kryptomien a po celom svete sa začali šíriť rôzne pravdivé, či nepravdivé fakty. Táto práca vysvetľuje to, čo vlastne tieto digitálne tokeny sú a ako fungujú, a či budú na niečo dobré. Práca sa pritom zameriava čisto na technický pohľad veci. Akékoľvek finančné či filozofické otázky nie sú zodpovedané.

Na to, aby sme umožnili prístup užívateľov ku kryptomenám, potrebujeme peňaženku. Táto peňaženka je software, ktorý umožnuje správu adries a digitálnych tokenov. Peňaženka sa taktiež stará a bezpečnosť, ako spracovanie kľúčov, či podpisovanie tranzakcií. V práci sú opísané najmä technické detaily k tomu, ako vytvoriť takúto zabezpečenú virtuálnu peňaženku. Následne je takáto peňaženka aj navrhnutá a implementovaná. V práci je tiež obsiahnuté porovnanie dnešných populárných riešení peňaženiek, a to aké štandardy či zaužívané praktiky by mali byť v takýchto peňaženkách naimplementované. Avšak práca s kryptomenami nie je vždy zadarmo a peňaženka potrebuje vypočítať aj poplatky za tranzakcie, či to ako si užívateľ kedykoľvek môže overiť akú tranzakciu vykonal. Aj tomuto je v práci venovaný dôraz.

Jeden z kľúčových faktorov je pochopiť čo je to *blockchian*. Blockchain je dôveryhodná, otvorená, globálna a distibuovaná účtovná kniha. V jednoduchosti, akákoľvek tranzakcia s kryptomenou vznikne je zapísaná do bloku a tento blok je zapísaný do reťazca blokov (anglicky block-chain). Blockchain je distribuovaný na tisíckach počítačov a kryptograficky overený. Čo zasa znamená, že ak by útočník chcel túto informáciu pozmeniť, musel by ju pozmeniť na veľkom množstve zariadení a s neuveriteľne silným výkonom na výpočty, čo je takmer nemožné. Informácia zapísaná do blockchainu je takto na stálo transparentná a nezmeniteľná. Práca porovnáva, aký je v konečnom dôsledku rozdiel medzi typickými centralizovanými platobnými systémami a decentralizovanými či distribuovanými platobnými systémami, ako sú kryptomeny.

Jedny z najpopulárnejších kryptomien založených na blockchaine sú *Bitcoin* a *Ethereum*. Práve Ethereum je kryptomena, na ktorú sa práca najviac zameriava. Nakoľko väčšina digitálnych tokenov funguje na podobných princípoch, množstvo technických detailov je možné využiť ako solídny základ do celkového porozumenia kryptomien.

Ale je Ethereum len ďalšie digitálne aktívum? Ethereum je často nazývané ako *svetový počítač*, ktorý dokáže spúšťať decentralizované programy a meniť týmto svoj stav za pomoci verejného blockchainu. Prostredníctvom takých decentralizovaných programov si každý dokáže vytvoriť aj svoj vlastný užívateľský token. Práve tokeny Etherea a užívateľské tokeny spracováva peňaženka naimplementovaná v tejto práci.

Práca sa tiež zameriava na rôzne porovnanie algoritmov ako sú nedeterministické či hierarchické peňaženky. Je možné sa dočítať o generovaní kľúčov a použitých algoritmoch pre derivovanie adries. Opísané sú rôzne komunitné štandardy, ktorými je dobré sa riadiť, či to ako je možné naprogramovať kompatibilnú peňaženku s viacerými kryptomenami a viacerými užívateľskými účtami. Je kladený dôraz na to, ako je možné prenášať peňaženku medzi zariadeniami, či v prípade straty privátneho kľúča peňaženku obnoviť. V kapitole o testovaní je následne popísané, ako je možné na úrovni kódu zabezpečiť funkcionálne, či integračné testy.

Virtual Wallet Compatible with Cryptocurrency

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Mgr. Ing. Pavel Očenášek, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Marek Marušin
May 15, 2019

Acknowledgements

I would like to say thank you to Mgr. Ing. Pavel Očenášek, Ph.D. for his professionality and all bits of advice during two semesters of consultations.

Contents

Chapter 1

Introduction

Cryptocurrency, another word for digital money. At the beginning of the year 2018, some of the most-well known cryptocurrencies as Bitcoin or Ethereum were gaining on huge popularity. Most of the cryptocurrencies are driven by a technology called blockchain. Blockchain is trusted, open, public, global and distributed ledger. By providing access to the blockchain to ordinary users, we are providing access to their cryptocurrency, their digital assets, and all transparent transactions. One of the ways how to provide this access is via wallet software.

This paper is focused mainly on principles of the Ethereum network, explaining what it is and how it works. This knowledge from chapters 2 - 5 is then used in following chapters to build fully working virtual wallet compatible with Ethereum build-in tokens and all custom tokens created on the Ethereum blockchain. Such a wallet is responsible for manipulating with keys and transaction management.

Beyond wallet implementation, which is the main goal, another aim is to enhance awareness of the basic technical principles. In this way, I would like to show that the technology has great potential for shaping the whole society. However, writing this text has only technical motivation and does not solve financial or philosophical questions.

In general, cryptocurrencies have a short history and as popularity bubble is getting bigger, more researchers and projects are building applications based on blockchain. In the future, it can be considered as a new scientific discipline. Currently, there are very little guidelines on how to build a secure Ethereum wallet using Python complying with best practices and community standards with better understanding. It is my belief that many of the described standards and practices within this paper become widespread across most of the cryptocurrencies. This knowledge can be used as a solid background to go deeper and build more sophisticated solutions on blockchain technology.

Chapter 2

Digital cash

Within this chapter, a basic comparison between payment systems is provided with a basic explanation of what a technology stands behind many cryptocurrencies.

2.1 Typical payment systems and centralization

Over the years, alternative electronic payment system becomes an everyday part of our life which includes debit or credit cards, electronic fund transfers, internet banking, e-commerce and much more. All these entities have one thing in common. They are centralized. That means that there is always one central authority. Servers owned by one or more financial institutions like banks, governments, credit cards companies, and others. People rely on their electronic products as trusted third parties to process electronic payments. All these payment systems are working well for most of the transactions and business logic. They are providing for example identity authentification, digital signatures with cryptography or trying to fix double spending problem [30] where a single unit can be spent twice. Anyway, there is growing problem linked to centralized payment systems. It is a weakness of a trust base model. Financial institutions can be hacked, slow money transfer across countries, demand big fees, cannot completely avoid to non-reversible transactions and so on [31].

Centralized network scheme is shown in Figure 2.1.

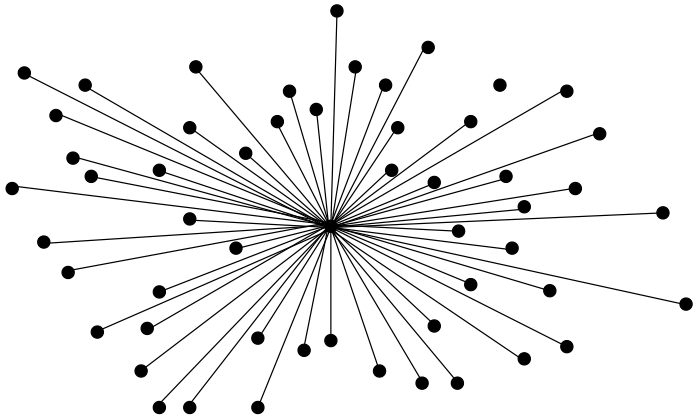


Figure 2.1: Centralized network scheme (adapted from [26]).

2.2 Decentralized payment systems and blockchain

Fintech industry brought us many inventions and one of the latest are protocols for decentralized payment systems as Bitcoin or other cryptocurrencies. This enables people via a peer-to-peer electronic cash system to established trust and make transactions without a third party. Cryptocurrencies, unlike traditional banking and payment systems, are based on decentralized trust [31] [32].

For example, in the Bitcoin network, this is reached by following components [24]:

- a decentralized peer-to-peer network.
- a decentralized mathematical and deterministic currency issuance called distributed mining.
- a decentralized transaction verification system.
- a public transaction ledger called blockchain.

The blockchain is underlying technology of most cryptocurrencies that allows us to establish trust not by big institutions but by collaboration of many computers and cryptography on the highest level. Every single transaction is sent and verified by millions of devices over the world which creates a growing list of trusted records, called blocks. All these blocks are linked cryptographically into one open, public, global, distributed ledger which can be verified any time by everyone in the network. In this way, transactions are free of any central authority that can be corrupted and avoiding a single point of attack [31].

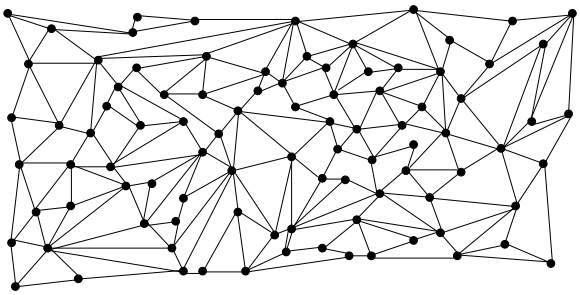


Figure 2.2: Distributed network scheme (adapted from [26]).

Cryptocurrencies may differ between each other in many meanings. In this section Bitcoin's high-level abstraction was described. As we will see in the next section, there are many more use-cases for blockchain technologies than just creating digital cash. It can be used to achieve consensus on decentralized networks to prove the fairness of elections, lotteries, asset registries, medical records and much more. Distributed network scheme is shown in Figure 2.2 while decentralized network scheme which is a subset of a distributed scheme is shown in Figure 2.3.

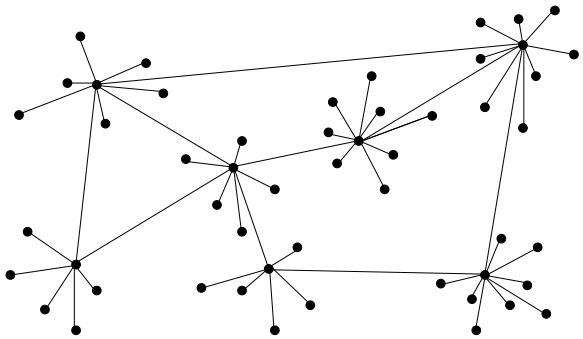


Figure 2.3: Decentralized network scheme is a subset of a distributed network (adapted from [26]).

