



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Hector Joseph Smith

Technical analysis of the Bitcoin
cryptocurrency

Hector Joseph Smith

Technical analysis of the Bitcoin cryptocurrency

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang Bachelor European Computer Science
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Dr. Klaus-Peter Kossakowski
Zweitgutachter : Prof. Dr. Philipp Jenke

Abgegeben am 04.12.2015

Hector Joseph Smith

Thema der Bachelorarbeit

Technische Analyse der Bitcoin Kryptowährung

Stichworte

Bitcoin, Kryptowährung

Kurzzusammenfassung

Das Bitcoin-Protokoll hat sich zum beliebtesten Zahlungssystem weltweit entwickelt. Um den Anwendern zu helfen, dieses System besser zu verstehen, stellt diese Arbeit eine detaillierte Beschreibung dar, wie dieses Protokoll entwickelt wurde und wie es funktioniert. Dies schließt neben historischen Informationen, anhand welcher Bitcoin entwickelt wurde, einer Einführung in einige mathematische Operationen und Datenstrukturen, worauf Bitcoin basiert, ein. Anschließend wird eine Beschreibung der verschiedenen Komponenten des Protokolls, als auch eine technische Analyse, in Bezug auf technische und soziale Fragen im Rahmen der Entwicklung und Nutzung von Bitcoin dargestellt. Diese Analyse enthält auch einige mögliche Angriffe gegen Bitcoin, sowie sich diese auch auf Benutzer_Innen auswirken. Es wird festgestellt, dass die behandelten Themen keine ernstzunehmenden Bedrohungen für Bitcoin sind, welche aufgrund der Maßnahmen entwickelt wurden um die Effektivität zu verringern.

Hector Joseph Smith

Title of the paper

Technical analysis of the Bitcoin cryptocurrency

Keywords

Bitcoin, Cryptocurrency

Abstract

The Bitcoin protocol has become the most popular cryptocurrency currently in use. In order to help it's users better understand how Bitcoin operates, this thesis provides a detailed description of how this protocol was designed and how it functions. This includes historical information related to how Bitcoin was developed, an introduction to some of the mathematical operations and data structures it is based on, a description of the various components of the protocol, and finally, a technical analysis is made on some of the technical and social issues raised by the development and use of Bitcoin. During the analysis some possible attacks against Bitcoin, and how each one affects users, are discussed. It is determined that the discussed issues are not serious threats against Bitcoin due to the measures in place designed to reduce their effectiveness.

Table of Contents

Table of Contents.....	4
Glossary.....	6
1 Introduction.....	7
1.1 Goal of the thesis.....	8
1.2 Target audience.....	8
1.3 Structure of the thesis.....	9
1.4 Thesis delimitations.....	10
2 Origins of Cryptocurrencies.....	11
2.1 What is a cryptocurrency?.....	11
2.2 Current cryptocurrencies.....	14
2.2.1 Bitcoin.....	15
2.2.2 Other implementations.....	16
2.3 The history of Bitcoin.....	20
2.4 What makes Bitcoin so Important.....	25
2.5 Related Research.....	26
3 Protocol Design.....	27
3.1 Underlying mathematics.....	27
3.1.1 Hashing algorithms.....	28
3.1.2 Public key cryptography.....	32
3.1.3 Proof of Work.....	34
3.1.4 Merkle Trees.....	36
3.2 Design concepts.....	38
3.2.1 Transactions.....	38
3.2.1.1. Regular Transactions.....	38

