FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	A miner for FITCOIN
Student:	Filip Volf
Supervisor:	Mgr. Jan Starý, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2019/20

Instructions

1. Implement the process of mining (i.e. creating new blocks) for FITCOIN, which is simultaneously implemented in other theses (see references).
2. Write the implementation in C.
3. Be portable at least accross OpenBSD, FreeBSD, NetBSD, MacOSX, Linux.
4. For hash and crypto functions, use the LibreSSL library.
5. Test your implementation. Document your testing.
6. Document your implementation.

References

Satoshi Nakamoto: Bitcoin - a Peer to Peer Electronic Cash System, 2009
Andreas Antonopouos: Mastering Bitcoin, O'Reilly 2017
Andrea Zábojníková: Správa blockchainu pro FITCOIN, ČVUT 2019
Jiří Havruševič: Peněženka pro FITcoin, ČVUT 2019

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

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

Prague January 15, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

A miner for FITCOIN

Filip Volf

Department of Theoretical Computer Science
Supervisor: Supervisor: Mgr. Jan Starý, Ph.D.

May 15, 2019

Acknowledgements

I would like to thank my supervisor Mgr. Jan Starý, Ph.D. for his guidance and helpful approach.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 15, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Volf, Filip. *A miner for FITCOIN*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Bitcoin je digitální měna, která se snaží řešit problémy stávajícího finančního systému. Ačkoliv původní návrh Bitcoinu byl jednoduchý, složitost této kryptoměny neustále roste.

Tato práce se zabývá základními myšlenkami kryptoměny a implementuje software nazývaný miner, který umožňuje decentralizaci. Práce je součástí projektu Fitcoin, jehož cílem je vyvinout kryptoměnu oproštěnou od nadbytečných funkcí, která je přenositelná mezi unixovými operačními systémy a různým hardwarem.

Software jsem implementoval v programovacím jazyce C s použitím LibreSSL knihovny kryptografických funkcí. Kryptoměnu Fitcoin se začleněným minerem jsem otestoval na různých operačních systémech a různém hardwaru, také jsem prověřil funkčnost vybraných částí pomocí unit testů.

Klíčová slova kryptoměna, bitcoin, těžař, blockchain, decentralizace, consensus, dvojité utrácení, proof of work, hash, nonce

Abstract

Bitcoin is a digital currency, that tries to solve the failures of the current financial system. Although the original proposal for Bitcoin was simple, this cryptocurrency increasingly grows on complexity.

This thesis deals with fundamental ideas of a cryptocurrency and implements a software called miner, which facilitates the decentralization. The thesis is part of the project Fitcoin, which aims to develop a barebones cryptocurrency stripped out of all unnecessary features, which would be portable across different Unix operating systems and a variety of hardware.

I implemented the software in the C programming language with the help of LibreSSL library of cryptographic functions. I tested the cryptocurrency Fitcoin with the incorporated miner on different operating systems and a variety of hardware. I also verified the correct function of selected parts of the miner with the help of unit tests.

Keywords cryptocurrency, bitcoin, miner, blockchain, decentralization, consensus, double spending, proof of work, hash, nonce

Contents

List of Figures

Chapter 1

Introduction

The progress of humankind moves at an incredible speed. Self-driving cars appear, machines are capable of face recognition, but it is extremely difficult to transfer money on weekends. It is only a matter of time until the current financial system will be replaced by something superior. In the aftermath of the 2008 financial crisis a new digital currency, Bitcoin, came into existence.

Bitcoin is a *cryptocurrency*, i.e., decentralized digital currency secured by cryptography with no central issuing or regulating authority. It tries to resolve the defects of the current financial system by taking the power from banks. Blockchain is the underlying technology and is widely considered to be the most promising innovation of this generation. The market capitalization of Bitcoin skyrocketed to 334 billion dollars on December 17, 2017, which indicates that it is not just for hobbyists.

1.1 Fitcoin

Fitcoin is a barebones portable cryptocurrency supervised by Mgr. Jan Starý, Ph.D. and developed by students from Czech Technical University (CTU) within the scope of theses:

- Správa blockchainu pro FITCOIN (Andrea Zábojníková)
- Peněženka pro FITcoin (Jiří Havrusevič)
- A miner for FITCOIN (Filip Volf)

Similar cryptocurrency with the same name was developed in the past at CTU in Prague within the scope of theses *FITCOIN: transakce* by Jan Tománek and *blockchain pro FITCOIN* by Mikuláš Dvořák. Project Fitcoin described in this thesis does not share anything with the previous one other than the name.

1.2 Goals

The first goal is to analyze the inner workings of a cryptocurrency with the focus mainly on the concepts relating to the miner. The second one is to implement the miner for the newly emerging cryptocurrency Fitcoin.

The goal is not to develop the cryptocurrency Fitcoin, nor describe its implementation, however, it is described briefly due to the close connection to the miner.

1.3 Structure

I start with introducing the fundamental cryptography used in cryptocurrencies, such as secure hash functions, public-key cryptography, digital signatures, Merkle trees, Byzantine generals problem and proof of work.

In the Cryptocurrency chapter, I focus on how is the cryptography utilized to implement a decentralized digital currency. I first explain the transactions, then I talk about decentralization and the problems it brings. Finally, I describe the role of the miner in solving these problems.

In the next chapter, I briefly introduce the Fitcoin, focusing mainly on its ecosystem and blockchain structure. In the following chapter, I demonstrate the implementation of the miner in pseudocodes. The last chapter is devoted to testing.

1.4 Background

In 1998 a computer engineer Wei Dai proposed a distributed electronic cash system B-money, which is impossible to regulate [1]. The proposal describes the core concepts later implemented in Bitcoin, however, Dai questions the influence: "*...the creator of Bitcoin, who goes by the name Satoshi Nakamoto, did not even read my article before reinventing the idea himself. He learned about it afterward and credited me in his paper.*" [2].

Decentralized money got more attention in October 2008, when an unknown person or group of people using name Satoshi Nakamoto[1] published a paper *Bitcoin: A Peer-to-Peer Electronic Cash System* [3] describing a decentralized digital currency, that solves the double spending problem.

Satoshi Nakamoto states: "*What is needed is an electronic payment system based on cryptographic proof instead of trust*" [3]. On 3 January 2009, the bitcoin network was created when Nakamoto mined the first block of the chain, known as the genesis block with embedded following text: "*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.*" [**?**]. The note not only points out to the instability of fractional-reserve banking but also serves as a timestamp.

In 2014, Andreas M. Antonopoulos authored the book *Mastering Bitcoin* [4], considered to be the best technical guide written about the technology. Blockchain technology is the underlying technology of Bitcoin and together with the Internet, is being described as the advent for the era of decentralization. The functionality of decentralized ledgers can be used not only for currencies but to register and confirm any kind of data.

Blockchain is a combination of well-established concepts in an interesting way. It solves the Byzantine generals problem faced by any distributed system [5] in a narrow context of digital currency, by utilizing proof of work algorithm, which was proposed in 1992 to combat junk mail.

From the cryptography perspective, it utilizes public-key cryptography and hash functions described by Diffie and Hellman far back in 1976, digital signatures described in 1979 by Michael Rabin and Merkle trees described by Ralph Merkle in 1979.

[1] Some speculate Satoshi Nakamoto could be Wei Dai himself.

Chapter 2

Cryptography

Cryptocurrencies rely on well established cryptographic algorithms such as public key cryptography and digital signatures. In this chapter, I will introduce cryptography concepts needed to build a decentralized digital currency.

The reason for the name cryptocurrency is due to the fact that it heavily relies on cryptography to function. Although crypto means "concealed" or "secret", the contrary is true about cryptocurrencies, the main idea behind it is that everyone knows everything.

2.1 Hash Functions

In this section, I will thoroughly explain the hash function, since it is the fundamental primitive in modern cryptography and an essential part of the cryptocurrency technology.

In [6] a *hash function* is defined as a function, that takes as input an arbitrarily long document D and returns a short bit string h. The primary properties that a hash function H should possess are as follows:

- Computation of $H(D)$ should be fast and easy, e.g., linear time
- Inversion of H should be difficult, e.g., exponential time. More precisely, given a hash value h, it should be difficult to find document D such that $H(D) = h$.

A hash procedure must be deterministic, i.e., for a given input it must always generate the same output. A good hash function should also map the inputs evenly over the output range, meaning that every output value should be generated with the same probability. [7]

2.1.1 Digest

The hash function maps data of arbitrary size to a fixed size bit string called *digest*. You can think of a *digest*, as a fingerprint. A person produces the same

fingerprint every time it is taken, but it is hard to find another individual with the same fingerprint. The fingerprint does not unveil any other information, for example you cannot know what eye color does the person have. [8]

Hash functions can be used to detect changes in data. Suppose Alice sends a message to Bob, but it may be altered during the transfer. Alice calculates the cryptographic hash of the message and sends it to Bob. Bob then calculates the hash of the message he received. If the Bob's hash is the same as the one received from Alice, Bob knows for sure that the message has not been altered or corrupted. [7]

The computed digest is sensitive to all input bits. Even if we make a minor modification to the input, the fingerprint should look very different as shows figure 2.1. [8]

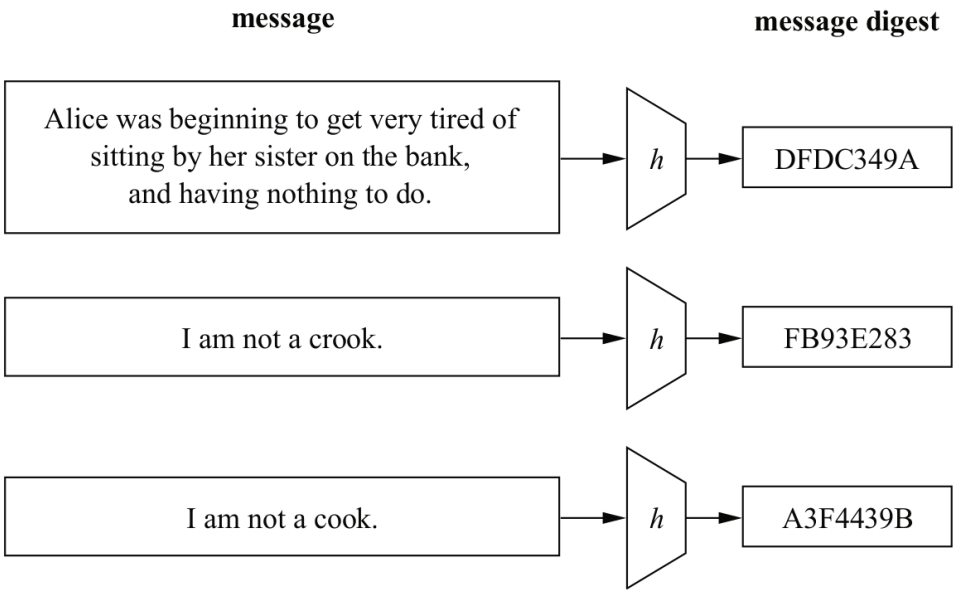

Figure 2.1: Digest[8]

2.1.2 Collision Resistancy

This section is based on [9]. For the application in cryptocurrencies, it is important that the hash function is cryptographically secure. Hash function H is *cryptographically secure*, if it is *preimage resistant*, *second preimage resistant* and *collision resistant*.

- Preimage resistant: A hash function H is preimage resistant, if it is hard to find any m for a given h with $h = H(m)$.
- Second preimage resistance: A hash function H is second preimage resistant if it is hard to find for a given m_1 any m_2 with $H(m_1) = H(m_2)$.

- Collision resistant: A hash function H is collision resistant if it is hard to find a pair of *m1* and *m2* with $H(m_1) = H(m_2)$.

For a good cryptographically secure hash function there should be no way to solve preimage resistance and second preimage resistance faster than a brute force attack.

To break the preimage resistance, the attacker chooses m randomly until $h = H(m)$. If s is the length of the result in bits, then $H(m)$ can have 2^s different results and each result P_h has the same probability $1/2^s$. The attacker has to choose on average $2^s/2$ different inputs m, until he finds $h = H(m)$. The same idea applies to second preimage resistance, so the complexity of an attack against first and second preimage resistance is $1/2 * 2^s = \mathcal{O}(2^s)$.