2.3 Ethereum and smart contracts

Is Ethereum just another cryptocurrency?

It is very often described as „the world computer." That is because Ethereum's purpose is not primarily to become digital cash. Ethereum is an open globally decentralized computing infrastructure that executes programs called *smart contracts*[1] [25]. It uses a blockchain described in Chapter 2.2 to synchronize and store the system's state changes.

But Ethereum has also currency unit called *ether*. Ether is used as an build-in payment method on Ethereum platform. The Ethereum users can pay with ether for using Ethereum network as the world computer.

Of course, everybody can use ether also for paying merchants for their goods. But more important is that the Ethereum platform enables to deploy smart contracts and **decentralized applications** with built-in economic functions. While comparing it to the Bitcoin which just track the state of digital tokens and their ownership using open distributed blockchain.

While Bitcoin is considered as a payment system, the groundbreaking innovation of Ethereum is to combine general-purpose computing architecture with a decentralized blockchain. That means that Ethereum creates a distributed single-state (singleton) world computer. Such technology has great potential to change how people interact with the internet and computers at all. It also changes the trust base model since everything that happens in the network is transparent and cryptographically verified by many computers without a central authority [14] [11] [10].

Ethereum platform and smart contracts are key factors for this paper. In the next chapters, the creation of virtual wallet for Ethereum tokens and Ethereum ERC20 tokens is shown.

ERC20

ERCs are nothing else than just technical standards similar to IETF's RFC[2]. ERC stands for Ethereum Request for Comments and defines technical aspects of development on Ethereum platform [25].

[1] Smart contract: is a computer program written in high-level languages (e.g., Solidity) executable on the Ethereum platform. There is an effort to use open blockchain smart contracts as real-world legal agreements between two parties enforced by law without using middleman and third parties.

[2] RFC: Request for Comments are technical notes and standards about the internet [21].

Smart contracts[1] can define any rules for it's users. When a smart contract is deployed onto blockchain anyone who wants to interact with it must follow these rules. It is also possible to implement a new token with custom behavior and in a fixed amount of these tokens. Issuing custom tokens was so popular that the community become with standard marked as **ERC20** which makes all custom tokens compatible with each other and with Ether wallets. This protocol describes simple rules of how new digital tokens must be implemented on the Ethereum network. That means that anyone can issue cryptocurrency owned by himself.

Therefore, developers of Ethereum wallets can have just one major implementation for all ERC20 tokens. ERC20 also includes how tokens can be transferred or accessed. At the time of writing this text, there are 184 687 [5] different ERC20 tokens implemented on Ethereum network. ERC20 tokens become popular also because of crowdfunding companies working on ICO[3] [9] [8].

In the next chapters, a virtual wallet for Ethereum ERC20 tokens is developed and principles on how to manipulate and secure this tokens are shown.

[3]initial coin offering or initial currency offering (ICO) - funding using cryptocurrencies issued by private companies.

Chapter 3

Wallets

The term "wallet„ has many meanings. In this text, the term wallet is used for any software application that helps to manage user's Ethereum accounts and give access to his digital tokens. There is one often misunderstanding that wallets are storing some virtual coins. Ethereum wallets at a very fundamental level just connect to the blockchain and hold private keys or broadcast signed transactions. Ethereum platform makes the rest of the work. Wallet's balances and executed transactions are verified and available globally on a distributed public blockchain.

In my opinion, wallet's major must-have features are:

- Generate and securely store private keys.
- Give user option to decide what to do with private keys.
- Elegant, easy to use UI.
- Backup and restore features.
- Open source code, which means better community control and contributions.

3.1 Ethereum clients

Ethereum is defined by a formal specification called the *Yellow Paper* [33]. An implementation which follows specification's rules defined in Yellow Paper is called Ethereum client. There are two options to run Ethereum client. One is full node another is a remote client.

3.1.1 Full node

A full node must download blockchain data and store it locally on its hard drive. This requires some specific hardware resources as blockchain increases in time with new transactions and blocks. A full node is also responsible for an authoritative and independent verification of all transactions and smart contracts. Behavior and hardware resources of a full node depend also on connected Ethereum network. There exist a variety of Ethereum-based networks. Difference between various networks is explained in the next chapters. For now, the mainnet network is live Ethereum network where users spend real money for their transactions. Testnets are networks created mainly for testing purpose [25].

3.1.2 Remote client

Remote clients do not store blockchain locally or validate any transactions and blocks. This is the main difference against full node clients. Remote clients offer a subset of full node clients functionality. Remote clients mostly create transactions and broadcast them. They are used interchangeably with the term "wallet„. In practice, most of the wallets which intended to be used by ordinary users are remote clients. Also all mobile wallets are remote clients, because phone hardware cannot satisfy required hardware resources for running full node [25]. Within this paper I went through the most popular wallets and summarize their main difference in several tables which are listed in the next sections. These wallets are Ledger Nano[1], Trezor[2], Atomic[3], Exodus[4], Mist[5], Jaxx[6], MyEtherWallet[7], MetaMask[8].

3.2 Nowadays wallets

Nowadays popular wallets are divided into three types: hardware wallets, desktop/mobile wallets, and web-based wallets [24]. From a user perspective wallets are managing keys and addresses, tracking the balance on addresses and create or sign transactions. Some Ethereum wallets can interact with smart contracts and manage ERC20 tokens. A user then selects between wallets witch dispose of additional features or support required underlying OS platforms which are shown in Table 3.1. Some wallets support more cryptocurrencies and can manage more than just Ethereum tokens but also Bitcoins or others.

	Unix version	Windows version	OSX version	Android version	iOS version	Chrome extension	Firefox extension	HW wallet	Web based
Ledger Nano								✓	
Trezor								✓	
Atomic	✓	✓	✓	✓	✓				
Exodus	✓	✓	✓						
Mist	✓	✓	✓						
Jaxx	✓	✓	✓	✓	✓	✓			
MyEtherWallet									✓
Coinbase									✓
MetaMask						✓	✓		

Table 3.1: Ethereum wallets with their supported underlying OS platforms.

[1] https://www.ledger.com/
[2] https://trezor.io/
[3] https://atomicwallet.io/
[4] https://www.exodus.io/
[5] https://github.com/ethereum/mist
[6] https://jaxx.io/
[7] https://www.myetherwallet.com/
[8] https://metamask.io/

3.2.1 Hardware wallets

Hardware wallet is a literally small piece of hardware, typically contains a small display. These wallets just store encrypted private keys and make access only to the owner's wallet. Some wallets also offer to create a transactions and more features. The most popular of them are listed in Table 3.2.

	Ether	ERC20	Other tokens
Ledger Nano S	✓	✓	✓
Trezor	✓	✓	✓

Table 3.2: Supported tokens by popular hardware wallets.

3.2.2 Desktop/mobile wallets

All wallets in the following Table 3.3 are desktop wallets. These wallets support more or less desktop or mobile platforms like Linux, Windows, iOS and others. They always provide storing private encrypted keys in user's device and provide at least basic functionalities. A web browser extension wallets are also considered as desktop wallets and usually storing data within users browser.

One of the most popular wallets is MetaMask which is a browser-based extension and provide switching between Ethereum-based networks as mainnet or testnets. It supports smart contract exploring and is available as Chrome, Firefox, Opera, and Brave Browser extensions [17]. Another popular desktop wallet is Jaxx, shown in Table 3.1. This wallet support multiple currencies and multiple OS and browser platforms. It is based on BIP-39 mnemonic seeds described in Chapter 3.3.3 [16]. Mist is official Ether wallet created by Ethereum Foundation. Mist runs a full node and offers a full smart contract browser, however, during writing this paper Mist become deprecated [18].

	Ether	ERC20	Other tokens
Mist	✓	✓	
MetaMask	✓	✓	
Jaxx	✓	✓	✓
Atomic	✓	✓	✓
Exodus	✓		✓

Table 3.3: Supported tokens by popular desktop wallets.

3.2.3 Web-based wallets

A community around cryptocurrencies is discussing if the web-based model has any place in the decentralized world for one fundamental reason. When a user interacts with server-based

wallets, it is centralized and the user has to make sure if service provider as a middleman is trusted. A whole peer-to-peer model can be corrupted and advantages lost. Phishing attacks are often a problem and user has to make sure if he working with an official web-based wallet. This bring us back to the classical centralized payment system model.

However, there are popular solutions which are accepted also by the community. MyEtherWallet generates wallet and stores encrypted private keys on a user's machine within keystore file. The user manages this keys by himself. Keys and password from these keys are needed to create transactions and a user interacts with them. Newest version of MyEtherWallet can be downloaded as a web page to local machine which increases the security level against phishing [19].

Another popular wallet and *exchange*[9] is Coinbase. Coinbase is good for storing Ether for short term, for example for exchanging for fiat money or other tokens. A big disadvantage is that private keys are under third-party control [15].

These wallets are shown in Table 3.4.

	Ether	ERC20	Other tokens
Coinbase	✓	✓	✓
MyEtherWallet	✓	✓	✓

Table 3.4: Supported tokens by popular web-based wallets.

3.3 Technical perspective

Wallets can be considered as a container or keychain of private keys. It is a system which generates, store and manage these keys. There are approaches in Ethereum network when a new key and address is generated for every transaction which is hard to track and increase privacy in the network. Such an approach is difficult to manage and wallets must maximize simplicity for users. For ensuring this behavior there are two primary types of wallets - nondeterministic and deterministic wallets.

3.3.1 Nondeterministic (Random) Wallets

Nondeterministic wallet generate private keys which are not related on each other. Every single key is picked independently as a new random number. Then every new private key has to be stored in a new file or in the growing list of keys. Such approach deals with the problem of regular backups. Loss of newly created private key without backup means loss of access to funds.