3.2.1.2. Coinbase Transactions.....	41
3.2.1.3. Restrictions.....	42
3.2.2 Script.....	43
3.2.2.1. Script Templates.....	44
3.2.2.2. Hash Types.....	49
3.2.2.3. Signature Creation.....	52
3.2.3 Addresses.....	53
3.2.4 Blocks.....	55
3.2.5 Mining.....	60
3.3 The Bitcoin network.....	65
3.3.1 Block chain.....	65
3.3.2 Network communication.....	68
3.3.3 Bitcoin mining pools.....	70
4 Analysis.....	72
4.1 Technical Analysis.....	72
4.1.1 Address anonymity.....	73
4.1.2 Double spending.....	78
4.1.3 Botnet farming.....	81
4.1.4 Flooding.....	83
4.1.5 Software Errors.....	85
4.1.6 Comparison.....	86
4.2 Social Analysis.....	89
4.2.1 Bitcoin Economics.....	89
4.2.2 Bitcoin Legality.....	91
5 Conclusion.....	93
5.1 Was the goal reached?.....	94
5.2 Proposed improvements to Bitcoin.....	95
5.3 Future work.....	96
Table List.....	97
List of Figures.....	98
References.....	99

Glossary

TERM	MEANING
BTC	Bitcoin currency unit. Serves the same purpose as “€” or “\$”.
Satoshi	Smallest fraction of a Bitcoin which can be sent with the current protocol implementation. $1 \text{ satoshi} = 1.0 \times 10^{-8} \text{ BTC}$
Tx	Abbreviation for transaction.
ASIC	Application-Specific Integrated Circuit. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit
Fiat Currency	Term used to categorize 'traditional' currencies, such as US Dollar, Euro, etc.

Table 1: Glossary Items

1 Introduction

I believe the topic of this thesis is of great importance in the modern age. As advancements in computer science are made, new complex technologies are developed and implemented faster than ever. One of these advancements was the recent introduction of cryptocurrency systems. These new currency systems have since become increasingly popular and revolutionized the way individuals can conduct business in an online environment.

Cryptocurrencies such as Bitcoin are unique as they provide a distributed, secure and anonymous currency which does not require a central financial institution to process transactions as is the case with traditional currencies. And I believe it is important for the public to understand how these systems function, thus providing them the power to make informed decisions on whether or not to use them and even have the possibility to propose improvements and new features.

1.1 Goal of the thesis

The main goal of this thesis is to provide a detailed technical description of what Bitcoin actually is and how it was designed to function, as well as an analysis of the security and privacy granted to its users.

More specifically, my goal is to provide the reader with a resource which will allow an understanding of what Bitcoin and all of its various components are and how they all function together in order to create a system which is stable and secure enough to be used as a currency. This will include a broader overview of the complete system as well as an in-depth technical analysis of each component and how they work together.

I will also aim to provide the reader with historical information about some socio-economic aspects of Bitcoin such as how it generates value and how it can be exchanged for goods and services, as well as some of the social and legal impacts its introduction and widespread use has caused.

1.2 Target audience

This thesis has been targeted at readers who have a general interest or knowledge in computer systems and wish to learn more about how the popular cryptocurrency Bitcoin operates on a more technical level. This does not include readers interested in the social and economic impacts of the use of Bitcoin. These topics will not be discussed in any detail during this thesis.

As the goal of this thesis is mainly technical, the reader is expected to hold some knowledge regarding computer systems, security protocols and network communication. However, a brief overview of some of the most important topics, such as hashing algorithms and public key cryptography, will be provided.

1.3 Structure of the thesis

The structure of this thesis is divided into five main chapters,

1. Introduction
2. Origins of Cryptocurrencies
3. Protocol Design
4. Analysis
5. Conclusion

Each chapter will contain two or more subsections which will discuss topics related to the main chapter separately. Some sub-chapters will also require further segregation by creating a third level with more sections.

The first chapter includes topics related to the introduction of the thesis, not the topic. Topics such as the goal of the thesis and the target audience will be included here.

The second chapter, 'Origins of Cryptocurrencies' will introduce the concept of a cryptocurrency, provide some information on the various alternative implementations as well as some historical information related to Bitcoin.

Chapter three is the main focus of the thesis, and as such, the longest. This chapter will include topics related to how the protocol was designed and how it functions.