Unfortunately it takes on average only $\mathcal{O}(\sqrt{2^s})$ hash operations to find a pair m_1, m_2 such that $H(m_1) = H(m_2)$. Breaking the collision resistance is easier than breaking preimage resistance due to the *birthday paradox*. It deals with the probability that, in a set of n randomly chosen people, some will have the same birthday.

The probability is 100% when the number of people reaches number of days in a year. However, 99.9% probability is reached with just 70 people, and 50% probability with 23 people, assuming each day of the year is equally probable for a birthday.

To demonstrate the importance of collision resistancy, assume that Alice is able to find D_1 and D_2 such that $H(D_1) = H(D_2)$. Suppose D_1 says "Pay Alice 1 Bitcoin" and D_2 says "Pay Alice 1000 Bitcoins". If someone signs D_1, he also signs D_2, thus Alice can now present the signature as being on D_2 and getting paid 1000 Bitcoins instead of 1 Bitcoin. [6]

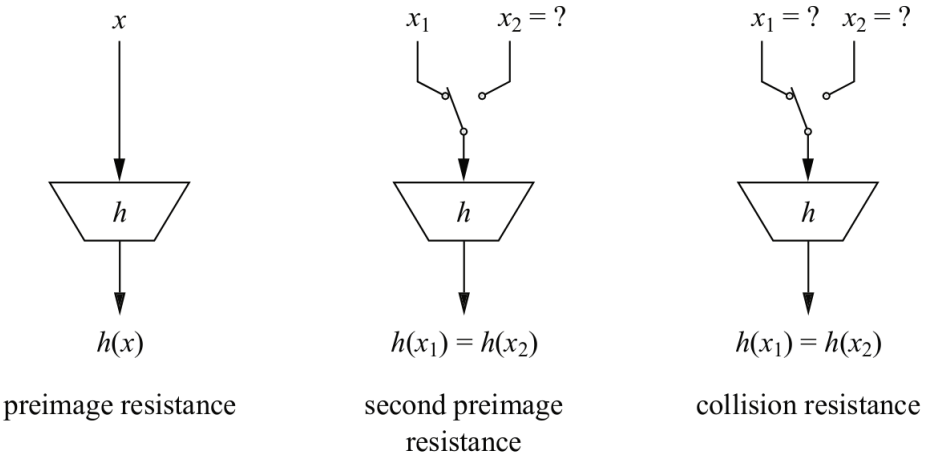

Figure 2.2: Hash function security properties[8]

2.1.3 SHA

The hash function needs to be executed fast, for example when digitally signing a document, it can can be many megabytes in length. Due to the speed requirement, hash functions are constructed using mixing operations rather than basing them on classical hard mathematical problems. The most used hash function nowadays is called SHA (Secure Hash Algorithm). There are several versions of SHA that achieve different level of security, such as SHA-224, SHA-256 or SHA-512. SHA-n signifies that the output is n bits in length, it takes approximately $2^{n/2}$ to find a collistion. The exception is the original SHA-1 whose output length is 160 bits. [6]

Hash functions are used extensively in cryptocurrencies for creation of addresses and in the mining algorithm. Since Bitcoin and Fitcoin utilizes the SHA-256, I briefly demonstrate how the SHA family works by illustrating the structure of SHA-1 in Alogrithm 1 from *An Introduction to Mathematical Cryptography* [6]. [4]

Algorithm 1 SHA-1 Algorithm

```
Break document D (with extra appended bits) into 512-bit chunks
Start with five specific initial values h_0,...,h_4
for all 512-bit chunks do
    Break a 512-bit chunk into sixteen 32-bit words
    Create eighty 32-bit words w0, ..., w_79 by rotating the initial words
    for i ← 0 to 79 do
        Set a = h_0, b = h_1, c = h_2, d = h_3, e = h_4
        Compute f using XOR and AND operations on a, b, c, d, e
        Mix a, b, c, d, e by rotating some of their bits, permuting them
        Add f and w_i to a
    end for
    h_0 ← h_0 + a, ..., h_4 ← h_4 + e
end for
return h_0||h_1||h_2||h_3||h_4
```

2.2 Hash Pointer

In cryptocurrencies hashes are utilized within several datastructures in the form of *hash pointers*. "*A hash pointer is a pointer to where data is stored together with a cryptographic hash of the value of that data at some fixed point in time.*" [10]

A regular pointer provides a way to get an information, a hash pointer enables us to verify that the information hasn't been changed. By implementig familiar structures with hash pointers, they acquire interesting properties like for example immutability (it cannot be changed). [10]

2.3 Merkle Tree

"*A merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data.*" [4] It is named after its inventor Ralph Merkle.

Suppose there are some blocks of arbitrary data. A merkle tree can be built with the hash pointers. The blocks are put into the leaves of the tree. They are then grouped into pairs of two, and a data structure with two hash pointers pointing to each of the blocks is built for each pair. These structures form the next level of the tree. They are again grouped into pairs and a new structure containing hash of each is created. By continuing we reach a single block called the merkle root. Figure 2.3 illustrates this whole process. [10]

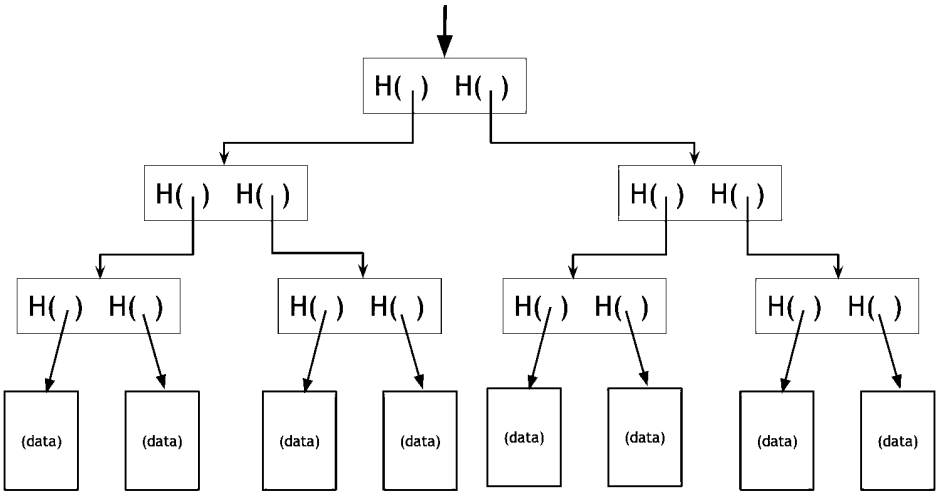

Figure 2.3: Merkle Tree[4]

Merkle trees are used in cryptocurrencies to summarize all the transactions in a block, creating a fingerprint of the entire set of transactions. This facilitates an easy way to verify whether a transaction is included in a block. [4]

It doesn't matter how many transactions are in a block, the merkle root always summarizes them into a single digest. It takes only $2 * log_2(N)$ calculations to check if a transaction is in a tree made from N transactions. [4]

Merkle tree reduces the amount of data that has to be maintained for verification purposes and helps verify that data is is presented and recorded in a chronological order. The ability to prove that a log is complete and consistent is essential to blockchain technology. [10]

To prove that certain a element is a member of the merkle tree just the hashes on the path from the element to the root are needed. The rest of the tree can be ignored, as the provided hashes on the path are enough to verify the hashes all the way up to the root as is visualized in figure 2.4. [10]

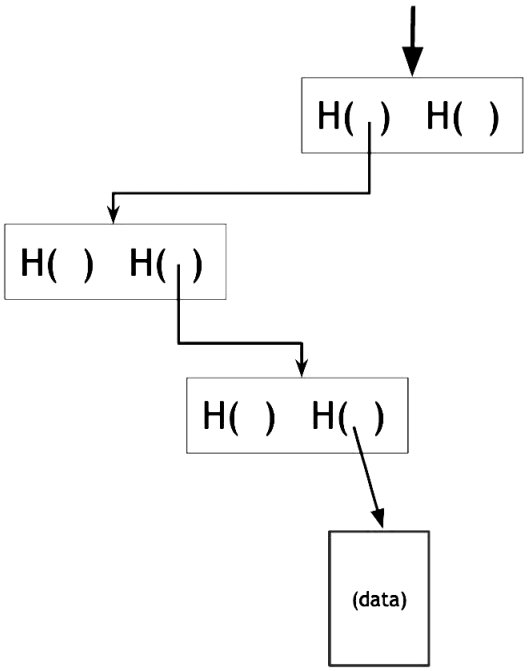

Figure 2.4: Proof of Membership [10]

Although Fitcoin does not utilize merkle trees for this purpos, in Bitcoin a method called Simplified Payment Verification (SPV) implemented. SPV allows a lightweight client to verify that a transaction is included in the Bitcoin blockchain, without downloading the entire blockchain.

2.4 Public Key Cryptography

Symmetric key algorithms use the same secret key and both of the parties have to keep it secret. The key in such a system has to be exchanged in a secure way and that is a nontrivial requirement. On top of that, if messages are meant to be secure from other users, a separate key is needed for each possible pair of users. [8]

Public key cryptography is based on a revolutionary idea, that the key to decrypt the message has to be secret, but the key for encrypting does not. [8]

Public-key cryptography is a system that uses pairs of keys: public keys which may be shared, and private keys which should be kept secret. Only the private key is required to be unpublished. Sharing the public key does not compromise the security. [7]

The generation of keys depends on mathematical problems used to construct one-way functions. They are practically irreversible, which means that

it's easy to calculate the output, but it is infeasible to get the input from the output. [4]

Let's imagine that Bob shares a public encryption key and keeps the matching private key used for decryption secret. The key K consists of two parts, a public part, K_{pub} , and a private one, K_{pr}. Anyone is able to encrypt data using K_{pub}, but that encrypted data can only be decrypted with K_{pr}. [8].

In cryptocurrencies the public key can be shared to receive funds, while the private key is kept secret and is needed to spend the funds [4].

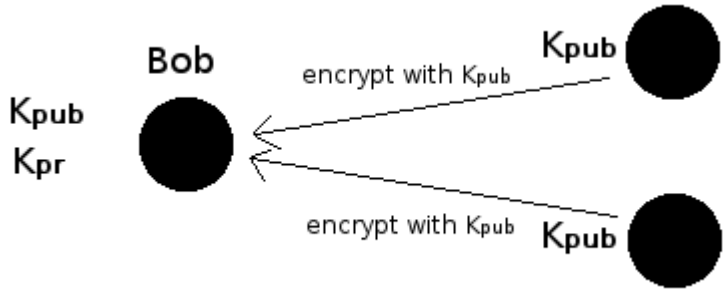

Figure 2.5: Public Key Cryptography[8]

2.5 Digital Signatures

To explain the importance of digital signatures, an example with two communicating parties Alice and Bob from the book *Understanding cryptography* is presented. [8]. It is often the case that Alice and Bob want to communicate securely with each other, but at the same time they might be interested in cheating each other.

Suppose that Alice owns a dealership for cars where you can order cars online. Bob chooses a pink car and sends the encrypted order to Alice, she decrypts it and is happy to have sold another model.

Upon delivery Bob changes his mind, but Alice has a "no return" policy. Since Bob now claims that he never ordered the car, she sues him. Alice's lawyer presents Bob's digital car order together with the encrypted version of it, but the judge has no way of knowing whether it was generated by Bob or Alice.

This can be solved, by Bob signing the message x with a signature algorithm, which is a function of K_{pr} and the message x itself. The message is then sent together with the the signature s to Alice and the signature serves as a proof, that the message was generated by Bob.

A digital signature without the message is like a handwritten signature on a piece of paper without the contract. Assuming the signer kept his private key secret, noone else is able to generate a signature on his behalf.