Nondeterministic wallets usually use JSON-encoded file for every private key which is considered as a keystore file. Keystore like this uses key derivation function (KDF)[10], which protects against brute-force, rainbow table, and dictionary attacks.

[9] exchange - online place where users can exchange fiat money for digital assets like Bitcoin or Ether. Exchanges are mostly centralized web portals, however, there is an effort to create decentralized exchanges.

[10] KDF - known as a password stretching algorithm, where a private key is not encrypted at all but repeatedly hashed [33].

However, nondeterministic wallets have a place for historical reasons and usually marked as JBOK[11]. Nowadays wallets use the concept of deterministic wallets [25].

On Listing 3.1 is shown JSON-encoded keystore file that contains a single (randomly generated) private key, encrypted by a passphrase for extra security. The parameter `crypto.kdfparams.n` shows how many times was passphrase repeatedly hashed.

```
{
    "address": "001d3f1ef827552ae1114027bd3ecf1f086ba0f9",
    "crypto": {
        "cipher": "aes-128-ctr",
        "ciphertext": "233a9f4d236ed0c13394b504b6da5df0 \
                       2587c8bf1ad8946f6f2b58f055507ece",
        "cipherparams": {
            "iv": "d10c6ec5bae81b6cb9144de81037fa15"
        },
        "kdf": "scrypt",
        "kdfparams": {
            "dklen": 32,
            "n": 262144,
            "p": 1,
            "r": 8,
            "salt": "99d37a47c7c9429c66976f643f386a61 \
                     b78b97f3246adca89abe4245d2788407"
        },
        "mac": "594c8df1c8ee0ded8255a50caf07e8c1 \
                2061fd859f4b7c76ab704b17c957e842"
    },
    "id": "4fcb2ba4-ccdb-424f-89d5-26cce304bf9c",
    "version": 3
}
```

Listing 3.1: Encrypted keystore file (adapted from [25]).

3.3.2 Deterministic (Seeded) Wallets

Deterministic wallets or Hierarchical Deterministic (HD) Wallets generate private keys which are related on each other. They use derivation method based on a tree-like structure described as BIP-32 [1]. These wallets can derive other private keys from one single master key by hashing it. A master key is a seed - randomly generated number with another piece of data for example chain of chars. Seed is considered as sufficient for migration wallet and recovers all the derived private keys. Also, only one backup after wallet initialization is needed. However, the disadvantage is that an attacker can focus effort to break security only on a single piece of data. Hierarchical Deterministic Wallet is show in Figure 3.1. All sequence of child keys and grandchild keys can be used for different purposes as calculating addresses for incoming or outgoing payments [25].

[11] JBOK - Just a bunch of keys.

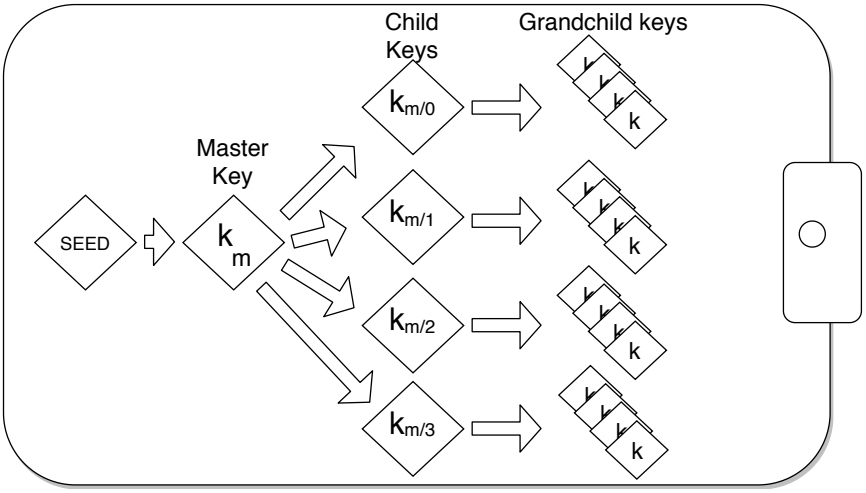


Figure 3.1: Hierarchical deterministic wallet (adapted from [1]).

3.3.3 Widely adopted standards

Cryptocurrency wallet software started to share some main aspects as standards were developing. Some of these standards become widely adopted and currently they are used as common by a broad range different wallets across many cryptocurrencies [25] [24].

BIP-39

Standard BIP-39 [2] describes the implementation of a mnemonic code which is a group of easy to remember words. It is widely adopted standard between many cryptocurrencies which can be used for importing and exporting seeds for backup and recovery [24].

Example of seed in hex:

```
8cfa96d0d9f7d1dde1a2aa39fc6d916d882e7406f6878fbfb2310671e83ed918
843844a23e3ae7b6a695a346c981484b554ff1718299b0b42df3045f04b94f05
```

Example of a BIP-39 seed for a deterministic wallet using 12-word mnemonic code:

```
omit speak giant bright enable increase
tube worth object timber bleak bullet
```

A mnemonic code can be used to recreate the original seed and recreate the wallet with all derived keys. BIP-39 do not implement brain wallets. Brain wallets are based on user-chosen words, whereas BIP-39 describe how to randomly generate this word within wallet and present to the user. This process is shown in Figure 3.2. More about generating mnemonic sequence is written in official standard deffiniton [2].

BIP-32

BIP-32 [1] standard describes a hierarchical deterministic wallet mentioned in Chapter 3.3.2. These wallets generate deterministic keys based on tree-like hierarchical relationships. BIP-32 allows to use one master private key generate other derived private or public keys for a different purpose. Any private or public key in a tree can become a parent key for its child

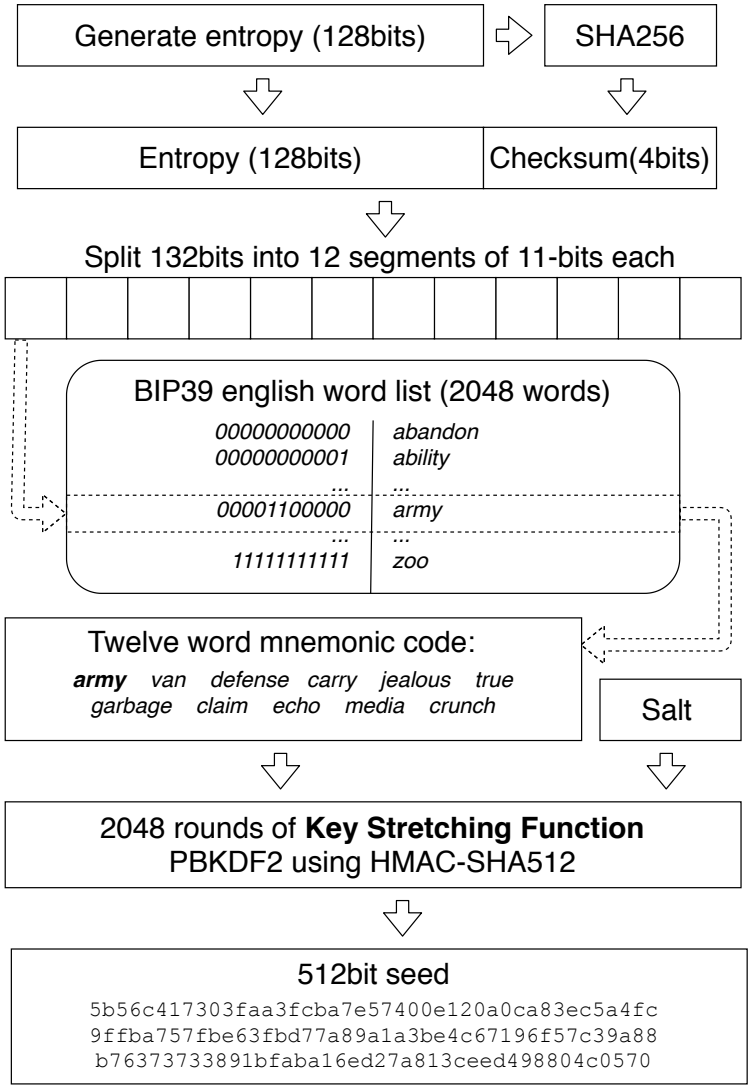


Figure 3.2: Generating mnemonic words and seed based on BIP-39 setandard (adapted from [25]).

keys. Public and private keys become parents by extending them with prefix *xpub* or *xpriv* respectively [1]. There is a relationship between child branches and parent branches which is a potential vulnerability. Therefore, BIP-32 also define *hardened* child derivation which is optional and create a firewall in such keys sequence [25].

Generation of private keys is based on this tree aspects: Parent key, special child derivation function, and index number. An index is 32-bit integer number and helps to manage desirable derivation. To distinguished between unhardened and hardened keys this number was split into two ranges. All numbers between 0 and $2^{31} - 1$ is considered for unhardened derivation. All numbers between 2^{31} and $2^{32} - 1$ are used only for hardened derivation [1].

To make indexes easier to read all indexes for unhardened child keys are just number counted from 0 and indexes for hardened derivation are numbers from 0 with prime symbol suffix '. First index for hardened derivation is 0', second is 1' i index is i' which means 2^{31} + i [25].

BIP-32 also define a path naming convention. That ensures identifying keys in tree structure with every level separated by sign /. Public keys are marked as M. Private keys are marked as m [1].

The following Table 3.5 makes naming convention easier to understand:

HD path	Key description
m/0	The first child private key of the master private key (m)
m/0'/1	The second normal grandchild of the first hardened child (m/0')
m/2/0	The first grandchild private key of the third child (m/1)
M/5/45/3/1	The second great-great-grandchild public key of the 4th great-grandchild of the 46th grandchild of the 6th child

Table 3.5: BIP-32 naming convention examples (adapted from [24]).

BIP-43

BIP-43 [3] introduces a "Purpose Field„ in a tree structure of naming convention based on BIP-32. It means that the first level hardened child index is purpose file:

```
m / purpose' /...
```

Practically, HD wallet will always use branch m/i'/..., where i is a number of purpose field. A particular application may be the usage of another popular standard BIP-44 which many wallets implements. Such wallets always use m/44&#x27/... branch of the tree where 44&#x27 signifies usage of BIP-44 standard [25].