Chapter number four is dedicated to the analysis portion of the thesis and is divided into two sub-chapters. An analysis of some security issues and theoretical attacks against the protocol will be included in the first sub-chapter, with the second including a very brief social analysis, where some of the economical and legal aspects of Bitcoin will be exposed.

The final chapter is devoted to the conclusion of the thesis. This will include a review of what was discussed and an analysis on whether or not the goal was reached.

As well as these five chapters, the thesis includes some unnumbered sections devoted to the table of contents, glossary, table and figure lists and the references.

1.4 Thesis delimitations

As previously mentioned, legal, economic and social aspects of the Bitcoin protocol will not be discussed in this thesis. While these topics are important for the full understanding of the Bitcoin protocol and how it has come to affect our society, they do not fit with the skill-set and knowledge of the author and the technical focus of the thesis.

Other topics which will not be discussed include components of the protocol such as contracts, wallets and alternate operating modes. While these topics are important to the complete understanding of the Bitcoin protocol, as the author has limited resources it was decided to remove these topics in favour of increased focus on other aspects of the protocol which are fundamental to its operation.

2 Origins of Cryptocurrencies

The goal of this chapter is to provide the reader with a introduction to the concept of a cryptocurrency as well as some historical information related to how cryptocurrencies first appeared.

This chapter will also touch on the reasons why this thesis is dedicated to discussing the Bitcoin cryptocurrency instead of one of the many other variants and some related research the development of Bitcoin has launched.

2.1 What is a cryptocurrency?

With the incredible speed of developments in computer science, more and more new technologies have emerged along with new concepts and new terms. One such development is the concept of a “cryptocurrency”.

Cryptocurrencies such as Bitcoin represent a sub-set of a more broad term, 'digital currencies'. To clarify the differences between these two terms, a definition for each is provided:

Digital currency:

“Digital currency or digital money is an Internet-based medium of exchange distinct from physical (such as banknotes and coins) that exhibits properties similar to physical currencies, but allows for instantaneous transactions and borderless transfer-of-ownership.”

[Wiki:DigitalCurrency]

Cryptocurrency:

"A cryptocurrency (or crypto currency) is a medium of exchange using cryptography to secure the transactions and to control the creation of new units."

[Wiki:Cryptocurrency]

The main difference between these definitions is "... and to control the creation of new units". This means that as well as using cryptographic algorithms to secure transactions, similar algorithms are also employed in the process of generating new currency units. This is possible because a cryptocurrency is a complete, independent currency system which is not based on other currencies and is not owned or controlled by any governmental or banking entity. Therefore it is free to create new currency units without conforming to the same regulations as banks.

A more complete definition of a cryptocurrency is this:

A cryptocurrency is an Internet-based medium of exchange which uses cryptography to secure transactions and control the creation of new units. It is distinct from physical currency (such as banknotes and coins) but exhibits properties similar to physical currencies, allowing for instantaneous transactions and borderless transfer-of-ownership.

Another similar term which may appear while researching this topic is "virtual currency". This is another sub-set of digital currencies, but contrary to cryptocurrencies, it is only used in virtual environments and can be created or destroyed by the administrator of that specific virtual environment. Usually it is also unique to the virtual environment where it was created, and can only be legally exchanged for other currencies if the administrator allows it. These types of currencies are often used in online games or online casinos.

While Bitcoin and other cryptocurrencies are completely independent from other currencies, services such as Paypal do not have a unique currency, and are instead based on fiat currencies. This type of services are better defined as online payment services and are separate from cryptocurrencies. While it is possible to purchase BTCs in exchange for fiat currency, this does not increase the number of currency units available, it only redistributes them among its various users. Bitcoin design does not allow any entity to manually create new currency units without doing the required computational work. Which in the case of Bitcoin is solving a proof-of-work.