Public key cryptosystem is like a digital version of a bank deposit vault. Anyone can put an envelope through a narrow slot to a bank deposit, however only the one with the key to the vault's lock can read the the messages. By contrast, digital signature can be viewed as a digital signet ring. The owner of the ring puts some wax onto a document and presses the ring to make an impression. Anyone can verify that the wax impression was made by the owner, since he is the only one who has the ring. [6]

"*A digital signature is a data string which associates a message (in digital form) with some originating entity.*" It is a number dependent on a secret known only to the signer and on the message being signed. In case of a dispute, an unbiased third party can resolve the matter, without having the access to the signer's private key. [7]

To create a digital signature, the document is hashed and the digest is encrypted by the private key. The signature is then sent together with the document. Any change in the document produces a different digest, which enables others to validate the integrity. The authenticity is also ensured, because only the owner of the private key can encrypt the digest. This encrypted digest can be reverted back to the original only with the use of the corresponding public key through the verification algorithm.[6]

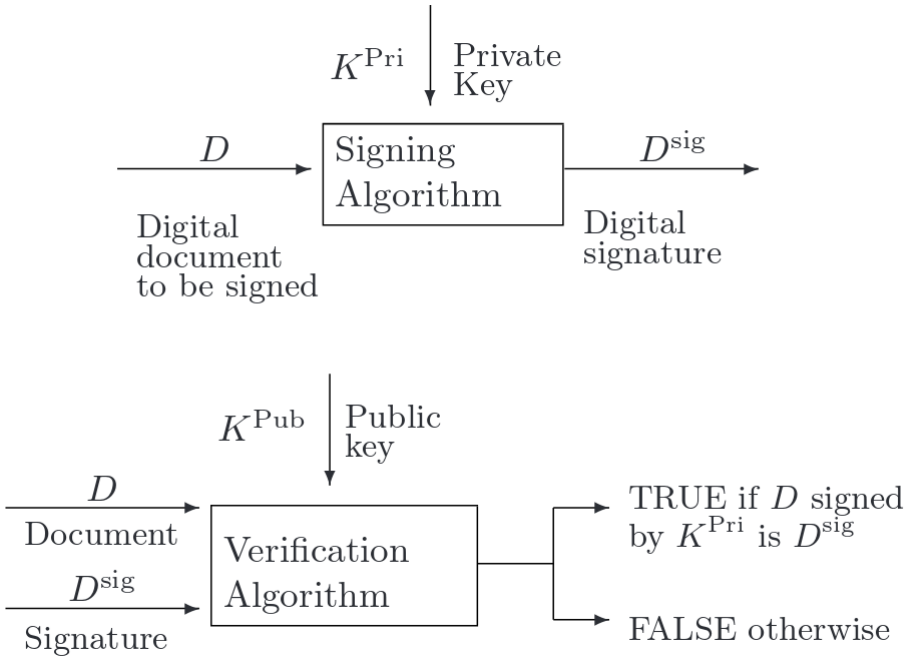

Figure 2.6: Digital Signature[6]

The first method discovered was the RSA signature scheme remaining one of the most practical approach available. Following research has resulted in

alternative techniques. Bitcoin utilizes elliptic curves for digital signatures, but public-key cryptography family of integer factorization and discrete logarithms can be also used. [4]

2.6 Byzantine Generals Problem

This whole section is based on the publication *The Byzantine generals problem* [5]. The Byzantine generals problem is faced by any distributed computer system network and can be expressed with the following question: "*How do you make sure that multiple entities, which are separated by distance, are in absolute full agreement before an action is taken?*" [11]

To demonstrate the problem, imagine several divisions of an army, commanded by their generals, attacking a city. Generals are too far from each other, therefore they have to communicate via a messenger. The goal is to agree on a battle plan, however, some of the generals might be traitors, trying to prevent an agreement.

Byzantine generals problem can be depicted as follows: A commanding general must send an order to his n-1 lieutenant generals such that:

- All loyal lieutenants obey the same order.
- If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

If the generals can send only oral messages, no solution will work unless more than two-thirds of the generals are loyal. There is no solution with only three generals in presence of just a single traitor.

An oral message is wholly under the control of the sender, so a traitor can transmit any possible message, which corresponds to how computers communicate.

In both scenarios shown in figure 2.7 and figure 2.8, Lieutenant 1 does not know who the traitor is. He cannot tell, what message did the commander send to Lieutenant 2. Both of these scenarios look exactly the same to Lieutenant 1, so he must obey the "attack" instruction in both of them, to satisfy the second condition in the Byzantine Generals' Problem.

The solution to the problem relies on an algorithm that can guarantee that loyal generals decide upon the same plan. The algorithm has to guarantee the first condition no matter the traitors' behaviour, however the traitors may do anything they wish.

In a decentralized cryptocurrency network, all participants are exactly of equal hierarchy. Nodes in a distributed ledger must agree on certain rules and agree on transaction assesment before it is added to the database.

If a corrupted message is transmitted, the network as a whole should resist such an attack. The network in its entirety has to agree upon all messages in the network.

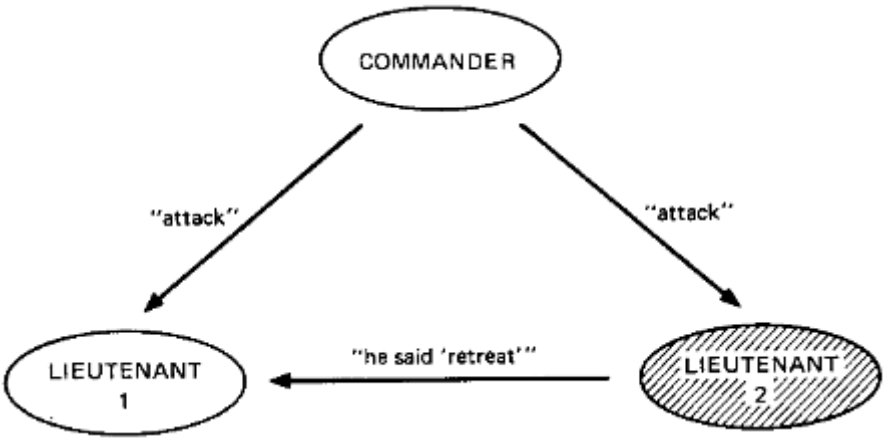

Figure 2.7: Lieutenant is a traitor[5]

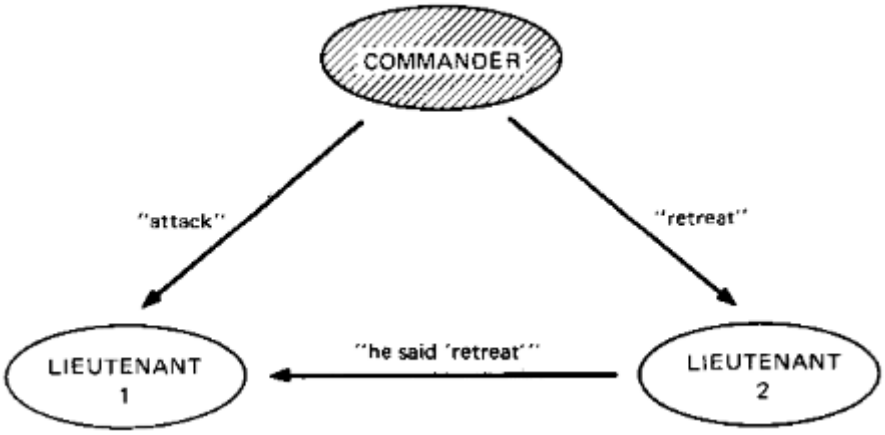

Figure 2.8: Commander is a traitor[5]

2.7 Proof of Work

Proof of Work was originally designed to prevent spam attacks. "*The main idea is for the mail system to require the sender to compute some moderately expensive, but not intractable, function of the message and some additional information. Such a function is called a pricing function.*" [12]

According to [13] a *proof of work* system is: "*a protocol in which a prover demonstrates to a verifer that she has expended a certain level of computational effort in a specified interval of time. Although not defined as such or treated formally, proofs of work have been proposed as a mechanism for a number of security goals, including server access metering, construction of digital time capsules, and protection against spamming and other denial-of-service*

attacks."

To calculate the solution, extra work has to be done, which can be easily verified when presented to others. It has to be hard to find a solution but easy to verify that an answer solves the problem. [12]

Proof of Work system disincentivizes malicious behavior, due to the fact that if a moderate work is required for an action, then the attacker wouldn't have enough computational power or the financial costs of the processing power would likely exceed the profits. [12]

In cryptocurrencies if enough nodes are searching for a specific solution, then the processing power required to manipulate a network becomes unattainable for any single bad actor.

Chapter 3

Cryptocurrency

Cryptocurrency is a digital asset used as a medium of exchange secured by cryptography to prevent fraudulent behaviour.

Cryptography means "secret writing" in Greek, but ironically encryption is not an important part of Cryptocurrency as the transaction data does not need to be secret to protect the funds [4].

At the core of cryptocurrency is the distributed ledger which contains a record of all transactions in a system. *Distributed* means that the data is stored across a network of nodes. The key is to ensure the nodes on the network agree with the order of the transactions in the network, preventing double spending or other invalid data. [4]

3.1 Transaction

The ownership of coins is established with the help of digital keys, digital signatures and addresses. Transactions require a digital signature to be accepted by others. Only the owner of the secret key can generate such signatures and spend the funds.

A *transaction* T moving n coins between user a and b is a tuple: $T = (Pub_a, Pub_b, n, Proof)$, where Pub_a is the public key of user a, Pub_b of user b, n is the number of coins and $Proof$ indicates the proof of ownership. [14]

The proof is a digital signature generated by a corresponding private key to Pub_a. It provides authenticity (information is from the source it claims to be), integrity (data cannot be modified) and non-repudiability (author cannot dispute it's authorship). [14]

3.1.1 Transfer

To demonstrate how transaction works, a simplified example from book the *Bitcoin and Cryptocurrency Technologies* [10] is presented. Imagine three entities, Alice, Bob, Charlie and an electronic coin with two rules.

First rule is that a chosen entity, Alice, can generate new coins that belong to her. To create coins, Alice constructs a string "Generate [ID]" where "[ID]" is a unique coin id that has never been generated before and computes a digital signature of this string using her private key. The string with Alice's signature is a coin. Anyone can check that the signature of the statement is valid, thus is a valid coin.

The second rule is anyone owning coins can transfer them to someone else. Let's say Alice wants to pay to Bob. She creates a statement "Pay [ID] to Bob" where "[ID]" is hash pointer referencing the coin.

Identities are just public keys, so "Bob" refers to Bob's public key. The statement has to be signed by Alice, because she owns the coin. Bob can then prove to anyone that he owns the coin by showing the statement signed by Alice.

Bob can now send the coin to Charlie by signing a statement "Pay [ID2] to Charlie" where "[ID2]" is the hash referencing to previously received coin.

Anyone can verify that Bob is the owner by following the chain of hash pointers and verifying at each step, that the owner signed a correct statement. An example of series of transactions is demonstrated in figure 3.1. For simplicity, the hash in the table is just a short made up 4 digit number signifying the hash by which is the coin referenced.

hash	statement	signature
6742	Generate id00001	Alice's signature
1342	Pay 6742 to Bob	Alice's signature
3441	Pay 1342 to Charlie	Bob's signature

Figure 3.1: Series of transactions in the simplified electronic coin example

The problem with such system is, that the coins cannot be subdivided. To solve this problem, let there be two kinds of transactions. "Generate coins" and "Transfer coins".

"Generate coins" is valid only if it's signed by Alice and it creates multiple coins, each having a value, recipient (public key), and a serial number within the transaction. The "Transfer coins" consumes coins and destroyes them and then creates new of the same total value, which might belong to someone else.

The transaction is valid if all following conditions are met:

- Destroyed coins were previously generated
- Destroyed coins were not already spent
- Total value of destroyed coins equals total value of new coins
- The transaction is digitally signed by the owners of all of destroyed coins.

Although coins are still undivisible in such system, the effect of coin division is achieved by using transactions. For example there can be created a transaction destroying one coin and creating two new coins assigned back to the owner.