BIP-44

BIP-44 [4] proposes a logical hierarchy for HD wallets based on BIP-32 and purpose scheme BIP-43. More specifically, BIP-44 contains five predefined tree levels:

```
m / purpose' / coin_type' / account' / change / address_index
```

Individual fields are explained in the following Table 3.6. Apostrophe indicates BIP-32 hardened derivation.

tree level	level meaning	description
level 0	m/M	type of key private/public based on BIP-32
level 1	purpose'	always 44&#x27 as it is a particular application of BIP-43
level 2	coin_type'	type of cryptocurrency defined within - SLIP-0044 [23] e.g. Ethereum: 60', Bitcoin: 0'
level 3	account'	allow to separate subaccounts logically for example in one organization
level 4	change	whether the address is receiving address or change address (only in Bitcoin). In Ethereum this field has always value 0 (receiving addresses).
level 5	address_index	numbered address sequence from 0 with increasing manner. Used as child index in BIP-32 derivation.

Table 3.6: BIP-44 fields (adapted from [25] and [24]).

Practical example based on the previous table could be:

`M/44&#x27;/60&#x27;/0&#x27;/0/4`

Which basically means the 5th receiving public key for the primary Ethereum account.

Chapter 4

Keys and Addresses

This chapter reveals the principles of generation keys and addresses for Ethereum. In later chapters, most of these principles are used as part of the wallet implementation. The math behind these principles use not only Ethereum but also Bitcoin or other cryptocurrencies. However, they can differ in some cases.

4.1 Ethereum cryptography overview

Ethereum wallets use asymmetric cryptography (also known as public key cryptography - PKC). In such systems, keys come in pairs consisting of a private (secret) key and a public key which are derived from private key.

Private keys should always remain secret because who owns the private key is also an owner of Ethereum account which means that such user can create authorized digital signatures for any transaction and spend funds. Therefore private keys are mostly stored in encrypted form in special files and managed by wallet code.

If a randomly picked number is private key(k), then elliptic curve multiplication, which is a one-way cryptographic function, generates a public key(K) from k. Ethereum address(A) can be generated from the public key(K) using another one-way cryptographic hash function. The process of generation public keys and Ethereum addresses[1] is shown in Figure 4.1 [25].

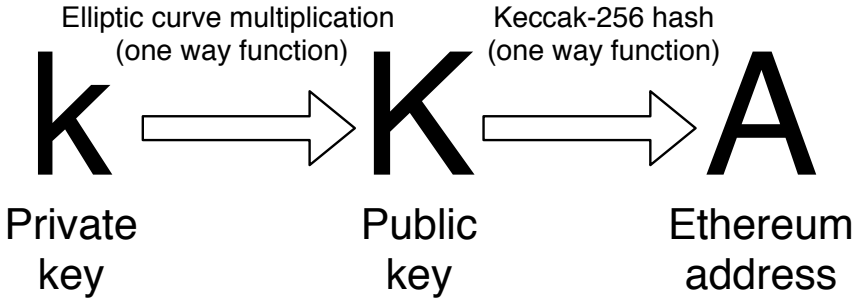


Figure 4.1: Ethereum address generation scheme.

[1] Other cryptocurrencies use similar scheme (e.g. Bitcoin uses the same addresses generation as Ethereum but instead hashing public key by Keccak-256 function it uses A = RIPEMD160(SHA256(K))) [24].

4.1.1 Elliptic curve

Elliptic curve cryptography (ECC) is asymmetric cryptography method based on the discrete logarithm problem as expressed by addition and multiplication on the points of an elliptic curve [24] [25]. Ethereum and Bitcoin use specific elliptic curve standard called secp256k1 [28] for generation public keys.

Elliptic curve cryptography is an important invention and lot of cryptocurrency-related projects can use it. Therefore implementation of Secp256k1 occurred in libraries as OpenSSL [20] or C-language Bitcoin Core's libsecp256k1 [13].

4.1.2 Keccak-256

Keccak-256 [12] is implemented as a hash function algorithm and it is used in many places in Ethereum system or Ethereum wallets.

4.2 Private keys

Private keys as mentioned before are used for controlling Ethereum accounts and creating digital signatures. Every key must be first generated and stored, it can be achieved with multiple methods.

Generating a private keys

Ethereum private keys are simply random numbers and usually generated by wallet software. A wallet should use the best possible approach of picking a secure source of entropy, or randomness to find a truly random number. It is on wallet's developers which picking method they use. Private key creation should not be predictable or deterministic.

A private key is a numbers between 1 and 2^{256}. Wallets usually use an underlying operating system's random generator to produce a random 256-bit random number. 2^{256} private key space means an unfathomably large number. This generation is combined with a source of user randomness as noise from a microphone or let the user move a mouse for few seconds or type random keys. Generation process can be executed offline, Ethereum network has no effect for producing private keys. Pseudorandom generator function of random numbers, which is provided by most programming languages are not recommended. In practice, wallets use cryptographically secure pseudo-random number generator (CSPRNG) which use sufficient entropy seeds [25].

Example of random generated 256-bit private key formatted in 64 hexadecimal digits:

```
a18f998097dd14034f42dd6c4d22808e2d649fb79ff6ddba3111d0d50971841d
```

Storing a private keys

Private and public keys are mostly stored as pairs in encrypted form. However, calculation of public key is trivial and therefore private keys can be stored alone. Such an example is shown on Listing 3.1.

4.3 Public keys

Figure 4.1 shows that a public key is generated thanks to an elliptic curve multiplication from a private key which is practically irreversible one-way function [29].

Generating a public keys

Generating public key can be expressed as:

$$K = k \times G \tag{4.1}$$

Where k is a private key, G is generator point specified as part of the secp256k1 standard, operator $\times$ is special one-way elliptic curve multiplication (reverse operation is as difficult as brute-force search algorithm to find all possible values of k), K is a point on an elliptic curve which is the corresponding public key. This public key K can be expressed as a set of axis $K = (x, y)$.

Generator point G is always the same, because of that, one private key will always generate the same public key using elliptic curve multiplication [25].

The following example shows a public key calculation:

```
K = a18f998097dd14034f42dd6c4d22808e2d649fb79ff6ddba3111d0d50971841d * G
```

That operation created point $K = (x, y)$ where:

```
x = 797d46d4ff54c5368e2a2cf8d03fc6f058c79741a66b8ebd8a7d459d724b6039
y = 0d2989a6af744bd96792aae86c421fb2167bb29e66a06e31428dca96130c882b
```

Formatting a public keys

A public key $K = (x, y)$ is usually serialized by x and y into one number which creates 130 hexadecimal characters. Ethereum as a lot of other cryptocurrencies uses standard SEC1 [28]. This standard defines four possible prefixes how can be identified the point on an elliptic curve shown in Table 4.1.

Prefix	Meaning	Length (bytes counting prefix)
0x00	Point at infinity	1
0x02	Compressed point with even y	33
0x03	Compressed point with odd y	33
0x04	Uncompressed point	65

Table 4.1: Serialized elliptic curve public key prefixes (adapted from [25]).

For the Ethereum, only relevant prefix is 0x04 because Ethereum uses only uncompressed public keys. Next example shows how public key looks like after concatenation of x, y and SEC1 prefix as 0x04 + x-coordinate (64 hex) + y-coordinate (64 hex) from previous example:

```
04797d46d4ff54c5368e2a2cf8d03fc6f058c79741a66b8ebd8a7d459d724b603
90d2989a6af744bd96792aae86c421fb2167bb29e66a06e31428dca96130c882b
```

4.4 Addresses

Ethereum address is a unique identifier of Ethereum's account. An address is calculated with Keccak-256 one-way hash function. When the hash of the public key is calculated, last 20 bytes, least significant bytes, representing Ethereum address [33].

The public key (public key is not formated with prefix 0x04 before address calculation):

```
K = 797d46d4ff54c5368e2a2cf8d03fc6f058c79741a66b8ebd8a7d459d724b60390d2989...
```

Keccak-256 hash of public key K in hexidecimal form:

```
Keccak256(K) = a54e53712efd827be401672cfe7ead4d9bed0f1ac2b9f7d8910d8717b505db4c
```

Final Ethereum address is:

```
fe7ead4d9bed0f1ac2b9f7d8910d8717b505db4c
```

Which is last 20 bytes of Keccak246(K) function.

Formating Ethereum addresses

Ethereum addresses are just raw hexadecimal numbers. Some other cryptocurrencies like Bitcoin use for example a built-in checksum to protect against mistyped addresses. However, Ethereum uses encodings like ICAP[2] or EIP-55[3] to improve addresses representation [25].

[2]ICAP - Inter exchange Client Address Protocol is an Ethereum address encoding protocol compatible with IBAN (International Bank Account Number) encoding. ICAP is supported only by a few wallets [25].

[3]EIP-55 - Ethereum Improvement Proposal 55 [7] is mixed-case checksum address encoding for Ethereum addresses. Adoption of EIP-55 by wallets is high.

Chapter 5

Transactions

All transactions in the Ethereum network are simply signed messages by an account or smart contract and recorded onto Ethereum blockchain permanently. Another point of view on transactions may be that they trigger changes of the global singleton state machine which is Ethereum.

5.1 Ether currency units

Build-in currency unit in the Ethereum network is called **ether**. Ether is also subdivided into smaller units where the smallest one is named **wei**. One ether is $1 * 10^{18}$ of wei. Wei is important as it is used as part of the transactions and most of the values must be converted to wei value. That means, when user sends 1 ether, he is actually sending 1 quintillion wei [25].

5.2 Gas

Ethereum, in general, is Turing complete so it can compute any program of any complexity. However, this fact can be dangerous in open access systems because of the halting problem. Every participating node in the Ethereum network has to validate and run smart contracts which produce transactions. This can result in resource management problems as a smart contract can possibly run forever in an infinite loop. Wheater a program runs in an infinite loop by an accident or as a DoS attack, Ethereum protects itself with a metering mechanism called **gas**.