A common question people may have when they first hear about the concept of a cryptocurrency is: *"Is it really a currency?"* This question is often raised because cryptocurrencies appear to not have any intrinsic value whatsoever. Currency units are generated when a computer expends a certain amount of work to solve an arbitrary problem, and after these currency units are generated, they do not possess any physical representation. This is distinct to fiat currencies, which can be represented by physical bank notes and coins, or can be exchanged for precious metals such as diamonds or gold. This allows any client to exchange their bank notes for a more 'real' representation of wealth.

The reason why cryptocurrencies such as the popular Bitcoin can be categorized as currencies lies in the fact that they possess three important characteristics: They are hard to earn, limited in supply and easy to verify. [Economist, 2015]

A very simplified overview of how the Bitcoin protocol operates is shown here:

1. Computers verify previous transactions and attempt to solve complex mathematical problems in order to earn currency units as a reward.
2. The owner of the computer which solved the current puzzle is the recipient of a special transaction where new value is created.
3. Users can transfer money to and from each other using their public addresses. Usually each transaction will have a small fee associated with it.
4. Users can exchange currency units for fiat currency at currency exchange services.
5. Businesses can offer a public address where customers can deposit currency units in exchange for goods and services.

2.2 Current cryptocurrencies

The story of digital online cash schemes, and later cryptocurrencies, begins with David Lee Chaum and his 1982 paper which first discussed the possibility of using cryptographic protocols to allow users to obtain and spend currency anonymously. [Wiki:Chaum] A year later, in 1983, he published another paper proposing a new protocol for digital cash. This new protocol was named 'eCash' and was later realised when the DigiCash company was founded in 1990 by David Chaum.

While David Chaum had already implemented services similar to a cryptocurrency, these services are better categorized as 'Digital Currencies'. The first concept of an actual cryptocurrency appeared in 1998 when Wei-Dai published a description of "b-money". Unlike previous proposals, b-money can be summarized as an anonymous, distributed electronic cash system. [Wei-Dai, 1998] The goal being to create a secure and private way to transfer money from one person to another without relying on government or private companies.

Shortly after Wei-Dai's article another concept of cryptocurrency was presented by Nick Szabo in 1998 named "Bit-Gold" which introduced the idea of users requiring to complete a *proof-of-work* while using the system. [Szabo, 2008] Neither of these implementations received significant support, but are considered precursors to the Bitcoin system. [Wiki:Szabo]

In 2008 an anonymous entity known only as "Satoshi Nakamoto" released a white paper describing a new cryptocurrency named Bitcoin. This was the first true decentralized cryptocurrency yet proposed and soon gained much more support than any other previous proposals. [Nakamoto, 2008]

Satoshi's paper describing Bitcoin initiated what could be described as a cryptocurrency gold rush of sorts. Since then, hundreds of new cryptocurrency concepts and implementations have been presented. This was helped, in part, by the open source nature of the Bitcoin system. However, most did not add any real technological innovation and remained very similar to the original description, only altering minor details such as which cryptographic algorithms are used or the value generation rate.

In more recent history, a so called 2nd Generation of cryptocurrencies began in 2014, with the development of new cryptocurrencies with new functionality to achieve increased user privacy, and better usability. Examples of such implementations are Monero, Ethereum or Nxt. While some of these new developments are very important for the future of cryptocurrencies, the main focus of this thesis is Bitcoin, and as such these other implementations will not be discussed in detail.

2.2.1 Bitcoin

The history of Bitcoin itself began with Satoshi Nakamoto's paper in 2008 and became the first decentralized cryptocurrency. And while other protocols and services, such as b-money and DigiCash, were already in existence, Bitcoin attempted to be unique by creating a true decentralized cryptocurrency without any ties to any form of fiat currency or material wealth. This helped make Bitcoin unlike any previous implementation, as the digital currencies available at the time were either based on fiat currencies or were not completely decentralized.