3.1.2 Address

In the original Bitcoin whitepaper, transactions are proposed to be done by "*...signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin.*", where coin is defined as a chain of digital signatures, visualized in figure 3.2. The receiver of funds can verify the chain of ownership by checking the signatures.[3]

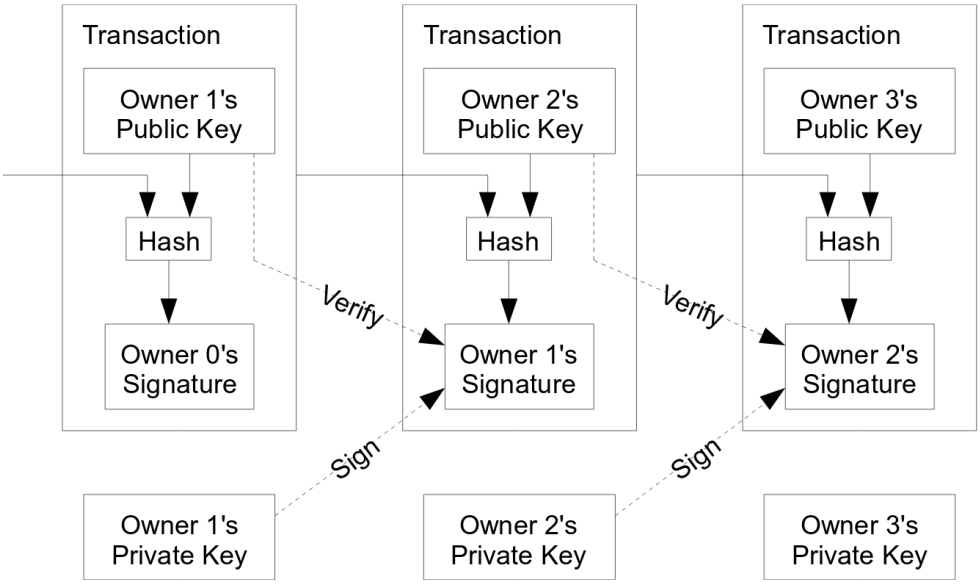

Figure 3.2: Electronic Coin[3]

In reality coins are not send to public keys, rather to an *address*, which is a string of alphanumeric characters. The address resembles the recipient of the funds and is derived from the public key through the use of one-way cryptographic hashing. [4]

Addresses abstract the reciever, making transactions more flexible, similar to paper checks which can used to pay to accounts, pay for bills, or pay to cash. [4]

$$Address = hash(Public\ Key)$$

$$Private\ Key \rightarrow Public\ Key \rightarrow Address$$

$$Private\ Key \not\leftarrow Public\ Key \not\leftarrow Address$$

3.2 Decentralization

Decentralization is a process by which the activities are distributed away from a central location or group. Figure 3.3 visualizes different types of networks. It can be seen that, a centralized organization is like a spider, it is "*controlled by its head, cut off its head and it dies.*" [15]

On the other hand, decentralized organization is more like starfish, it has *no head, and its major organs are replicated throughout each arm. Cut it in half and you get two starfish.*" [15] Decentralized organizations are for example Al-Qaeda, Alcoholics Anonymous, Torrent and Bitcoin.

Distributed currency presents many problems to be solved, such as who will maintain the ledger of transactions or who has authority over which transactions are valid.

The solution to these problems lies in achieving a general agreement of the nodes in the network. Mechanism through wich cryptocurrencies achieve decentralization is a combination of technical methods and clever incentivizing. [10]

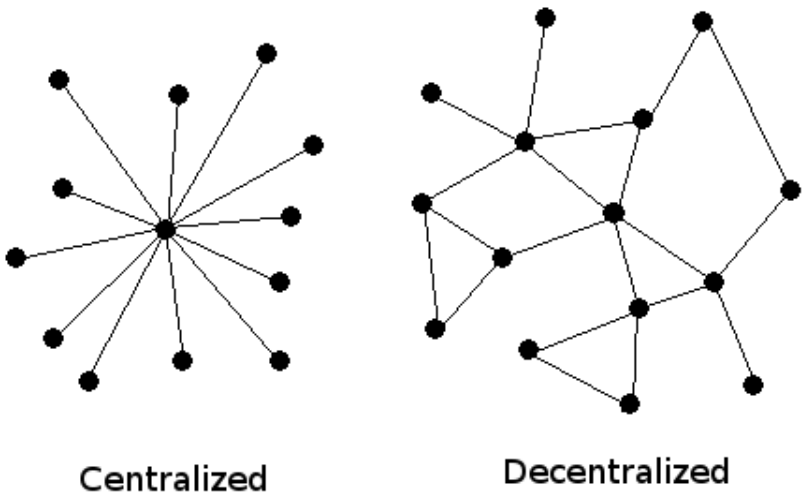

Figure 3.3: Type of networks

3.3 Double Spending

Counterfeit notes could be easily identified by observation, but since digital money, that is not the case anymore. *"The double spending problem is a potential flaw in a cryptocurrency or other digital cash scheme whereby the same*

single digital token can be spent more than once, and this is possible because a digital token consists of a digital file that can be duplicated or falsified." [3]

The transactions couldn't be trusted, if the payer could redirect the money back to himself afterthe payee accepted the transaction. A double spending attack is basically convicing the merchant the transaction is confirmed and then convincing the entire network to believe in an alternative history. The attacker would then receive the goods and keep his money. [16]

As with counterfeit money, such double-spending leads to inflation, devaluates the currency and diminishes user trust. It is usually solved by placing a third party that can verify the money has been spent, e.g. banks, thus creating a single point of failure from both availability and trust viewpoints. "*While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model.*" [3]

To explain this further, imagine three communicating parties (Alice, Bob and Charlie) in a decentralized currency system. Alice has exactly $10 in her balance and buys a coffee for $10 from Bob. However, Alice is dishonest and buys at the same time a slice of pizza from Charlie, even though she doesn't have any balance anymore. Since Charlie doesn't know about Alice buying a coffee from Bob as demonstrates figure 3.4, he is happy to have sold another portion. Charlie later finds out he has been cheated.

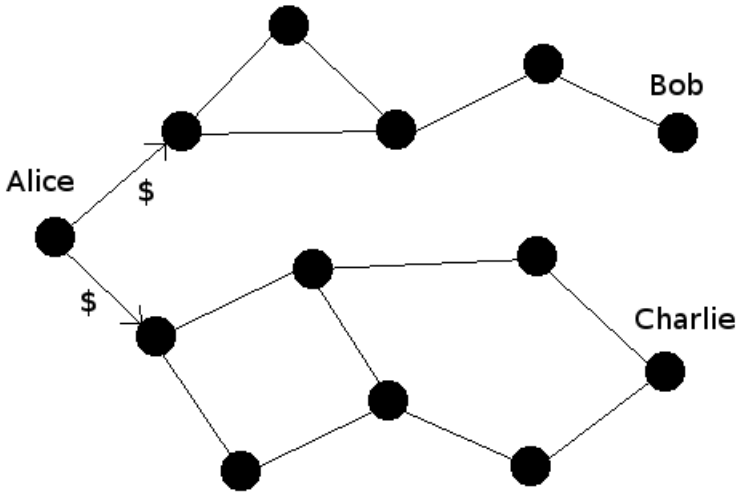

Figure 3.4: Double Spending

3.3.1 Timestamp Server

In the original Bitcoin whitepaper, the double spending attack is prevented by introducing a *timestamp server*. It confirms the transaction itself was not double spent, nor the previous transactions in the chain. [3]

It works by "*taking a hash of a block of items to be timestamped and widely publishing the hash.*" The timestamp proves that the block must have existed at the time (the hash would change by changing the time record). "*... each timestamp includes the previous timestamp in it's hash, forming a chain, with each additional timestamp reinforcing the ones before it*". This is shown in figure 3.5. [3]

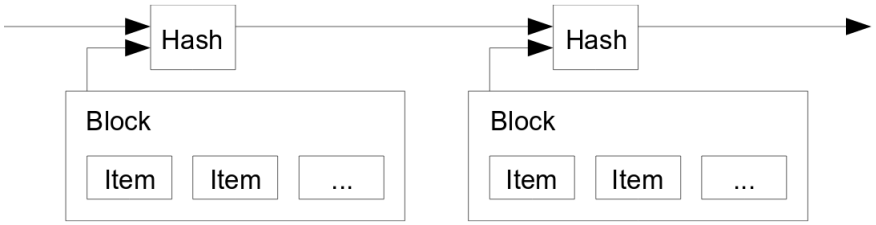

Figure 3.5: Time Stamp Server[3]

In the timestamp server proposal an append-only ledger is described. Because transactions can only be appended, they will remain forever. Nodes can then defend against double spending by requiring a transaction to be written to the ledger before it is accepted to be sure it has not been spent yet. [10]

Double spending could be prevented if this append-only ledger signed by a designated entity, Filip, is shared, so that anyone can check whether a transaction is included in the ledger before accepting it. [10]

Filip of course does not include transactions that reference to already spent coins. If Filip tries to change something in the append-only ledger, it will change all the following hashes. Assuming everyone is keeping track of the last hash, the change would be easily detected. [10]

Although such system prevents double spending and Filip can't create fake transactions, he could still stop accepting transactions from some users or even stop updating the ledger. Most of the early proposals assumed there would be such a central authority. [10]

3.4 Blockchain

In a narrow sense, "*blockchain is a linked list that is built with hash pointers instead of pointers. A use case for a blockchain is a tamper-evident log.*" [10]

Blockchain allows to append data to the end and makes it easy to detect any change of previous data. Each block references the previous block and because it also contains previous block's digest it can be easily verified the previous block hasn't been changed. [4]

The head of the list is a hash pointer pointing to the last block. If the attacker alters the data in the middle of the chain, the hash stored in the next block will not match. The attacker would have to change the following block

and then the next block and so on. As long as we store the head of the list at some location, where it cannot be altered, the attacker will not be able to change any block without being detected. [10]

In a broader sense, blockchain is "*... a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties.*" Records are verified by general agreement of the majority and once the information is recorded, it can never be erased. [17]

Blockchain is a way to reach a distributed consensus, allowing participants to know with certainty that a digital event happened. Cryptocurrencies utilize blockchain to maintain a decentralized database of ordered transactions. [4]

It is a combination of proven technologies like the internet, public key cryptography and a protocol governing incentivization applied in a new way. It allows people to write entries into a record of information, which is managed by the community of users. [17]

On the surface, it seems similar to Wikipedia, which is maintained by the community and anyone can add new articles. However, Wikipedia uses the client-server model, where the client with certain permissions is able to change entries stored on a centralized server. Users browsing Wikipedia then get a copy of the entries from a central database managed by central authority. [18]

The distributed database created by the blockchain technology is fundamentally different. Data is broadcast throughout the network and nodes update their versions of the database. They independantly come to conclusions and the most popular record becomes official. It eliminates the need for a trusted party to facilitate digital relationships. [18]

3.4.1 Cryptocurrency Blockchain

As of now, I have talked about blockchain as a linked list. However distributed blockchains are a bit more complex. In this section I will briefly introduce a typical cryptocurrency blockchain.

A distributed blockchain is not just a linked list, rather a tree. A block has only one parent, however it can have several children during a temporary situation, called fork. It occurs when two or more blocks are discovered simultaneously and gets resolved by appending just one child and discarding the other. [4]

In cryptocurrencies, blocks are identified by the hash of the block header and contain the hash of its parent (previous block) inside its own header. The sequence of hashes linking each block to its parent creates a chain going back to the first block called genesis block, which is encoded within the software. [4]

3.4.1.1 Merkle Root

Only the header of the block is hashed, because the block header contains the merkle root summarizing all the transactions in the block. The spent transactions can be then discarded to save disk space.