As Ethereum virtual machine executes smart contract's code it accounts every single instruction. These instructions have some costs. Gas is a unit which can be purchased with ether and must be included in every payment transaction or a transaction which triggers smart contract. Gas sets a limit of maximum cost which can be spent by CPU resources, memory resources etcetera. Simply, gas is a mechanism which limits resources consumption and Ethereum virtual machine and stops executing code when this limit is exceeded [25].

5.3 Transaction structure

Every transaction in the Ethereum network has the following structure shown in Table 5.1 and defined in [33]:

title	purpose
Nonce	A sequence number, issued by the Ethereum account
Gas price	The price of gas (in wei) the originator is willing to pay
Gas limit	The maximum amount of gas the originator is willing to buy for this transaction
Recipient	The destination Ethereum address
Value	The amount of ether to send to the destination
Data	The variable-length binary data payload
v,r,s	Components of an ECDSA digital signature of the originating Ethereum account

Table 5.1: Transaction structure (adapter from [33]).

Transaction nonce

The nonce is a scalar value equal to the number of transactions sent from one unique address [33]. It is useful in a sense of making every single transaction unique. It is used for example for preventing message replay. Transaction nonce starts counting from zero and it is an up-to-date count of confirmed transactions. It is also important that software wallets are tracking the nonce and checking if they were confirmed. There could be created several transactions but if first is never delivered to a node for various reasons all other transactions will be waiting for the transaction with the first nonce. The wallet should recreate this transaction and allow all others to be confirmed and sent to the network. It is also important to know that when one of these missing transactions are recreated, all others broadcasted transaction with higher nonce will incrementally become valid and we cannot recall these transactions back.

Another view could be if the nonce is duplicated. In that case, the first transaction received by a node will be confirmed and second rejected [25].

5.4 Digital Signatures

A digital signature is a mathematical scheme which allow us to prove the ownership of a private key. Ethereum digital signature algorithm is based on the Elliptic Curve Digital Signature Algorithm (ECDSA) which uses Elliptic cure mentioned in Chapter 4.1.1. ECDSA use private-public key pairs. In fact, there are two algorithms. One for signing the transactions, second for verifying the signature [25].

Signing the transactions

ECDSA create a digital signature with the private key of Ethereum account and hashing RLP-encoded [22] transaction with Keccak256. The formula for creating a digital signature is described in [25]:

$$Sig = F_{sig}(F_{keccak256}(m), k) \tag{5.1}$$

where:

- k is the signing private key.
- m is the RLP-encoded transaction.
- $F_{keccak256}$ is the Keccak-256 hash function.
- F_{sig} is the signing algorithm.
- Sig is the resulting signature.

The function F_{sig} produces a signature Sig that is composed of two values, commonly referred to as signature values r and s refered in Table 5.1:

$$Sig = (r, s) \tag{5.2}$$

In Table 5.1 is also signature variable v which indicates the chain ID and the recovery identifier. The chain identifier is useful to identify to which Ethereum network was single transaction sent and therefore not valid on another blockchain. The chain identifiers are listed in Table 5.2. More information about chain identifiers or recovery identifiers is specified in [6].

id	chain
1	Ethereum mainnet
2	Morden (disused), Expanse mainnet
3	Ropsten
4	Rinkeby
5	Goerli
47	Kovan
1337	Geth private chains (default)

Table 5.2: Chain identifiers (adapted from [6]).

Verifying the transactions

Everyone in Ethereum network who must or who want to verify transaction can use signature verification algorithm which takes the hash of the transaction, the signer's public key and $Sig = (r, s)$. Verification is the inverse of the signature creation function without revealing the private key of the owner. However, ECDSA is quite a complex piece of math and cryptographical libraries provide us powerful interface to use it easily. For a deeper understanding of ECSSA math and signing or verifying messages see [29].

Chapter 6

Specification and design

Based on all theoretical concepts described in previous chapters the following wallet specification and design was created. This chapter can be considered as full specification which implementation is described in the next Chapter 7.

6.1 Specification

The wallet run as remote Etherum client which does not download the whole blockchain but is connected to the Ethereum network via one of the participating nodes. In this case, it will be Infura[1]. The wallet is following some of the base standards and the best practices described in Chapter 3. This wallet is a combination of nondeterministic wallet described in Chapter 3.3.1 and deterministic wallet described in Chapter 3.3.2. It is because of storing master private key within the single encrypted keystore file as nondeterministic wallets do. On the other hand, there is a deterministic key generation and restoring private key from a mnemonic sentence which follows BIP39 standard mentioned in Chapter 3.3.3.

This wallet is supporting:
EIP-55 - addresses in hex encoding with checksum in capitalization.
BIP-39 - the private key (seed) encoded and restorable based on mnemonic code.
ERC-20 - token standard for Ethereum smart contracts, known as custom tokens.
KDF - key derivation function, known as password stretching algorithm against attacks.
CMD - command line interface to control wallet functionalities.
GUI - graphical user interface to control wallet functionalities.

In the Ethereum world, Python is one of the most supported programming languages with active libraries maintainers and contributors. Therefore, I have used the Python programming language for the development of this wallet. The wallet is using some of the most popular official libraries developed by Ethereum Foundation which stands for an original reference implementation of Ethereum's protocols. More about used libraries see Chapter 6.3.

[1] https://infura.io/

6.2 Design

In Figure 6.1 is shown wallet architecture which is abstraction reflecting python source code and modules.

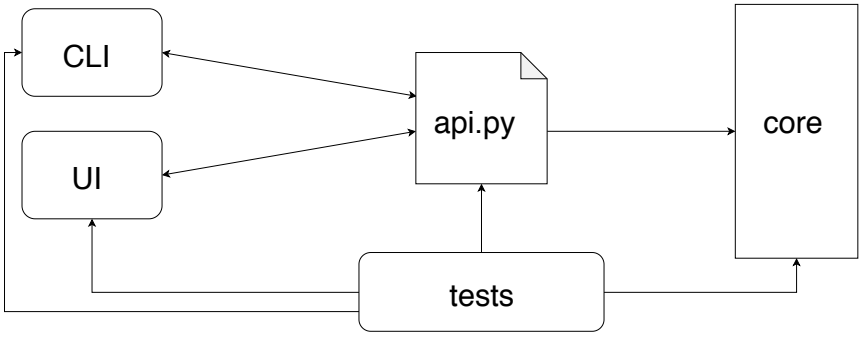


Figure 6.1: Wallet architecture.

core

This element stands for all core classes and modules in the implemented wallet. That means that the core consists of functionalities responsible for account creation, keys and addresses manipulation, transactions, transactions signing and many more.

api.py

Functions of api.py directly call wallet core functionalities and try to proceed them. Source file **api.py** is meant to be an application programming interface which will be called by other modules as **CLI** or **UI**. This interface is considered to be middleware between wallet core functionalities and users interactions.

CLI

This module contains tools for wallet interaction from the command line. The **CLI** module is useful mainly for testing and development purpose. It is responsible for handling with the command line input and calling **api.py** functions to make the wallet works also in the terminal.

UI

A module which creates user interface for the desktop user. Similar to the **CLI** module it is calling **api.py** functions to make wallet features available to the user.

tests

Tests are simply verifying functionality of the wallet functions and are based on PyTest[2] framework.

[2] https://docs.pytest.org/en/latest/

6.3 Libraries

eth-account

Account abstraction library for creating private keys and addresses for Ethereum network. Library also provides methods for encryption and decryption of these keys which allow us to store them on local machine's disk space. Library also provides methods for signing transactions or message hashes. The library provides functionaligy to generating private keys based on CSPRNG of an operating system. Then it uses elliptic cure algorithms to produce a public key and keccak-256 hash algorithm to produce an Ethereum address as it is described in Chapter 4. A reference implementation of the library is available on public Github account of Ethereum Fundation[3] and distributed under the MIT license.

eth-keys

A common API for Ethereum key operations. A reference implementation of the library is available on public Github account of Ethereum Fundation[4] and distributed under the MIT licence.

web3.py

A python interface for interacting with the Ethereum blockchain and ecosystem. One of the biggest advantages is that the library allows us to manipulate with smart contracts in general and trigger smart contract's functions. But it can be used for a simple communicating with the blockchain via API for example querying for address balance. This library is inspired by popular library for Javascript web3.js. A reference implementation of the library is available on public Github account of Ethereum Fundation[5] and distributed under the MIT license.

python-mnemonic

The last library which is worth to mention is python-mnemonic[6] which is the reference implementation of BIP-39 mnemonic code for generating deterministic keys described in Chapter 3.3.3. Library is distributed under the MIT license.

Other used libraries and their licenses

Click
Python composable command line interface toolkit[7] distributed under BSD license.
PyYAML
YAML parser and emitter for Python[8] distributed under MIT license.

[3] https://github.com/ethereum/eth-account
[4] https://github.com/ethereum/eth-keys
[5] https://github.com/ethereum/web3.py
[6] https://github.com/trezor/python-mnemonic
[7] https://github.com/pallets/click/
[8] https://github.com/yaml/pyyaml/

Chapter 7

Implementation

All chapters above contain the necessary theory for the creation of a wallet which is based on community standards and the best practices. Within this chapter, the practical implementation of such a wallet is presented.

7.1 Wallet implementation and functionalities

Within this section, there is a description of how the wallet is implemented and which functionalities are necessary for wallet users. Most of the theoretical knowledge of earlier chapters is used for creating wallet keys and addresses, querying for wallet balances or sending transactions and much more. Primary user's access to the wallet functionalities was designed to use a command line interface.

7.1.1 Manipulation with keys

First of all, the user should be able to create a new wallet to access all other functionalities. Command `eth-wallet new-wallet` will create new private and public key with corresponding Ethereum address. The private key is generated from the 512-bit seed of BIP-39 standard which also generates a mnemonic sentence to restore the wallet. Process of creation mnemonic sentence and seed is shown in Figure 3.2.

```
$ eth-wallet new-wallet
  Passphrase from keystore:

Account address: 0xB1f761734F00d1D368Ce6f82F755bBb3005538EB
Account pub key: 0xf94e03524a1bd803ee583a1f0de7eb1eb67a90d680
                 2eeac22b90cfdd7ff491039441472e8db543467c045
                 0d1b7c31b5e8f81616b99226775770f9dd531afd31a
Keystore path: /Users/Marek/.eth-wallet/keystore
Remember these words to restore eth-wallet: omit speak giant bright
                                            enable increase tube worth
                                            object timber bleak bullet
$
```

Listing 7.1: Creation of a new wallet.