The main reasons which allow Bitcoin to be categorized as a decentralized cryptocurrency is the fact that, unlike the previous implementations, Bitcoin does not have any central servers or administrators of any kind. Instead it relies on every user to agree on a pre-determined set of rules in order to verify the validity of a public transaction ledger. When new transactions are made they are validated by the network of users and, if the transaction is valid, it is added to the transaction ledger.

2.2.2 Other implementations

Since Bitcoin was first implemented, hundreds of alternative implementations of the cryptocurrency concept have been presented over the last few years. At the time of writing this thesis over 600 different cryptocurrencies are available for trade. [Wiki:ListCryptocurrency]

In part, the cause of the vast number of cryptocurrencies available today was the open source nature of the Bitcoin protocol and software. This allows anyone with some understanding of programming to copy, modify and release a new version of the Bitcoin software to create their own new currency. However, similarly to *fiat* currency systems, a cryptocurrency requires a large amount of users who trust the system in order to be successful. This did not happen for the vast majority of the numerous other available cryptocurrencies, leading to most of these having very limited adoption.

The list of available implementations include some which were designed specifically to appeal to a certain user base or to provide an alternative to Bitcoin with small adjustments without necessarily adding anything new to the original implementation. This category include cryptocurrencies such as:

Litecoin

Litecoin is currently one of the largest cryptocurrencies available and is nearly identical to the Bitcoin protocol. It was first made available on October 7th 2011 by Charles Lee, a former Google employee. [Wiki:Litecoin]

There are three main differences between Litecoin and Bitcoin;

- While the Bitcoin protocol aims to generate a block every 10 minutes, the value is reduced to 2.5 minutes in Litecoin. The goal being a faster transaction confirmation time.
- While Bitcoin uses a hashcash proof-of-work scheme, based on the hashing algorithm SHA-256, Litecoin uses scrypt in an attempt to reduce the effectiveness of GPU and ASIC miners.
- The total number of currency units produced is approximately 4 times larger, with Litecoin being capped at 84 million coins, instead of Bitcoin's 25 million.

DogeCoin

DogeCoin is a Bitcoin derivative named after a popular internet meme. It first appeared in December of 2013 when Jackson Palmer released it as a “joke currency”. However, DogeCoin soon gained a large support base and its market capitalization reached \$60 million by January 2014. [Wiki:DogeCoin]

DogeCoin functions in a similar manner as the Bitcoin currency, however, while Bitcoin is a deflationary currency, with less and less coins being mined over time, DogeCoin does not have any limit on the number of coins which can be mined. At the time of writing, over 100 billion coins are in circulation and even more are being generated at a rate of approximately 5 billion a year.

PotCoin

The protocol implemented by PotCoin is quite similar to that of Litecoin, but was developed with the aim of being the standard form of payment for the legalized cannabis industry.

It was first release on January 21, 2014 through GitHub and within a month was added to a cryptocurrency exchange where it was possible to exchange Bitcoins for PotCoins. [Wiki:PotCoin]

Coinye

Coinye (formerly known as Coinye West) is a cryptocurrency based on Bitcoin and named after the popular American hip hop artist Kanye West. It was released on January 7th, 2014 but soon became defunct due to a trademark infringement lawsuit by Kanye West's layers.

While the peer-to-peer network which supports the currency is still operational, the block generation difficulty decreased dramatically between January and May 2014, indicating that the computational power available on the network decreased by roughly 99%. [Wiki:Coinye]

Other implementations have attempted to add new functionality to the original concept.

Peercoin

Peercoin was inspired by Bitcoin and shares a large portion of its technical description and source code, but it is the first cryptocurrency to use both a proof-of-work scheme combined with a proof-of-stake algorithm when generating currency.

In simple terms, the proof-of-stake scheme rewards users based on the number of coins they possess, thus reducing the risk of a 51% attack by requiring an attacker to control a large amount of currency as well as computing power.