"*Once the latest transaction in a coin is burried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree, with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree.*" [3]

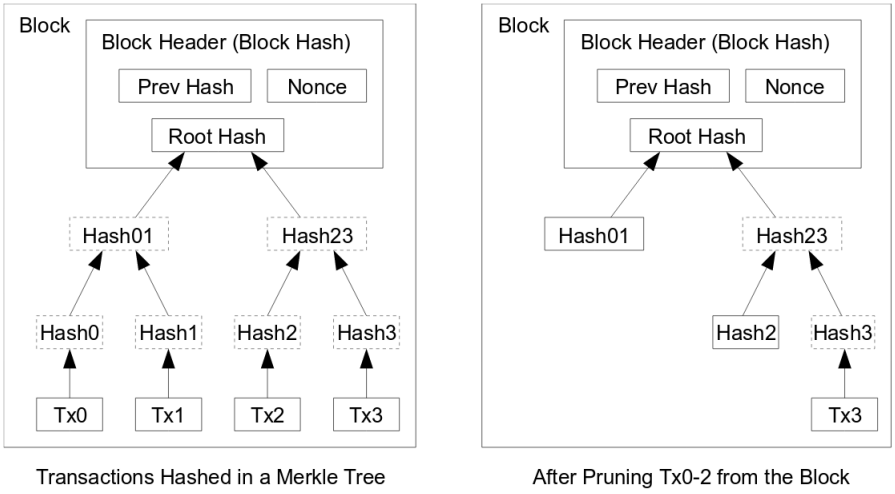

Figure 3.6: Reclaiming Disk Space [3]

Merkle root also facilitates payment verification, without keeping the whole blockchain locally. A user keeps just the block headers of the longest chain and obtains from the network the merkle branch linking the transaction to the block it is timestamped in. "*He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.*" [3]

3.4.2 Nonce

The block header also contains a nonce. Nonce is an arbitrary number that serves as a security against malicious nodes. Nodes search for a nonce, such that the hash of the block produces a hash with a fixed number of leading zeros. It's not possible to find a correct nonce other than trying random values one by one. It is used to prove, that someone exerted a lot of computation power. [4]

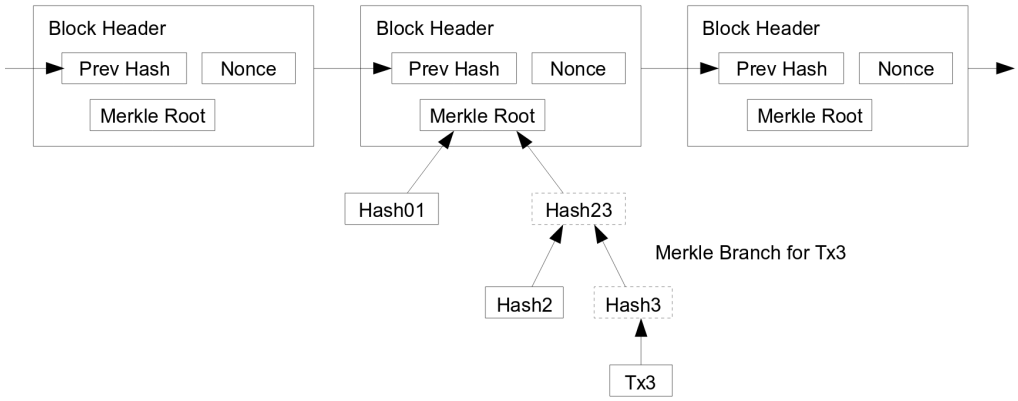

Figure 3.7: Cryptocurrency Blockchain [3]

The block header changes when the parent changes. It creates a cascade effect, causing that a block cannot be altered without recalculation of all consecutive blocks. Because a correct nonce has to be found for each block, the recalculation would require immense computation power. A long chain of blocks makes the blockchain's history practically immutable. [4]

3.4.3 Timestamp and Difficulty

The timestamp provides an approximate information when the block was found. It's main purpose is to know, how often the blocks are created. This fact is then used to establish how many preceding zeros should be required for the hash of a block to be accepted as valid. This requirement for preceding zeros has to be flexible, so that it doesn't take too little or too much time to create new blocks. Both the timestamp of the block and the calculated difficulty are stored in the block header. [10]

3.5 General Agreement

To achieve decentralization, all nodes have to agree on a single valid history. A mechanism to ensure all nodes agree on which transactions are legitimate is needed.

Consensus is a process that facilitates synchronization across a distributed network of untrusted nodes, simply said it's a general agreement of the network. According to [17], the following conditions have to be met to establish consensus:

- No single party should be able to change or influence the records
- Preventing the spread of information across the network should be impossible

A consensus mechanism is a *fault-tolerant* mechanism, which means, that it can recover from failure of a node participating in the consensus. Fault tolerance in distributed system is usually achieved by distributing the shared state across multiple nodes. Updating the shared state happens according to pre-defined transition rules. [17]

Let there be a network of nodes and some of them are faulty or malicious. According to [10] a distributed consensus protocol has to[2]:

- Terminate with all honest nodes in agreement on the value
- The value must have been generated by an honest node

3.5.1 Possibility of Consensus

Most of the literature on distributed consensus is pessimistic. "*With the current state of research, consensus in Bitcoin works better in practice than in theory.*" Due to the fact that cryptocurrencies are in fact currencies, incentives to act honestly can be built in to the consensus protocol. Crytocurrencies thus solve the Byzantine generals problem in a narrow context and not in a general sense. [10]

Consensus in cryptocurrencies also embraces the randomness. The consensus happens over a long period of time and the nodes are never certain that any particular transaction is agreed on, but the probability that a node's view matches the consensus increases exponentially over time. [10]

Nodes in a cryptocurrency network achieve to have a complete history that they can trust with no central authority. The history is assembled independently by each node in the network. Nodes in the network communicating over insecure network assemble the copy of the same public ledger as everyone else. [4]

When Alice wants to pay Bob, she broadcasts the signed transaction to the whole network. The ownership of the coin will change to Bob's whether he is running his node or not, but obviously he gets notified only when he runs it. The nodes keep track about transactions they have heard about, but they accept only those which the network has agreed on using a consensus protocol. Some nodes might have heard about transactions others might not. [10]

Consensus is a hard problem since there could be faults in the network, so running a consensus where all nodes have to participate is impossible. The network is also vulnarable to *sybil attacks*, which are fake identities made by a malicious adversary, which are in fact controlled by him. Further, due to the high latency of the network, there is no global time, which constraints the set of algorithms that can be used. [10]

[2]This is the Byzantine generals problem

3.5.2 Simplified Consensus

This section discusses a simplified consensus mechanism and possible attacks in such system. The contents of this section is based on a chapter *Consensus without identity using a blockchain* from *Bitcoin and Cryptocurrency Technologies* [10].

One way to achieve the consensus is, all nodes would propose at regular intervals its transaction pool to be the next block. The nodes then together decide which block will be appended to the blockchain. If some valid transactions were not included, they will be in the next blocks.

Assume there are no sybil nodes and that we are able to choose a random node from all nodes in the network. With such assumptions an *implicit consensus* is possible with following algorithm:

- New transactions are broadcast
- All nodes collect transactions to a block
- In each round a random node broadcasts its block
- Other nodes accept it only if the transactions in it are valid
- The acceptance of the block is done by including a its hash in the next broadcast block

If a malicious node is chosen to propose a new block, it cannot steel others' coins, because it is not possible to forge the signatures. What the node could do, is a denial attack. The node could not include transactions of a particular user. However, that is not a problem, because an honest node would include the denied transaction in the next round.

The malicious node could double spend coins. Imagine the attacker orders online from a merchant. Assume the latest block includes the transaction for the online order and that it was generated by an honest node. When the merchant sees it is included in the block he knows that the network agreed on it, so he ships the package.

If the malicious node is chosen to generate the next block, it can ignore the last block with the transaction sending coins to the merchant and instead propose another block pointing to the previous block, thus creating an alternative chain. The attacker can then replace the transaction sending the coins to the merchant by a different one sending the coins to the attacker himself, which demonstrates figure 3.8.

The honest nodes will always behave according to the protool and will extend the longest valid branch of the blockchain. From technological view, these transactions are both valid and nodes looking at them have no way to tell which is the morally legitimate transaction.

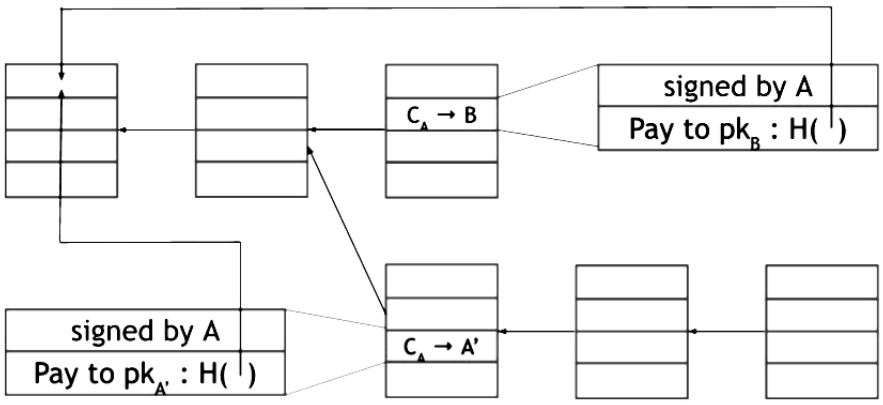

Figure 3.8: Double Spend Attempt[10]

The merchant can protect himself by waiting for more blocks than just one. Shorter branch with the double spend attack has low probability to catch up due to the fact that several malicious nodes would have to be chosen in succesion.

3.6 Mining

Present cryptocurrencies take advantage of various concepts for securing the system and reaching the consensus, but only the proof of work based cryptocurrencies are considered secure. This chapter focuses on a proccess of mining.

Miners record transactions to the global ledger. Transactions added to the blockchain are considered confirmed, which allows the new owners to spend what they received in those transactions. Miners compete to find a solution of a difficult problem, called proof of work, to receive a reward. [4]

3.6.1 Incentives

Mining secures the system and enables network-wide consensus without a central authority. Mining is incentivized by a reward that aligns the actions of miners with the security of the network. It is the incentive system by which the network is decentralized. [4]

Once a solution is found, it is broadcast, verified and appended to the blockchain. Miners start to mine a new block of transactions which can be chosen arbitrarily. The transactions are accompanied by a small fee, that the miners could claim, so the miners usually choose in such a way to collect as many fees as possible. A lot of attacks are infeasible if the majority of miners are following the protocol. [4]

It was assumed, there are no sybil nodes in the network, thus we are able to pick a random node. A good enough algorithm to establish consensus without such assumptions is possible with incentivizing the honest behaviour, which is possible only because the application build through this consensus is in fact a currency. [10]

Since the coins are accepted by others only if they are in the longest chain, it incentivizes nodes to behave in such a way, that others will extend their blocks. [4]

The users can be incentivized through new minted coins or through through transaction fees. The creator of a transaction can make the total output less than the total input. Whoever then includes such transaction into a block, gets the difference. [4]

3.6.2 Proof of Work

To pick a random node and decrease the probability for choosing a sybil node, "*... we can approximate the selection of a random node by instead selecting nodes in proportion to a resource that we hope that nobody can monopolize*". [10] If the resource is computing power, it is a proof of work system, but it could be in proportion to ownership of currency, that is called proof of stake [17].

"*Another way of understanding this is that we're allowing nodes to compete with each other by using their computing power, and that will result in nodes automatically being picked in that proportion.*" Yet another way to look at it is that it's difficult to create new identities. [10]

Proof of work is highly wasteful, around 3 billion dollars are spent for electricity by miners each year trying to solve deliberately hard problem just to establish the consensus [17].

Although the wastefulness may be a small price to pay for the reward of a decentralized global currency network, there are other options to establish the consensus like proof of stake or proof of burn. However, they are still considered unworkable alone and have to be combined with proof of work. [17, 4]

3.6.2.1 Searching for Nonce

The idea behind Proof of Work is to have nodes solve a computationally expensive problem easily verifiable by others. The computational problem in proof of work currencies is searching for a hash, such that it begins with a given number of zero bits. [4]

It is implemented by adding an arbitrary number called a nonce to a block, which is repeatedly incremented, until a value is found, that produces a hash with the required number of zero bits. The time to find the digest grows exponentialy with the number of required zero bits, but the solution can be

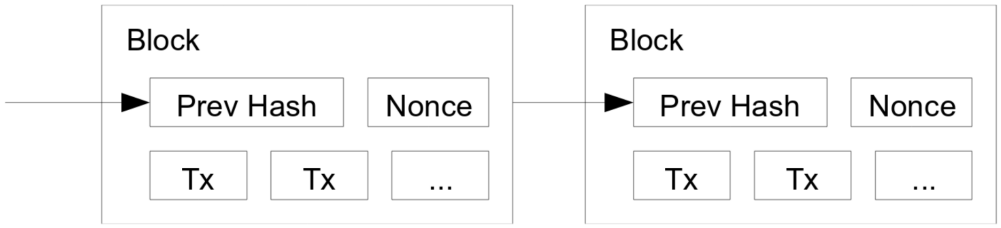

Figure 3.9: Proof of Work [3]

verified easily by just hashing the block. The block could be changed only by redoing all the work. [3]

Nodes search for a nonce such that when hashed together with the block, the output hash is a number that falls into a target space.

$$Hash(\text{block with set nonce}) < target$$

"*The idea is that we want to make it moderately difficult to find a nonce that satisfies this required property, which is that hashing the whole block together, including that nonce, is going to result in a particular type of output.*" [10]

This completely does away with the requirement to pick a random node. Nodes independently compete to find the hash. If a node is lucky and finds a nonce that satisfies this property, it gets to propose the next block. That's how the system is decentralized, there is noone deciding which node proposes the nextblock. [10]

Searching for hashes is probabilistic, due to the fact that noone can predict which nonce will result in an acceptable hash, because the hash function spreads the outputs evenly over its output range. The only way to do it is to try different nonces until one succeeds. [7]

Hash rate is the unit of measurement for the amount of computing power a proof of work cryptocurrency network is consuming in order to be continuously operational.