After the master private key is created it is stored as an encrypted file similar as it was shown on Listing 3.1. As we can see on Listing 7.1, the location of the encrypted file is in

the user's home directory within `.eth-wallet/keystore` by default.

After the wallet creation, user can ask for its details by running command `eth-wallet get-wallet`. This process is shown on Listing 7.2

```
$ eth-wallet get-wallet

Account address: 0xB1f761734F00d1D368Ce6f82F755bBb3005538EB
Account pub key: 0xf94e03524a1bd803ee583a1f0de7eb1eb67a90d680
                 2eeac22b90cfdd7ff491039441472e8db543467c045
                 0d1b7c31b5e8f81616b99226775770f9dd531afd31a
$
```

Listing 7.2: Show wallet information.

If the master private key has been lost it is possible to repair wallet from the mnemonic sentence shown on Listing 7.3 using command `eth-wallet restore-wallet`. **It is important to mention that generating a master private key from BIP39 seed is not fully compatible with other wallets since this implementation does not follow BIP32 standard and therefore only this implementation of the wallet can reproduce the very same private key from BIP39 seed**.

However, if the user has access to the keystore file and passphrase, he can restore wallet from this keystore, for example, using MetaMask wallet mentioned in Chapter 3.2.2.

```
$ eth-wallet restore-wallet
  Mnemonic sentence []: omit speak giant bright enable increase
                        tube worth object timber bleak bullet
  Passphrase:

Account address: 0xB1f761734F00d1D368Ce6f82F755bBb3005538EB
Account pub key: 0xf94e03524a1bd803ee583a1f0de7eb1eb67a90d680
                 2eeac22b90cfdd7ff491039441472e8db543467c045
                 0d1b7c31b5e8f81616b99226775770f9dd531afd31a
Keystore path: /Users/Marek/.eth-wallet/keystore
Remember these words to restore eth-wallet: omit speak giant bright
                                            enable increase tube worth
                                            object timber bleak bullet
$
```

Listing 7.3: Restore wallet.

The last useful command is `eth-wallet reveal-seed` which reveals master private key to the user. Therefore the user is able to move his funds into any other wallet or device. Revealing the master private key is shown on Listing 7.4.

```
$ eth-wallet reveal-seed
  Password from keystore:

Account prv key: 0x843844a23e3ae7b6a695a346c981484
                 b554ff1718299b0b42df3045f04b94f05
$
```

Listing 7.4: Reveal master private key.

7.1.2 Token management

When a new wallet is created the user obtain automatically access to the Ethereum blockchain. That basically means he can check his account's ether balance by running command `eth-wallet get-balance` shown on Listing 7.5 and send this funds into another address in the Ethereum network by running command `eth-wallet send-transaction` shown on Listing 7.6. Within this section is also shown how to add ERC-20 smart contract token to the wallet and how to send such tokens to another user. Now let's look closer on these functionalities and what implementation is hidden under cover of command's level abstraction.

Query for balance

Reading account's balance isn't a difficult process to do at all. It is a simple query to the Ethereum blockchain. In practice, most of the third-party software use the web3.py library described in Chapter 6.3. Web3.py sends the rest API query and parses JSON-RPC response from the connected node. In the Ethereum world, everything written onto blockchain is written in the smallest ether unit which is **wei** mentioned in Chapter 5.1. Therefore, when we obtain from blockchain that address's balance is 1000000000000000000 it is exactly 10^{18} of wei which means 1 ether.

```
$ eth-wallet get-balance
Balance on address 0xB1f761734F00d1D368Ce6f82F755bBb3005538EB is: 1.234ETH
$
```

Listing 7.5: Show user's balance.

Sending simple transaction

Sending transactions is a little bit more complicated than reading an account's balance. In this case, the wallet needs to know if the user wants to send just ordinary ether to another account or communicating with smart contracts, for example emit one of its functions. Such a function could be a transfer function to send ERC-20 tokens from account to account.

```
$ eth-wallet send-transaction
  To address:  []: 0xAAD533eb7Fe7F2657960AC7703F87E10c73ae73b
  Value to send:  []: 0.01
  Password from keystore:

transaction: {'to': '0xAAD533eb7Fe7F2657960AC7703F87E10c73ae73b',
```

```
                'value': 10000000000000000,
                'gas': 21000,
                'gasPrice': 20000000000,
                'nonce': 0,
                'chainId': 3}
Pending.................
Transaction mined!
Hash of the transaction: 0x193919d1ad2dc024349ccc035a15a697
                         987bd33e1ff04e33f878e6f89f2ebbdf
Transaction cost was: 0.00042ETH
$
```

Listing 7.6: Send transaction.

Listing 7.6 shows the creation of ordinary transaction for sending ether from one account to another. User must insert Ethereum address where he is intended to send funds and amount of ether to send. After that, the transaction structure described in Chapter 5.3 is created and shown to the user. Here we can see again that field `value` contains converted 0.01 ether to $0.01 * 10^{18}$ which is the amount of wei to send. The same behavior was implemented when we wanted to read the balance from the blockchain. The field `gas` representing gasLimit also described in Chapter 5.3 and is always fixed 21000 gas units per ether transaction from one account to another. This value is different in most of the cases when the wallet needs to communicate with the smart contract and for example send ERC-20 tokens. `gasPrice` field is obtained from the blockchain and multiplied by the wallet to make the transaction faster. Other two fields `nonce` and `chainId` shows us that it is very first transaction from this wallet's address and the transaction was sent on Ropsten testnet network defined in Table 5.2.

After the transaction is created it is signed with the help of web3.py library and sent to the node which distributes this transaction into entire Ethereum network. When the transaction is for the first time added to the block and block added into blockchain by miner it is considered as mined. User's wallet can then query for the transaction's receipt from the node. This process can take a few seconds depending on field `gasPrice`. Receipt from the mined transaction contains also hash.

Mined transaction hash example:

`0x193919d1ad2dc024349ccc035a15a697987bd33e1ff04e33f878e6f89f2ebbdf`

This hash is shown to the user and means that transaction was published onto blockchain permanently. We can lookup for this transaction in one of the Ethereum blockchain explorers[1].

Last important thing to mention is the transaction cost. It is a fee for the miner who included transaction into a block and added this block into blockchain. This fee is calculated by multiplying `gas` with `gasPrice` and converted to the ether. This cost is then deducted from transaction originator's balance for the miner. All this information is again accessible using Ethereum blockchain explorers[1].

[1]Transaction from the Listing 7.6 permanently published onto blockchain and available at https://ropsten.etherscan.io/tx/0x193919d1ad2dc024349ccc035a15a697987bd33e1ff04e33f878e6f89f2ebbdf

Working with ERC20 tokens

As it was described in Chapter 2.3, smart contracts are special digital contracts similar to real-world legal agreements. When a smart contract is deployed onto blockchain anyone who wants to interact with its functions via transactions must follow smart contract's rules. Every transaction is then verified by all nodes in the network and published onto blockchain. All Ethereum users can issue their own tokens by writing their smart contracts. To make this issuing clever and simple developers of Ethereum created ERC20 standard which describes how to create such a token. ERC20 tokens also guarantee that all wallets which support this standard can work with all tokens which are following the ERC20. For the purpose of testing this wallet, I created my own ERC20 token called FITCOIN[2] which is deployed on Ropsten testnet network. FITCOIN with symbol FIT has no value at the moment and there are 1000000 tokens issued. FITCOIN can by fund on the blockchain by using Ethereum blockchain explorer[2] but users of the wallet can add and manipulate with any other ERC20 token. Programming smart contracts are out of the scope of this paper and it is not covered. However, this wallet can work with any ERC20 tokens and following examples shows how.

```
$ eth-wallet add-token
  contract address []: 0x19896cB57Bc5B4cb92dbC7D389DBa6290AF505Ce
  Token symbol []: FIT
New coin was added! FIT 0x19896cB57Bc5B4cb92dbC7D389DBa6290AF505Ce
$
```

Listing 7.7: Add custom ERC20 token to the wallet.

```
$ eth-wallet get-balance --token FIT
Balance on address 0xB1f761734F00d1D368Ce6f82F755bBb3005538EB is: 1.0FIT
$
```

Listing 7.8: Show wallet's balance of custom ERC20 token.

```
$ eth-wallet send-transaction -token FIT
  To address:  []: 0xAAD533eb7Fe7F2657960AC7703F87E10c73ae73b
  Value to send:  []: 0.9
  Password from keystore:

transaction: {'to': '0x19896cB57Bc5B4cb92dbC7D389DBa6290AF505Ce',
              'value': 0,
              'gas': 36536,
              'gasPrice': 20000000000,
              'nonce': 2,
              'chainId': 3,
              'data': '0xa9059cbb000000000000000000000000aad53
                       3eb7fe7f2657960ac7703f87e10c73ae73b0000
                       0000000000000000000000000000000000000000
                       000000c7d713b49da0000'}
Pending.....................
```

[2]FITCOIN is smart contract following ERC20 standard and deployed on Ropsten testnet network available at https://ropsten.etherscan.io/token/0x19896cb57bc5b4cb92dbc7d389dba6290af505ce

```
Transaction mined!
Hash of the transaction: 0x118556d192c2efb13ade6ccc2f18a631
                          e14256972af9f7ec8a67067aaafc978c
Transaction cost was: 0.00073072ETH
$
```

Listing 7.9: Send transaction of custom ERC20 token.

Listings 7.7 and 7.8 shows how the user can add new ERC20 token to the wallet and check if there is any balance on his address. To find if the user has some balance, wallet emits smart contract's function `balanceOf(address tokenOwner)` written in programming language Solidity[3]. This function checks the blockchain and returns an appropriate number of owned tokens. Reading from the blockchain is always free and therefore there is no cost for emitting this function. On the other hand sending transaction with ERC20 tokens will write data to the blockchain and it must cost something. However, sending ERC20 tokens are very similar to sending a transaction with ordinary ether. The transaction for sending FITCOIN ERC20 token is shown on Listing 7.9.