This implementation attempts to solve a potential security issue which may arise in Bitcoin as the block mining reward decreases. As fewer machines attempt to mine blocks, the amount of combined computing power on the network decreases, leading to a decrease in cost for an attacker to gain control of 51% of the network. While this cost will likely remain very high, this would theoretically allow that attacker to reverse his payments by using a double spending attack.

Currently, peercoin is one of the largest cryptocurrencies available and can be exchanged for *fiat* currencies as well as some other cryptocurrencies such as Bitcoin. [Wiki:PeerCoin]

Dash

Dash (formerly known as DarkCoin) is another cryptocurrency based on Bitcoin which was released on January 18th 2014 and later renamed on March 25th, 2015.

Unlike Bitcoin or some of the other cryptocurrency implementations, Dash uses a hashing scheme known as X11 instead of the SHA and RIPEMD algorithms used by Bitcoin. The X11 approach hashes the input data using 11 different hashing algorithms. The result of this is an increased performance from CPU's when mining when compared with GPU's. [Wiki:Dash]

Another innovation brought by Dash is the use of a decentralized coin mixing algorithm known as Darksend. In the current implementation, transactions from various users are combined into one single transaction with several outputs, thus greatly increasing the difficulty for directly tracing the transactions and the flow of funds. This service is also provided by third parties for Bitcoin users, but it is not the default behaviour.

Bytecoin (BCN)

The Bytecoin cryptocurrency is the first cryptocurrency to implement the CryptoNote protocol. This protocol has since been implemented in other cryptocurrencies such as Monero and DigitalNote.

The advantage in using CryptoNote lies in the fact that transactions remain more private. While it still uses a public block chain similar to Bitcoin, the flow of funds cannot be traced back in the same way it can with the Bitcoin blockchain. With CryptoNote only the sender and recipient of a transaction hold all the data relevant to the transaction.

Another significant change relative to Bitcoin is the way in which the proof-of-work is calculated. While Bitcoin uses the SHA-256 hashing algorithm, cryptocurrencies based on CryptoNote use a memory bound function known as CryptoNight which cannot be easily pipelined and implemented in ASIC devices. [Wiki:CryptoNote]

2.3 The history of Bitcoin

Table 2 contains several of the important events related to the Bitcoin cryptocurrency, including some which lead to the first development of the cryptocurrency.

DATE	EVENT
xx.xx.1982	First paper discussing a secure digital cash protocol was published by David Lee Chaum.
xx.xx.1983	David Chaum publishes a paper proposing the eCash protocol. Which would allow users to perform secure anonymous transactions.
xx.xx.1990	DigiCash founded by David Chaum implementing the eCash protocol.
26.11.1998	Wei-Dai presents the concept of b-money.
01.08.2002	Adam Black publishes "hashcash". Hashcash is the proof-of-work algorithm which is currently used in the Bitcoin system.
xx.xx.2007	Satoshi Nakamoto begins developing Bitcoin. The exact date is unknown.
18.08.2008	"bitcoin.org" domain name registered.
31.10.2008	Bitcoin design paper published by "Satoshi Nakamoto". At the time of writing, the real identity of Satoshi Nakamoto remains unknown and it is unclear whether it represents a single individual or a group of individuals.
03.11.2008	Bitcoin project registered at "SourceForge.net".
03.01.2009	Genesis Block created at 18:15:05 GMT. The genesis block is the first block in the block chain