The *difficulty* is the cost to find the correct nonce and it has to be parameterizable. The difficulty is maintained by the protocol and it's calculated from the timestamps of the blocks. As the difficulty increases, there have to be more zeros at the start of the hash number. The probability of finding a lower hash value is smaller and so miners have to test more nonces. [4]

All the nodes recalculate the difficulty in regular intervals. If more miners are coming in, more blocks are going to be found than expected. Nodes will automatically readjust the difficulty, so it takes the same time. If blocks were to come in long intervals, it would be too inconvenient to wait for the confirmation of transactions. [10]

3.6.3 Fork

Blockchain is a decentralized data structure and nodes might have different perspectives of it. Nodes try to extend the longest chain of blocks representing the most work. The network converges to a consistent state when the nodes select the chain with most cummulative work done – the one where the sum of the work recorded in every block in that chain is the highest. [4]

As stated before, in a decentralized network the blockchain is a tree with the root being the genesis block. A branch is a path from a leaf block to the genesis block, it represents one history of the transactions. Each branch is internally consistent, however one branch can include a transaction which spends the same coins in another branch. [16]

A *fork* happens when there are two branches of the same length. This occurs whenever two different nodes find a correct nonce within a short time frame. When they discover the solution for their candidate blocks, they broadcast it to their neighbors who begin propagating it further accross the network. Every receiving node will append the block to its blockchain. Some nodes append one block first, while other nodes append the other one, thus two versions of the blockchain emerge. [4]

It is agreed that each node considers the longest chain as the valid chain. In case of two branches of the same length, the first known to the nodes is considered valid. It is expected that nodes creating blocks reference the longest chain, because if not, their reward for the newly created block would not be considered valid. Figure 3.10 demonstrates the fork. [16]

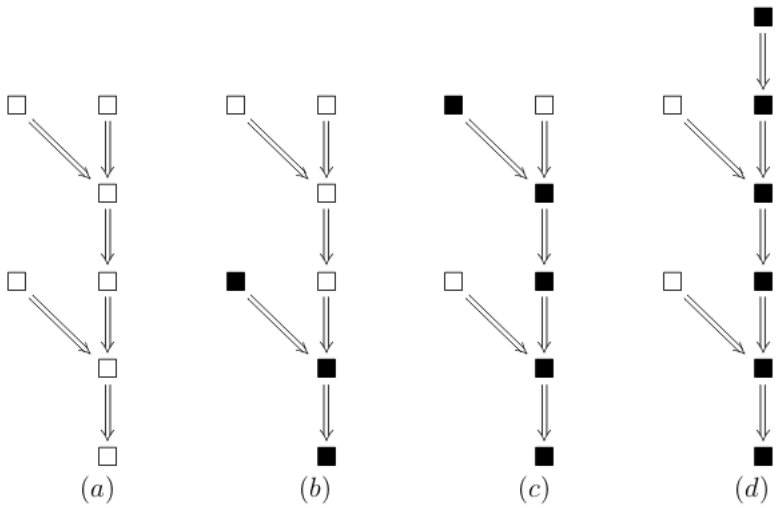

Figure 3.10: Branch selection[16]
(a) a possible structure of a blockchain, (b) an invalid branch, (c) a branch which might be valid, (d) a case when the previously possibly valid branch got invalidated

3.6.4 Dishonest Mining

In this section I will describe a problem, that can arise if someone has too much computing power in a decentralized cryptocurrency network and what are the consequences. Consensus mechanism is theoretically vulnerable to attack by miners that attempt to use their hashing power to dishonest ends [4].

"*The consensus mechanism depends on having a majority of the miners acting honestly out of self-interest.*" If an entity or a group is able to control the majority of the hash rate it can potentially cause a network disruption. It could intentionally exclude or modify the ordering of transactions, or even reverse transactions it made while being in control. [4]

A particular attack scenario is called the *51% attack*. A group of miners that control a majority of the total network's hashing power mine the majority of the blocks and thus can can cause deliberate forks. "*While in theory, a fork can be achieved at any depth, in practice, the computing power needed to force a very deep fork is immense, making old blocks practically immutable.*" [4]

In a senario where an attacker is able to mine blocks faster than the honest chain and generates an alternate chain, it does not allow him to make arbitrary changes, such as minting new coins or spending someone elses money. Honest nodes will not accept blocks containing such transactions. He can only take back money he already spent. [3]

The attacker wants to make the recipient believe he paid him for a while, but then change the transaction to pay himself. The receiver will find out he had been cheated, but the attacker hopes it will be too late. Once the transaction is broadcast, the attacker secretly starts to work on an alternative chain with his modified transaction, which sends the coins to himself instead of to the original recipient. Figure 3.11 shows the scenario of the attack. [3]

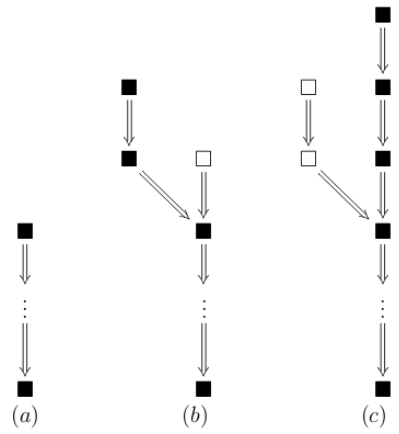

Figure 3.11: Double Spending in 51% Attack[16]
(a) Original state of blockchain. (b) The merchants sends the product.
(c) Attacker manages to get his branch to be longer.

3.6.4.1 Probability

The competition between the attacker and the honest chain can be described as a series of ±1 steps, where +1 means extending the honest chain by one block and -1 means extending the attacker's chain. [3]

The attacker usually finds himself in a situation, where the network has a transaction crediting coins to the merchant in the longest branch. The attacker has a branch, which is shorter. Both, the attacker and the honest nodes, are trying to extend their branches. [16]

q is the probability the attacker finds the next block, p is the probability an honest node finds the next block. According to [3], the probability q_z of an attacker ever catching up from z blocks behind is:

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

When the attacker dominates the block creation, he always succeeds no matter the disadvantage. However "*When $q < p$, the probability of success decreases exponentially with the disadvantage z; the lower q is, the faster the decay.*" [16]

Figure 3.12 visualizes the calculations from [16] on the probability of a succesful double spend attack.

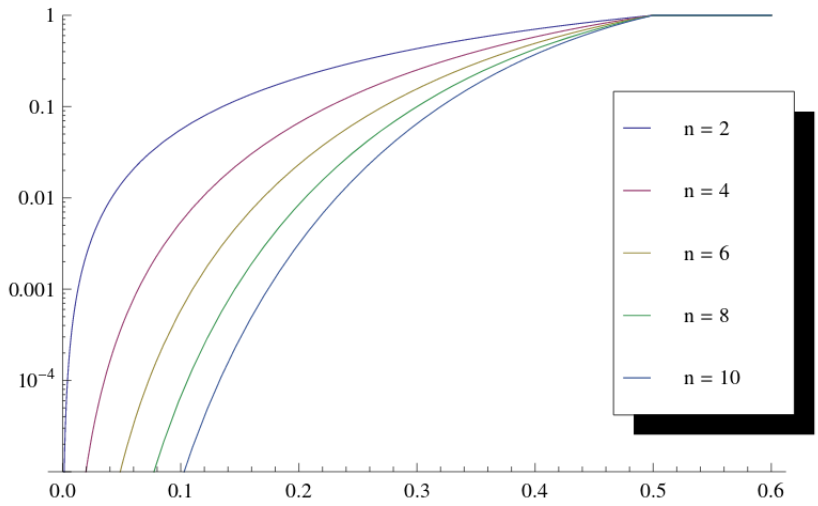

Figure 3.12: Probability of a succesful double spend attack [16]
The probability of a succesful double spend attack, as a function of percentage of the total hashrate of an attacking node. Value n signifies the number of confirmations.

Chapter 4

Fitcoin

Fitcoin is a barebones cryptocurrency and serves for educational purposes. The endeavors were directed toward keeping it as simple as possible, while incorporating important aspects of a cryptocurrency. Due to the widespread use of Unix-like operating systems in academic environment, it is desired for the cryptocurrency to be portable accross such systems. The main priority is to demonstrate the concept of cryptocurrencies.

Since Fitcoin is a utility running on Unix-like operating systems, the C programming language was an obvious choice for development. To keep a low entry point, dependencies were kept to a minimum. The only library used in the implementation is LibreSSL for cryptographic functions.[3]

To compile Fitcoin, one needs Unix-like operating system, the GNU Compiler Collection and OpenSSL/LibreSSL library. Some open-source tools were in consideration to enhance the development flow like CMake and Check for controling the software compilation process and defining unit tests. However, they were discarded to keep the project plain and simple, both of these development practices were handled by using simple makefiles.

4.1 Communication

Fitcoin consists of three parts: `ftcd` (Fitcoin Deamon), which represents a network node and it communicates with other such nodes over the network. The `ftcd` also forks a child process called miner, which is responsible for mining and communicates with it using Unix signals. The third part is a command line utility `ftcl` (Fitcoin Control), which can send commands to `ftcd` over a local socket.

[3]The OpenBSD project forked LibreSSL from OpenSSL in 2014 as a response to the Heartbleed security vulnerability, with the goal to improve security.

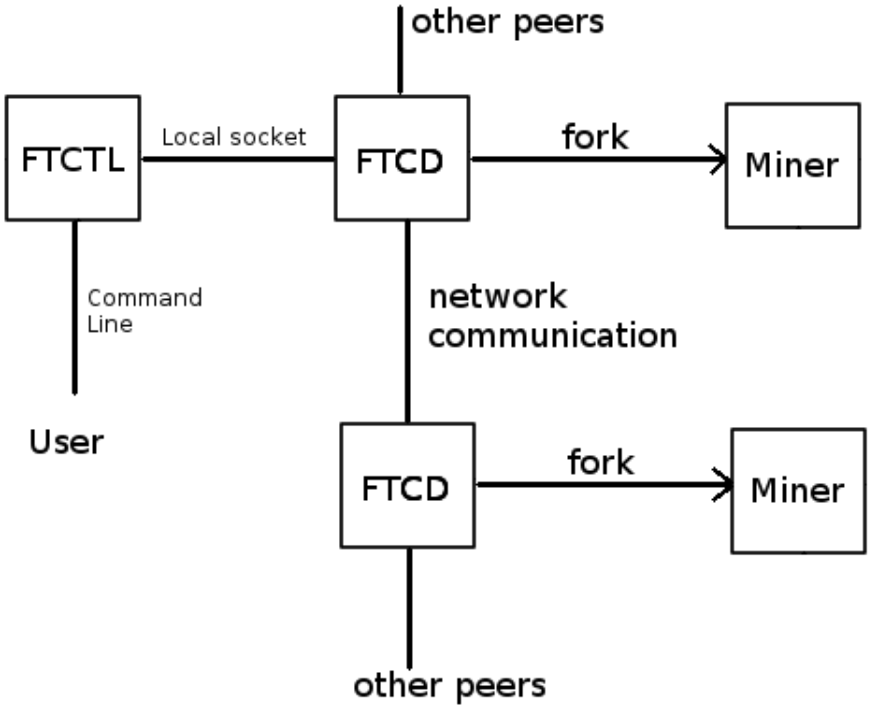

Figure 4.1: Fitcoin Communication

4.1.1 Peer to Peer

The Fitcoin Daemon starts to listen on port 8888 and whenever a peer tries to establish a connection, it is saved to a double linked list. This list of peers is saved to the peer file upon exiting the daemon. The next time the `ftcd` is started, it loads the IP addresses from the file and tries to connect to them.

4.1.2 Fitcoin Client

The Fitcoin Daemon listens on a local socket `/tmp/ftc` for new connections from the Fitcoin Client. Whenever a connection is requested, that is by starting the Fitcoin Client on the same machine, `ftcd` assigns a new socket to such connection and listens for incoming messages.