Difference between an ordinary transaction which sends ether and transaction which works with a smart contract is in building transaction itself. In case of the ordinary transaction the recipient address is another wallet's account address and the field `value` contains an amount of sending ether. When wallet wants to send transaction communicating with a smart contract, the recipient address of the transaction is smart contract's address of ERC20 token and the field `value` is 0. Everything that is going to happen with a smart contract is defined in field `data` of the transaction introduced in the transaction structure defined in Chapter 5.3.

Transaction's `data` field on Listing 7.9 contains one hexadecimal value which defines which smart contract's function is going to be called and which types are going to be inserted into this function's parameters. Within this example, the transaction is calling smart contract's function `transfer(address,uint256)` which is identified by first four bytes of the `data` field of transaction structure. The function name is converted to the hexadecimal value and put into Keccak256 hash function. The first four bytes for this function of every ERC20 token is $0xa9059cbb$. The rest of this hexadecimal value contains a smart contract's function types of parameters. First of them is `address`. This value is normalized address of the recipient who will obtain tokens from the originator of the transaction. Second is `uint256` which represent the number of tokens sent to the recipient. This amount must be converted into the smallest token's unit similar as ether is converted into wei. It is important because every token can have a different number of decimals. After all these conditions are met, wallet will create hexadecimal value from this amount and concatenate to the end of the hexadecimal string for `data` value.

Example of the final value of `data` field of a transaction is:

```
0xa9059cbb
000000000000000000000000aad533eb7fe7f2657960ac7703f87e10c73ae73b
0000000000000000000000000000000000000000000000000c7d713b49da0000
```

which basically means that smart contract will transfer 1 token to the address:

```
0xAAD533eb7Fe7F2657960AC7703F87E10c73ae73b
```

[3]Solidity - progarmming language for smart contracts.

Transaction cost, in this case, can depend on what action is this transaction's `data` triggering. To estimate gasLimit and fill transaction's `gas` value correctly, the wallet executes the given transaction locally without creating a new transaction on the blockchain. Then the fee is calculated in the same way as it was calculated during simple ether transaction with multiplying `gas` with `gasPrice` and converting to the ether. This ether is then deducted from transaction originator's ether balance.

7.1.3 Wallet's utilities

The following wallet's utilities are useful in case that user wants to check the wallet's configuration or make some configuration changes. Most of these listings don't need wider explanation.

```
$ eth-wallet network
You are connected to the Ropsten network!
$
```

Listing 7.10: Show connected network.

Listing 7.10 shows to which Ethereum blockchain is wallet connected. It can be main Ethereum network called mainnet where real funds are spent or one of the test networks (testnets) for example Ropsten. Some of these networks and their chainId is defined in Table 5.2.

Meanwhile, Listing 7.11 shows all configured and implicit tokens which can be currently manipulated by a user.

```
$ eth-wallet list-tokens
ETH
FIT
ZRX
BNB
$
```

Listing 7.11: Show all added custom tokens.

The last two listings 7.12 and 7.13 simply show how different help messages can be used.

```
$ eth-wallet --help
Usage: eth-wallet [OPTIONS] COMMAND [ARGS]...

Options:
  --help Show this message and exit.

Commands:
  add-token Add new ERC20 contract.
  get-balance Get address balance.
  get-wallet Get wallet account from encrypted keystore.
  list-tokens List all added tokens.
  network Get connected network (Mainnet, Ropsten) defined in...
  new-wallet Creates new wallet and store encrypted keystore file.
  restore-wallet Creates new wallet and store encrypted keystore file.
  reveal-seed Reveals private key from encrypted keystore.
```

```
  send-transaction Sends transaction.
$
```

Listing 7.12: Show wallet's commands.

```
$ eth-wallet send-transaction --help
Usage: eth-wallet send-transaction [OPTIONS]

  Sends transaction.

Options:
  -t, --to TEXT Ethereum address where to send amount.
  -v, --value TEXT Ether value to send.
  --token TEXT Token symbol.
  --help Show this message and exit.
$
```

Listing 7.13: Show command's options.

7.1.4 Configuration file

The configuration file was created for storing application settings and designed for any future extensions. This file is stored in YAML format and called `config.yaml`. By default, location of this file is in the user's home directory inside `.eth-wallet`. This path is default path for the encrypted keystore file too.

Example of such configuration file could be:

```
contracts:
  FIT: '0x19896cB57Bc5B4cb92dbC7D389DBa6290AF505Ce'
  ZRX: '0x70a68593BAfc497AC4F24Eaf13CF68E74135bA42'
eth_address: '0xB1f761734F00d1D368Ce6f82F755bBb3005538EB'
keystore_filename: /keystore
keystore_location: /Users/Marek/.eth-wallet
network: 3
public_key: '0xf94e03524a1bd803ee583a1f0de7eb1eb67a90d6802 \
               eeac22b90cfdd7ff491039441472e8db543467c0450 \
               d1b7c31b5e8f81616b99226775770f9dd531afd31a'
```

Listing 7.14: Example of configuration file conf.yaml.

On Listing 7.14 is shown that the configuration file can contain path and file name of the keystore file, the public key with derived Ethereum address or all currently supported ERC20 smart contracts added by user to the wallet. The file also contains connected Ethereum network which is in this example testnet Ropsten introduced in Table 5.2.

7.1.5 User interface

Graphical user interface (GUI) was tested on various operating systems including Linux, Windows and macOS. GUI is a minor extension to the command line interface which is the

major approach to the wallet's interaction. The main purpose of creating GUI was to show how easy is to create an extension to the pluggable design of wallet's API.

All graphical elements were created with *tkinter* which is a standard Python GUI package. However, some Linux distributions may require tkinter installation.

Chapter 8

Testing

Approaches to software development may differ in many ways and on many levels of abstraction. Finance technologies are working with money and user's funds and that is the reason why it must be properly tested. Maybe more than most of the other technologies. A mistake or a bug inside wallet software may result in loss of the funds or their stealing. Therefore, also the wallet created in Chapter 7 is covered with tests on code level. On the other hand the wallet is also tested in real environment where it created several transactions. This chapter is describing the testing of the wallet.

8.1 Unit tests

All the unit tests for this wallet were created using the PyTest framework. These tests are simply testing various wallet's functions. One of the examples is testing file api.py which is middleware between core functionalities and services like command line application and graphical user interface. The file api.py was designed for future extensions. For example, plugin which is implementing rest API can call api.py. That's why it is important to test api.py functions. However, it was said that api.py is calling some of the wallet's core functions. These functions are working with keys which must be generated properly and it is necessary to test the correctness of the keys after every future change of the wallet's code. In this way, tests will guarantee that the wallet will be able to work with the Ethereum ecosystem.

Code example of testing byte lengths of private and derived public keys with appropriate Ethereum address is shown on code Listing 8.1. Listing 8.2 shows a short example of command line output after tests execution.

```
def test_lengths(configuration):
    wallet = Wallet(configuration)
    wallet.create('eXtR4 EntroPy Str1Ng')

    # check length of private key
    assert len(wallet.get_private_key()) == 32

    # check length of public key
    public_key_bytes = decode_hex(wallet.get_public_key())
    assert len(public_key_bytes) == 64
```

```
    # check length of address
    address_bytes = decode_hex(wallet.get_address())
    assert len(address_bytes) == 20
```

Listing 8.1: Sample of unit test checking lengths of the keys.

```
========================== test session starts ==========================
platform darwin -- Python 3.6.5, pytest-4.4.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/Marek/Documents/Python/eth-wallet
plugins: mock-1.10.3collected 6 items

test_account.py [ 50%]
test_api.py [ 66%]
test_utils.py [100%]

================= 6 passed, 0 warnings in 4.01 seconds =================
Process finished with exit code 0
```

Listing 8.2: Short sample of running unit tests.

8.2 Integration tests and exceptions

Integration testing is based on PyTest framework similar as unit tests. However, it also combine usage of Exceptions during the code execution. Listing 8.3 is representing short sample of running integration tests. These integration tests cover mainly some of the complex features. In most of the cases, one Python test file contains tests for one command line feature and checks if final execution output is correct.

```
========================== test session starts ==========================
platform darwin -- Python 3.6.5, pytest-4.4.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/Marek/Documents/Python/eth-wallet
plugins: mock-1.10.3collected 18 items

test_add_token.py [ 5%]
test_cli.py [ 55%]
test_get_balance.py [ 66%]
test_get_wallet.py [ 72%]
test_transactions.py [100%]

================ 18 passed, 0 warnings in 24.26 seconds ================
Process finished with exit code 0
```

Listing 8.3: Short sample of running integration tests.

Exceptions are raised whenever a simple check of some basic inputs fails. All custom exceptions for the implemented wallet are defined in `exceptions.py` source code.

Example of some exceptions:
ERC20NotExistsException - Raised when custom token does not exist in wallet.
InfuraErrorException - Raised when wallet cannot connect to the Infura node.
InvalidValueException - Raised when some of expected values is not correct.
InvalidPasswordException - Raised when invalid password was entered.
InsufficientFundsException - Raised when a user wants to send funds and have an insufficient balance on the address.

8.3 Real environment testing

It is obvious that the development of applications based on blockchain technology like Ethereum cannot be tested in the real environment. Working with real transactions cost real funds and such development could get expensive very quickly. Therefore, there are more different networks for testing. Testing networks called *testnets* behave in the same or very similar way as the main network called *mainet*. The biggest difference is that testnets do not spend the real funds and any mined ether on these networks is worthless. The Ethereum testnets were mentioned more times. For example, in Table 5.2 are shown some of chain identifiers. That means that the same transaction can be sent to the various networks just with the changing one number as an identifier. Every application working with blockchain should be tested on one of the test networks before going live and make a transaction on mainnet. Of course, all testnets differ from each other in some features. This difference can be for example in used consensus algorithm. For wallet development described in Chapter 7, the Ropsten network was used.

Ropsten is a testing network which has almost the same behavior as Ethereum main network [27]. This was one of the reasons why it was chosen, however, the wallet should work in most of the Ethereum testnets. Any user with valid Ethereum address can generate some ether on testnet and use it for the testing purposes.