DATE	EVENT
	on top of which all other blocks are built. It is unique because, unlike every other block in the block chain, it does not reference the previous block. [BtcWiki:Genesis]
09.01.2009	Bitcoin client v0.1 released.
12.01.2009	First Bitcoin transaction, in block 170. From “Satoshi” to “Hal Finney”.
05.10.2009	Bitcoin exchange rate published by New Liberty Standard. \$1 = 1309.03 BTC
16.12.2009	Bitcoin client v0.2 released.
30.12.2009	First increase in difficulty. Difficulty is a calculated value which represents the average amount of effort required to solve the current proof-of-work problem.
06.02.2010	Bitcoin market established. Bitcoin market is a currency exchange service where users can purchase and sell BTCs in exchange for fiat currency.
22.05.2010	The user “laszlo” becomes the first user to buy pizza with Bitcoins, paying 10,000 BTC for approximately \$25 worth of pizza.
07.07.2010	Bitcoin client v0.3 released.
17.07.2010	MtGox established. MtGox is a Bitcoin exchange where BTCs could be traded for various real world currencies such as US Dollars or Euros. It later became the most widely used Bitcoin exchange service.
01.10.2010	First public OpenCL miner software released.

DATE	EVENT
	OpenCL software uses the GPU cores to mine BTCs instead of the traditional CPU cores. The inherent higher parallelism capability of the GPU allowed for a much higher mining speed.
06.10.2010	Bitcoin economy passed \$1 million as the exchange price reached \$0.50 / BTC.
09.12.2010	Difficulty passed 10,000.
16.12.2010	Bitcoin Pooled Mining found their first block. By mining Bitcoins in a pool, groups of individuals with weaker machines can cooperate in order to accelerate the mining process. Rewards are then split among the participants according to a pre-defined criteria.
28.01.2011	Block number 105000 created. At this time 5.25 million BTCs are currently in circulation, which, due to the reward reduction system, is approximately a quarter of the total number of BTCs to ever be available.
09.02.2011	Bitcoin value reached parity with the US Dolar. \$1 = 1 BTC
25.03.2011	10% difficulty reduction. Difficulty reductions are rare, as the amount of computational power available on the network rarely decreases. This represents the largest reduction in difficulty to date.
23.04.2011	Bitcoin value reached parity with the Euro and the GBP. The value of the Bitcoin currency passes \$10 million.
30.04.2011	Difficulty passes 100.000.
08.06.2011	Bitcoin exchange rate reaches an all time high of

DATE	EVENT
	\$31.91=1 BTC.
19.06.2011	<p>MtGox database compromised, resulting in user details of 60,000 users being leaked.</p> <p>On the same day, an unidentified individual gained access to an administrator account at MtGox and issues sell orders on thousands of Bitcoins. Reducing the exchange value of Bitcoins at MtGox from \$17.51 to \$0.01.</p> <p>Trading was closed for 7 days at MtGox and the invalid sell orders were reversed. However, BTCs were stolen from some MtGox users due to the database breach.</p>
24.06.2011	Block generation difficulty reaches 1,000,000.
23.08.2011	<p>P2Pool mines it's first Bitcoin block.</p> <p>P2Pool is the first Bitcoin mining pool based on peer-to-peer technology.</p>
01.03.2012	Largest ever Bitcoin theft due to a security breach at the web host Linode, resulting in approximately 50,000 BTC being stolen.
28.10.2012	<p>Halving day. The first date where the amount of BTCs rewarded for completing a block is halved. BTC reward dropped from 50 BTC to 25 BTC.</p> <p>In order to avoid super inflation, Bitcoin will continue to reduce the awarded BTCs to miners by 50% every four years.</p>
28.03.2013	Total Bitcoin market worth passes \$1 billion.
01.04.2013	Bitcoin value reaches \$100 / BTC.
29.10.2013	The first Bitcoin ATM is installed in Vancouver, Canada. [Wired, 2013]
23.11.2013	Bitcoin market worth passes \$10 billion for the first

DATE	EVENT
	time.
28.02.2014	MtGox files for bankruptcy. Reports suggest that the bitcoin exchange closed down and filed for bankruptcy after 744.000 BTCs (valued at \$350 million) were stolen. [BBC, 2014]
06.01.2015	Bitstamp suspends service after a security breach resulted in 19.000 BTCs being stolen (valued at \$5 million). [Reuters, 06.01.2015] Bitstamp is currently one of the largest Bitcoin exchange services, where users can exchange BTCs for USD or Euros Bitstamp later reopened with increased security and claimed users would not lose money due to the breach. [Reuters, 09.01.2015]