4.1.3 Fitcoin Miner

The communication of the miner with the Fitcoin node is visualized in figure 4.2. `ftcd` creates a temporary file `/tmp/newblock.[ID]`, where `[ID]` is a unique identifier and then forks the miner. The miner looks into the folder `~/.ftc/fresh` where the transactions that have yet not been mined are stored and starts to mine a block made of these transactions.

Each unix program by default sends a signal `SIGCHLD` to the parent process when it finishes. This is utilized in the miner to `ftcd` communication. After the miner mines a block, it writes it to the temporary file and sends the `SIGCHLD` signal to `ftcd`, which processes the block and eventually might fork miner again.

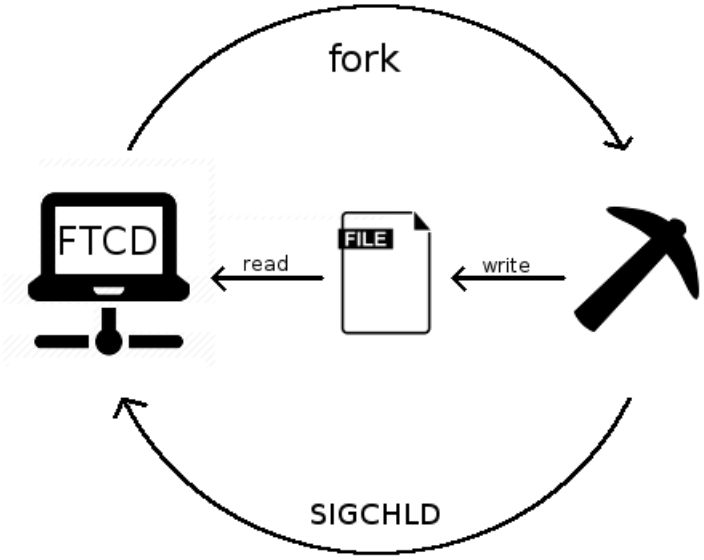

Figure 4.2: Fitcoin Miner communication

4.2 Blockchain

In this chapter, the structure of the Fitcoin's blockchain is described. The blockchain is saved on the file system in big endian byte order as individual files.

4.2.1 Block

A block is a data structure containing transactions. Fitcoin follows similar structure to Bitcoin's block, it consists of a header and a list of transactions. Its unique identifier is a hash made by hashing the block header. A block can also be identified by its height, where the genesis block has height 0 and each child increases its height by one. Although height can be derived from the blockchain and storing it is unnecessary, opposing to Bitcoin it is part of the header in Fitcoin to facilitate simpler implementation. This is how the block data stucture looks in Fitcoin:

```
struct block{
        uint32_t                size;
        char                    prev[32];
        uint32_t                height;
        uint32_t                ts;
        uint32_t                nonce;
        char                    target[32];
        char                    merkle[32];
        uint32_t                numtx;
        unsigned char   data[];
};
```

Figure 4.3: Block structure

`Size` is the total block size including the field itself and the size of all of the block's transactions. `Prev` is the the SHA256 digest of its parent block. `Height` is the current block height. `Ts` is a unix timestamp (Number of seconds since January 1st, 1970 UTC). `Nonce` is a number that makes the hash of the block header lower than the `target`. `Target` is a 32 bytes long number calculated from the timestamps of previous blocks, so that it takes fixed amount of time to mine each block. `Merkle` is the merkle root of the transactions in the block. `Numtx` conveys how many transactions are in the block and `data` contains the actual transactions. The `target` and `prev` are character arrays, however they are often converted to a special type *big number* provided by LibreSSL to be able to do mathematical operations with it.

4.2.2 Input/Output structure

The building block of a Fitcoin transaction is a transaction input/output. It is recorded on the blockchain and consists of an amount of Fitcoin and an address. User's balance is a sum of all unspent transaction outputs which are associated with an address generated from a public key that is controlled by the user owning the corresponding private key.

A transaction output is used as a transaction input for another transaction. It is always spent in its entirety by the next transaction. However, change can be generated to send coins back. A data structure shown in figure 4.4 is used for such purposes.

```
struct io {
        char            addr[32];
        uint32_t        amount;
};
```

Figure 4.4: Input/Output data structure

The `io` structure contains an address `addr`, which is always a SHA-256 hash of a public key. It also contains `amount`, which signifies the number of tied coins to that address. The data structures can represent three different things depending on its context:

- transaction input
- transaction output
- address balance

In the transaction output, the address is a destination address to which the coins are sent. In the transaction input, the address signifies the source address of the coins. If the structure is used as an address balance, the address signifies the amount of coins belonging to the address.

The address balance is calculated on the startup by going through all blocks in the blockchain and each transaction in it, suming up all unspent transaction outputs belonging to an address. These address balances are then saved to the coins directory and are recalculated every time a new block is appended.

4.2.3 Transaction structure

Transactions are data structures that encode the transfer of value between participants in the system. It carries information about its size, number of transaction inputs, number of transaction outputs and the actual input/outputs in an array.

```
struct tx {
        uint16_t        size;
        uint8_t         numi;
        uint8_t         numo;
        struct io       io[];
};
```

Figure 4.5: Transaction structure

`Tx` stands for transaction and it represents a transfer of coins. Opposing to the `struct io`, there is a need to keep track of the size, since each transaction can have different amount of transaction inputs and outputs. As mentioned before, there is no difference between transaction inputs and transaction outputs themself, they are both represented by the same structure. Their meaning is determined by the number of inputs and number of outputs. The first `numi` structures are inputs, and the other `numo` structures are outputs. A valid input is an output from another transaction, that has not yet been spent.

Chapter 5

Implementation

It is difficult to demonstrate the concepts in C programming language, therefore a pseudocode with C style programming conventions is used throughout this chapter to demonstrate the concepts better.

5.1 Ftcd

Fitcoin Daemon is the most important part of the whole Fitcoin ecosystem. It is the full node that communicates with the outside world. The whole structure of the program is described in pseudocode in Algorithm 2.

Algorithm 2 Ftcd

```
init directories
if blockchain does not exist then
  create genesis block
end if
check blockchain integrity
calculate address balances
start miner
load wallet
connect to peers and Ftctl
loop
  if someone tries to connect then
    sockets.insert(new socket for such communication)
  end if
  for all sockets do
    handle incoming message
  end for
end loop
```

5.2 Miner

Miner is the piece of software that bundles transactions together and creates valid blocks from them. First the algorithm for merkle root calculation and algorithm for calculating difficulty are described, which are needed for the creation of valid blocks, then the actual mining is described in Algorithm 5.

5.2.1 Merkle Root

Merkle root is utilized in several different ways in cryptocurrencies. In Fitcoin it's only purpose as of now is to simplify the process of hashing a block. Since each block contains merkle root, which is calculated from all the transactions in a block, only the header needs to be hashed.

The Algorithm 3 used for merkle root calculation is recursive. To not dive into unnecessary technicalities, in this pseudocode it is assumed 2^n number of transactions are on input. Of course the actual implementation in Fitcoin does not have this assumption.

Algorithm 3 Merkle root calculation

$\oplus$ means concatenation

Function Merkle is:
Input: Transactions $T_1 \dots T_N$, where $N = 2^n$
Output: Merkle root

1: **for** $i \leftarrow 1$ **to** N **do**
2: $D_i \leftarrow \text{hash}(T_i)$
3: **end for**
4: **if** N = 1 **then**
5: **return** D_1
6: **else**
7: **for** $i \leftarrow 1$ **to** $N/2$ **do**
8: $D_i \leftarrow \text{hash}(D_{i*2-1} \oplus D_{i*2})$
9: **end for**
10: **end if**
11: **return** merkle$(D_1 \dots D_{N/2})$

5.2.2 Difficulty

Valid blocks must produce a hash below the target, that is specified by the Fitcoin protocol. This target is recalculated every 12 blocks so that mining a single block takes about 30 seconds. It is calculated by this formula:

$$\text{next target} = \text{current target} \frac{\text{time to mine last 12 blocks}}{30 * 12}$$

Algorithm 4 is called every time the miner starts mining and it readjusts the difficulty every 12^{th} block. The `AdjustTarget` in the pseudocode is the equivalent to the formula stated above.

Algorithm 4 Target recalculation

Input: Current target T, Blockchain of blocks $B_0, B_1, B_2 \ldots$
Output: New target

1: $h \leftarrow$current height
2: **if** $h = 0$ **or** h mod $12 \neq 0$ **then**
3: new target $\leftarrow$ current target
4: **return**
5: **end if**
6: elapsed$\leftarrow$ B_h.timestamp - B_{h-12}.timestamp
7: AdjustTarget(current target,new target,elapsed,30*12)

5.2.3 Mining

Algorithm 5 shows the mining process. The mining loop is just about trying different nonces until a correct one is found.

Algorithm 5 Mining a block

Input: Unconfirmed transactions $T_1 \ldots T_N$
Output: Block B with correct nonce

1: **for** $i \leftarrow 1$ **to** N **do**
2: append serialized T_i to $B.data$
3: **end for**
4: $B.merkle \leftarrow$ merkle$(T_1 \ldots T_n)$
5: $B.prev \leftarrow$ highest block hash
6: $B.height \leftarrow$ highest block height + 1
7: $B.target \leftarrow$ calculate new target
8: $B.nonce \leftarrow 0$
9: $B.ts \leftarrow$ seconds since January 1st, 1970 UTC
10: **loop**
11: **if** hash$(B) < B.target$ **then**
12: **return** B
13: **end if**
14: $B.nonce \leftarrow B.nonce + 1$
15: **end loop**

5.3 Portability

The requirement is not only that Fitcoin runs on different operating systems but also on different hardware devices. To stay portable data is converted to big endian byte order before it is saved on disk and vice versa when reading from the file system. Such conversions are also needed when communicating over the network and also when hashing transactions and blocks.

This is solved by utilizing internet operations functions `htonl`, `htons`, `ntohl` and `ntohs` found in `<arpa/inet.h>` header. Function `htonl` converts long integer from host to network byte order (big endian), `htons` does the same for short integer. The funcsions `ntohl` and `ntohs` do the same in the opposite direction.

An example of converting a transaction structure to big endian byte order in C programming language is shown in Algorithm 6. The algorithm for conversion back to the host byte order is the same, but instead of `htonl` and `htons` functions, `ntohl` and `ntohs` are used. All of these functions are designed to work in place.

Algorithm 6 Transaction conversion to big endian

```
1  void htontx(struct tx *tx){
2    uint32_t i,len;
3    struct io *io;
4    if (NULL == tx)
5      return;
6    tx->size = htons(tx->size);
7    len = tx->numi + tx->numo;
8    for (i=0,io=tx->io;i<len;i++,io++)
9      io->amount = htonl(io->amount);
10 }
```

Chapter 6

Testing and Start Up

The importance of testing depends on the purpose of the program. Although the cryptocurrency Fitcoin is not meant to be used in the real world, there have been done at least following tests:

- Fitcoin was tried to run on different operating systems to test the software portability
- File system operations were tested on machines with different hardware architectures
- Unit tests were done for a selected set of functions

Fitcoin should work accross major Unix based operating systems. On top of that, it should correctly run without on systems with different hardware specifications. The focus was mainly on the correct byte order while working using the file system and network.

Since several people worked on Fitcoin, it had to be tested that everything works well together throughout the development. However, thorough integration tests were not done.

Only unit tests of functions utilized in the mining software were done as part of this thesis, however other students working on Fitcoin did write unit tests for the other areas of Fitcoin.

6.1 Portability

Fitcoin was tested on various machines. The tests are by no means thorough, but the most important issues were tested, such as that the program works correctly on big endian architecture as well as on little endian ones. Fitcoin was tested on following systems:

- OpenBSD macppc (big endian)

- SunOS SPARC-Enterprise-T5120 (big endian)
- OpenBSD armv7 (little endian)
- OpenBSD amd64 (little endian)
- FreeBSD i386 (little endian)
- NetBSD i386 (little endian)
- MacOSX amd64 (little endian)
- Linux Debian amd64 (little endian)

Fitcoin runs succesfully on all these machines, although the compilation process is not flawless. Problems can be caused by various issues, for example by old version of OpenSSL/LibreSSL, different header file definitions etc., but they can be easily solved, since Fitcoin has very little dependencies.