It is known that everything that happened on the Ethereum blockchain and is mined in one of the many blocks is permanently public. But this means a lot of data. Thankfully to block explorers, searching in blockchain and getting information is fast and simple. Block explorers are software which allows to visualize and explore blocks, addresses, transactions and smart contract's information or source code. Mainnet and testnet Ropsten has their own block explorers[1][2]. So now it is the only question of search hash or hexadecimal number within search-bar. Some of the examples how to work with block explorers was shown in 7.1.2 where wallet sent a transaction, therefore it wrote information to the blockchain permanently and anyone can find it with block explorer.

[1] Block explorer for Ethereum mainnet is available at https://etherscan.io/.
[2] Block explorer for Ropsten testnet is available at https://ropsten.etherscan.io/.

Chapter 9

Conclusion

Ethereum is one of the most popular cryptocurrencies which has great potential for changing the way how humans work with computers and the internet nowadays.

The main goal of this paper was to describe technical principles necessary for the implementation of wallet software compatible with Ethereum network.

The explanation of blockchain technology and the analysis of nowadays popular wallets were provided in chapters 2 and 3. In chapters 4 and 5 are all technical details for creation of wallet software. Based on this knowledge, a fully working virtual wallet was designed, built and tested. All these details are described in chapters 6 and 8. This wallet is following the community standards and the best practices. A user of the wallet can manage keys, addresses and send transactions with Ethereum build-in tokens or Ethereum custom ERC20 tokens. Therefore, I consider all points of specification fulfilled and thus the paper usable as a source of solid technical information about blockchain technology.

This paper can also be used as a guideline for wallet software creation compatible not only with the Ethereum network but also with other cryptocurrencies. Especially since all the principles are shared across most of the blockchain technologies. This is also one of the areas where wallet could be improved in the future. At the moment, the wallet is a single account compatible with Ethereum tokens. However, within this paper, I also mention some of the standards which describe multi-account and multicurrency architecture which can be implemented as a part of e-commerce solutions. I believe, that some other student can follow this paper and create such improved wallet.

Since the created wallet is fully working implementation and people can use it, there is another great potential for a business opportunity. The usage of this solution was already discussed with an organisation from the finance industry.

On the other hand, the wallet was designed for future extensions as it provides API to its core functionalities. I can imagine that some e-commerce solution would connect to the wallet via external service as rest API. Because of the easy pluggable design of the wallet, it was also easy to create more user interfaces. Command line interface was intended as the main approach, but also graphical user interface was created which could be considered as an extra job. Another extra task is that this wallet does not work only with the Ethereum build-in tokens. The wallet can also work also with smart contract ERC20 tokens deployed on the Ethereum network. Also, custom FITCOIN token was issued for testing purpose.

Bibliography

[1] *BIP-32*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[2] *BIP-39*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[3] *BIP-43*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki

[4] *BIP-44*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

[5] *Block Explorer and Analytics Platform for Ethereum, a decentralized smart contracts platform*. [Online; visited 15.05.2019].
Retrieved from: https://etherscan.io/tokens

[6] *EIP-155*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md

[7] *EIP-55*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/Ethereum/EIPs/blob/master/EIPS/eip-55.md

[8] *ERC20 token proposal*. [Online; visited 15.05.2019].
Retrieved from: https://github.com/ethereum/eips/issues/20

[9] *ERC20 token standard*. [Online; visited 15.05.2019].
Retrieved from: https://theethereum.wiki/w/index.php/ERC20_Token_Standard

[10] *Ethereum introduction*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/ethereum/wiki/wiki/Ethereum-introduction

[11] *Ethereum white paper*. [Online; visited 15.05.2019].
Retrieved from: https://github.com/ethereum/wiki/wiki/White-Paper

[12] *Keccak-256*. [Online; visited 15.05.2019].
Retrieved from: https://keccak.team/keccak_specs_summary.html

[13] *libsecp256k1*. [Online; visited 15.05.2019].
Retrieved from: https://github.com/bitcoin-core/secp256k1

[14] *Official webpage of Ethereum foundation*. [Online; visited 15.05.2019].
Retrieved from: https://www.ethereum.org/

[15] *Official webpage of the Coinbase wallet and exchange*. [Online; visited 15.05.2019].
Retrieved from: https://www.coinbase.com/

[16] *Official webpage of the Jaxx wallet*. [Online; visited 15.05.2019].
Retrieved from: https://jaxx.io/

[17] *Official webpage of the MetaMask wallet*. [Online; visited 15.05.2019].
Retrieved from: https://metamask.io/

[18] *Official webpage of the Mist wallet*. [Online; visited 15.05.2019].
Retrieved from: https://github.com/ethereum/mist

[19] *Official webpage of the MyEtherWallet wallet*. [Online; visited 15.05.2019].
Retrieved from: https://www.myetherwallet.com/

[20] *OpenSSL*. [Online; visited 15.05.2019].
Retrieved from: https://www.openssl.org/

[21] *RCFs*. [Online; visited 15.05.2019].
Retrieved from: https://www.ietf.org/standards/rfcs/

[22] *RLP*. [Online; visited 15.05.2019].
Retrieved from: https://github.com/ethereum/wiki/wiki/RLP

[23] *SLIP-0044*. [Online; visited 15.05.2019].
Retrieved from:
https://github.com/satoshilabs/slips/blob/master/slip-0044.md

[24] Andreas M. Antonopoulus: *Mastering Bitcoin*. O'Reilly Media. 2017. ISBN 978-1449374044.

[25] Andreas M. Antonopoulus, Dr. Gavin Wood: *Mastering Ethereum*. O'Reilly Media. 2018. ISBN 978-1491971949.

[26] Baran, P.: *On Distributed Communications Networks*. [Online; visited 15.05.2019].
Retrieved from:
https://www.rand.org/content/dam/rand/pubs/papers/2005/P2626.pdf

[27] Boily, F.: *Explaining Ethereum Test Networks And All Their Differences*. [Online; visited 15.05.2019].
Retrieved from:
https://medium.com/coinmonks/ethereum-test-networks-69a5463789be

[28] Brown, D. R. L.: *SEC 1: Elliptic Curve Cryptography*. Standards for Efficient Cryptography, [Online; visited 15.05.2019].
Retrieved from: http://www.secg.org/sec2-v2.pdf

[29] Darrel Hankerson, Alfred Menezes, Scott Vanstone: *Guide to Elliptic Curve Cryptography. Springer Professional Computing.* Springer. 2004. ISBN 0-387-95273-X.

[30] Frankenfield, J.: *Double-Spending*. June 2018. [Online; visited 15.05.2019]. Retrieved from: https://www.investopedia.com/terms/d/doublespending.asp

[31] Nakamoto, S.: *Bitcoin white paper*. January 2009. [Online; visited 15.05.2019]. Retrieved from: https://bitcoin.org/bitcoin.pdf

[32] Schollmeier, R.: *A Definition of Peer-to-Peer Networking for the Classification of Peer-toPeer Architectures and Applications* . Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002), [Online; visited 15.05.2019]. Retrieved from: https://www.computer.org/csdl/proceedings/p2p/2001/1503/00/15030101.pdf

[33] Wood, D. G.: *Official ethereum yellow paper*. [Online; visited 15.05.2019]. Retrieved from: https://ethereum.github.io/yellowpaper/paper.pdf

Appendix A

Content of attached storage medium

The attached storage medium contains:

MANUAL.md Markdown Documentation File, which contains brief manual how to work with attached storage or how to try source code locally.

eth-wallet Source code directory. The *eth-wallet* directory contains mainly Python files but also any kind of configuration for the project.

Bachelor's Thesis - Text Contains PDF file of Bachelor's Thesis with all theory and explained implementation.

Bachelor's Thesis - Latex Latex source code. After compiling this sources, PDF with Bachelor's Thesis will be created.

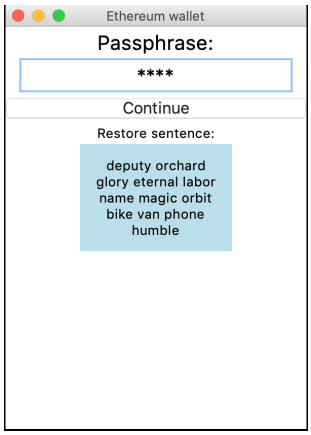
Appendix B

Graphical user interface

All following figures B.1, B.2 and B.3 are examples of graphical user interface (GUI) on various operating systems including Linux, Windows and macOS. GUI is an extension to the command line interface which is the main approach to the wallet's interaction.

The main purpose of creating GUI was to show how easy is to create an extension to the pluggable design of wallet's API.

All graphical elements were created with *tkinter* which is a standard Python GUI package. However, some Linux distributions may require installation.



(a) Creation of a new wallet on macOS.

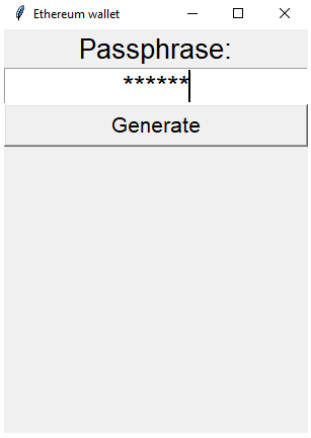
(b) Wallet's home page on macOS.

Figure B.1: Example of GUI on macOS platform.

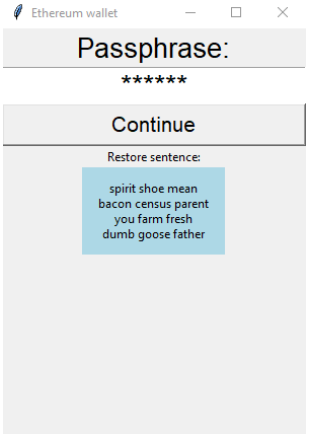
(a) Wallet's home page on Fedora.

(b) Wallet's transaction page on Fedora.

Figure B.2: Example of GUI on Fedora platform.



(a) Creation of a new wallet on Windows.



(b) Restore sentence on Windows.

Figure B.3: Example of GUI on Windows platform.