I recommended to compile the program with GCC 6.3.0. Other versions of GCC were also tested and some of them output warnings, however the program still compiles correcly. SunOS 5.11 comes with an ancient version of OpenSSL. Extra libs for socket() etc. are needed. To be able to compile on SunOS 5.11, one has to install a recent LibreSSL into `$HOME`, then build with: `make CC=gcc CFLAGS="-I$HOME/include" LDFLAGS="-L$HOME/lib -lcrypto -lsocket -lnsl -Wl,-rpath,$HOME/lib"`

There are also problems when compiling on Debian. The only solution that worked for me was to not specify `-std=c99` during compilation. This solution is not ideal. According to the manual pages, one should use `-D_XOPEN_SOURCE` and `-D_USE_MISC` flags during compilation however, that didn't fix the problem.

It was tested that the Fitcoin Daemon can connect to other peers and that the miner mines valid blocks on all systems. It was also tested that the genesis block looks exactly as in figure 6.1 on each tested machine.

```
$ hexdump -C ~/.ftc/chain/000000.blk
000000  00 00 00 9c 00 00 00 00  00 00 00 00 00 00 00 00
000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000020  00 00 00 00 00 00 00 00  5c 1b a7 40 00 00 00 00
000030  00 00 00 ff ff ff ff ff  ff ff ff ff ff ff ff ff
000040  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff
000050  28 ff 19 96 79 0c b8 9d  58 02 69 3b 46 a9 10 4e
000060  a5 55 2e f9 1f 39 b3 82  ce f0 29 55 ca 8b 86 43
000070  00 00 00 01 00 28 00 01  33 88 e2 b4 a6 d8 a4 8e
000080  d0 48 b8 12 57 f2 3b eb  66 4c bf de 9c b8 2e 96
000090  e0 b5 42 76 16 0c df 31  00 00 04 00
00009c
```

Figure 6.1: Genesis Block

6.2 Unit Tests

Unit tests including the functionality of the miner are available and can be started by following command:

```
$ make test
```

The following functionality is tested:

- withdraw and deposit
- transactions
- hashing
- difficulty
- blocks

Compilation

The process of compilation is managed by a Makefile. There are several possible commands that are implemented. The first and most important one is the the install command. It Installs the Fitcoin Client (Ftctl) and Fitcoin Daemon (Ftcd), so that it can be started anywhere. It also installs the manual pages for Ftctl as well as Ftcd. Other commands are just a subset of this one.

To install everything:

```
$ make install
```

To compile Ftcd, Ftctl and run the unit tests:

```
$ make all
```

To build the Fitcoin Client:

```
$ make ftctl
```

To build the Fitcoin Daemon:

```
$ make ftcd
```

To run the unit tests:

```
$ make test
```

To clean the build folder:

```
$ make clean
```

Running the program

Fitcoin Daemon can be started by typing `./ftcd` in the build directory or `ftcd` after installing it. The daemon does not have any user interface. One has to use `ftctl`, which is a command line interface communicating locally with Fitcoin daemon over a local socket `/tmp/ftc`. To be able to start `ftctl`, `ftcd` has to be already running. One can send following messages to `ftcd` through `ftctl`:

```
getpeers
addpeer
delpeer
newaddr
pay
quit
```

The commands are quite self explanatory. The `getpeers`, `newaddr` and `quit` commands don't accept any arguments. The `addpeer` and `delpeer` commands register or delete a peer and their argument is the IP address of the peer. The `pay` command accepts two arguments, first is the address of the payee specified as a string made of 64 hexadecimal characters, the second argument is the amount to be sent to that address specified as a decimal number.

When first running the Fitcoin Daemon, there are no transactions made yet, so the miner will will not mine. I will explain how to make new transactions to test the miner. First, compile and run the Fitcoin Daemon. On startup, the Fitcoin Deamon initializes the directory structure in `~/.ftc`. Now kill the Fitcoin Daemon and copy the `wallet` folder from the accompanied flash disk to `~/.ftc/wallet`.

The wallet folder contains the private key that corresponds to the public key in the genesis block. Without this key, you would not be able to create new transaction. Assuming you didn't connect to any peers, the only block in your

blockchain is the genesis block, which creates the first 1024 coins and sends it to `3388e2b4a6d8a48ed048b81257f23beb664cbfde9cb82e96e0b54276160cdf31`.

Now that you have the private key to the address in the genesis block, you can spend the available coins. Run the Fitcoin Daemon again and then start the Fitcoin Client. Type `newaddr` in the Fitcoin Client, which instructs the Fitcoin Daemon to create another key in the wallet folder.

The Fitcoin Client will output a new address, which you can use in the `pay` command to create new transaction, for example following command sends 512 coins to the address `fa341325`: `pay fa341325 512`. Keep in mind, that you can't spend more than 1024 coins. By running this command, the Fitcoin Dameon will create a transaction in `~/.ftc/fresh`. The miner should create a block from this single transaction and should start mining it.

New blocks will start appearing in `~/.ftc/chain`. These blocks consist of all of the transaction from the `~/.ftc/fresh` folder. Since Fitcoin Daemon does not remove the mined transactions from the folder, the miner will infinitely mine the same transactions. Apart from the fact, that the blocks are made from the same transactions, they are otherwise valid. This issue could be easily fixed, but it comes in handy for the demonstration of the miner software.

Data Storage

The Fitcoin Daemon creates a hidden folder `.ftc` in the home directory of current user on startup. The structure of the folder is shown in 6.2:

```
chain ..... The individual blocks forming a blockchain
coins ..... Addresses and the amount of coins belonging to them
fresh ..... Transactions which are not yet mined
peers ..... I.P. addresses of known peers
wallet ..... The private keys
ftcd.log ..... The default log file
```

Figure 6.2: Structure of the Fitcoin directory

The `chain` directory contains files `000000.blk`, `000001.blk`, ... These files represent individual blocks, where each block refers to the one before him. These files might vary in size, since each block contains different amount of transactions.

The `coins` directory contains files with 64 characters long names, which represent the hexadecimal representation of an address. Each file is exactly 4 bytes in size and contains one single integer recording the amount stored at that address.

The `fresh` directory contains files of different sizes with 67 characters long names. The files represent transactions that haven't been mined yet. The first

64 characters are a SHA-256 hash of the saved transaction in big endian format followed by a suffix ".tx".

`peers` is a file containing known peers in form of I.P. addresses separated by a newline. The Fitcoin Daemon dumps all addresses it knew about to this file on successful exit.

The `wallet` directory contains files, where each one stores a single private key in hexadecimal format. The names of the files are again 64 bytes long addresses,i.e., SHA-256 hashes of the corresponding public keys.

Conclusion

In this thesis, I briefly introduced important cryptography concepts facilitating the cryptocurrency technology and analyzed the inner workings of a cryptocurrency. I focused mainly on the concepts relating to the miner. Decentralization poses challenges, which can be solved by achieving a general agreement of the network. The general agreement emerges by maintaining a history of timestamped blocks of transactions accompanied by solutions to artificially difficult problems.

Miner is a node running special software that searches for such solutions. I implemented such software in the C programming language and used the LibreSSL library of cryptographic functions. The miner successfully creates valid blocks from new transactions and adjusts the difficulty for the network at regular intervals.

The miner was integrated to the barebones cryptocurrency Fitcoin and it was tested on major Unix based operating systems and a variety of hardware, including little endian and big endian architectures.

Fitcoin was developed for educational purposes. Its implementation is simple, portable, and extensible. It demonstrates the important concepts of a cryptocurrency and provides an environment to test new theories.

Bibliography

[1] WEI, Dai. *b-money, an anonymous, distributed electronic cash system.* [online]. 1998. [visited on 2019-4-3]. Available from: `http://www.weidai.com/bmoney.txt` Archived email sent to cypherpunk mailing list.

[2] WEI, Dai. *Making money with Bitcoin?.* [online] 2011 [visited on 2019-4-19]. Available from: `https://www.lesswrong.com/posts/ijr8rsyvJci2edxot/making-money-with-bitcoin`. Wei Dai comments on the article.

[3] NAKAMOTO, Satoshi. *Bitcoin: A peer-to-peer electronic cash system.* [online]. 2008. Available from: `https://bitcoin.org/bitcoin.pdf`

[4] ANTONOPOULOS, Andreas. *Mastering Bitcoin: unlocking digital cryptocurrencies.* [online]. O'Reilly Media, Inc., 2014. Edition 1. ISBN-13: 978-1491954386. Available from: `https://github.com/bitcoinbook/bitcoinbook/releases/tag/Edition1Print2`

[5] LAMPORT, Leslie and SHOSTAK, Rober and MARSHALL, Pease. The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems.* ACM, 1982, 4(3), 382–401.

[6] SILVERMAN, Joseph and PIPHER, Jill and HOFFSTEIN, Jeffrey. *An Introduction to Mathematical Cryptography.* New York: Springer-Verlag, 2008. Edition 1. eBook ISBN: 978-0-387-77994-2 DOI:10.1007/978-0-387-77993-5

[7] KATZ, Jonathan and MENEZES, Alfred J. and OORSCHOT, Paul C. van and VANSTONE, Scott. *A method for obtaining digital signatures and public-key cryptosystems.*, CRC Press, 1996. 1st edition. ISBN-10: 9780849385230

[8] PAAR, Christof and PELZL, Jan. *Understanding cryptography: a textbook for students and practitioners*. Springer, 2010. ISBN-10: 3642446493

[9] BECKER, Georg. *Merkle signature schemes, merkle trees and their cryptanalysis*. [online]. Ruhr-University Bochum, 2008 [visited on 2019-3-14]. Available from: `https://www.emsec.ruhr-uni-bochum.de/media/crypto/attachments/files/2011/04/becker_1.pdf`

[10] NARAAYANAN, Arvind and BONNEAU, Joseph and FELTEN, Edward and MILLER, Andrew and GOLDFEDER, Steven. *Bitcoin and cryptocurrency technologies: A comprehensive introduction.* Princeton University Press, 2016. ISBN13: 9780691171692

[11] MALONEY, Mike. *From Bitcoin To Hashgraph (Documentary) Hidden Secrets Of Money Episode 8.* [online video] 2017 [visited on 2019-3-12]. Available from: `https://www.youtube.com/watch?v=SF362xxcfdk`

[12] DWORK, Cynthia and NAOR, Moni. Pricing via processing or combatting junk mail. In: *Annual International Cryptology Conference*. Springer, 1992,139–147.

[13] JAKOBSSON, Markus and JUELS, Ari. Proofs of work and bread pudding protocols. *Secure Information Networks*. Springer. 1999, 258–272. 1540-7993, DOI: 10.1007/978-0-387-35568-9_18

[14] TOMÁNEK, Jan. *FITCOIN: transakce*. Prague, 2018. Bachelor's Thesis. Czech Technical University.

[15] BRAFMAN Ori. *The Starfish and the Spider: The Unstoppable Power of Leaderless Organizations*. Penguin Putnam Inc, 2006. ISBN10: 1591841836

[16] ROSENFELD, Meni. *Analysis of hashrate-based double-spending*. [online]. 2019 [visited on 2019-3-21]. Available from: `https://arxiv.org/pdf/1402.2009.pdf`

[17] BALIGA, Arati. *Understanding blockchain consensus models.* [online]. 2017 [visited on 2019-3-13]. Available from: `pdfs.semanticscholar.org/`

[18] TAPSCOTT, Don and Alex.*Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies Is Changing the World*, Portfolio Penguin, 2018. ISBN13: 9781101980149

[19] Anon. *Cryptography Digital signatures*. [online picture] [visited on 2019-3-15] Available from: `https://www.tutorialspoint.com/cryptography/cryptography_digital_signatures.htm`

Appendix A

Acronyms

CPU Central processing unit

BSD Berkeley Software Distribution

GNU Recursive acronym for "GNU's Not Unix!"

IP Internet Protocol

UTC Coordinated Universal Time

CTU Czech Technical University

SHA Secure Hash Algorithm

SPV Simplified Payment Verification

GCC the GNU Compiler Collection

Appendix B

Contents of enclosed USB drive

- `readme.txt` ..... brief description of the contents of the USB drive
- `A_Miner_for_Fitcoin.pdf` ..... the thesis
- `ftc` ..... the directory of Fitcoin source codes
- `wallet` ..... contains the private key to the address in the genesis block
- `latex` ..... the directory of LaTeX source codes of the thesis