Table 2: Bitcoin History [BtcWiki:History] [Wiki:Bitcoin]

2.4 What makes Bitcoin so Important

The reasons why Bitcoin is so often regarded as an important development in the area of cryptocurrencies comes down to two important aspects. Firstly, the technical innovation it brought, and secondly, the economic impact it had as it's user base rapidly surpassed any other previous attempt at a digital currency.

Technical innovation

By being the first decentralized cryptocurrency, Bitcoin brought the use of new technologies to the realm of digital currencies and cryptocurrencies. Leading the way for other protocols to be built based on Bitcoin.

Bitcoin has been at the source of the implementation of over 740 other cryptocurrencies. Most are very similar to Bitcoin and did not bring any new functionality of note, but a small number of them, such as Monero or NXT, have attempted to implement new technologies in their protocol in order to provide new functionality and better privacy to their users. While still following the same base concepts as Bitcoin.

As well as this, the open source nature of the Bitcoin project allows users to view and modify the source code behind the currency, allowing the Bitcoin protocol to evolve and improve over time based on proposals and developments made by the community. This allows Bitcoin to react to changes and new security threats as they are discovered.

Economic impact

As the number of users using the Bitcoin protocol rapidly increased, so did the value of the currency. Since its inception Bitcoin has consistently been the most widely used cryptocurrency available. As of November 2015 Bitcoin has a market capitalization of around 4.3 billion Euros, with daily transactions involving over 100 million Euros. [CoinMarket, 2015]

Other competing cryptocurrencies account for a much smaller share of the total cryptocurrency market, with the second biggest cryptocurrency having a market capitalization of only around 130 million Euros, approximately 3% that of Bitcoin.

2.5 Related Research

As well as being at the source of the development of a large number of alternate cryptocurrencies, Bitcoin has also introduced the concept of using a public blockchain as a way to store data in a distributed network. This concept has since been expanded beyond being solely used in the context of a currency.

Some believe that blockchains will be, or should be, at the core of a re-design of the internet, with email services, DNS servers and TLS/SSL certificate authorities being based on this concept instead of central servers.

“Suppose you replaced the Internet’s centralized Domain Name System with a blockchain for Internet names (like Namecoin) such that every DNS request included some proof-of-work effort. Or you used any blockchain (including Bitcoin’s) as a notary service. Or you built a new blockchain for crowdfunding. Or you replaced a centralized system which absolutely does need to be scrapped — that horrific barrel of worms known as TLS/SSL Certificate Authorities — with a blockchain-based solution powered at the browser level. ... Or you built a new distributed email service, with a blockchain for email addresses, and every time you checked your email you contributed to the network. Or a new distributed social network, with a blockchain verifying identities, powered by code that ran every time its users launched its app or visited its web page.”

[Evans, 2014]

Services such as Namecoin are already attempting to change the structure of the Internet by implementing a distributed DNS service with a similar structure to that of Bitcoin. To accomplish this, users can include data in transactions which allows them to register domain names in exchange for currency units (NMCs in this case). This helps prevent censorship as registered domains cannot be removed from the DNS system as easily as they can when the system is controlled by a single entity. [Wiki:Namecoin]

At the time of writing Namecoin users are limited to registering `.bit` domains, but it would be possible to extend the system to allow registering any type of domain.

Another use of the blockchain concept is being explored by several companies with ties to the music industry. Companies such as Peertrack and Ujo are attempting to change the way artists receive royalties for their work by creating a public blockchain where users can purchase music directly from the artist. [woolci, 2015]

3 Protocol Design

The goal of this section is to provide the reader with a technical description of the several components of the Bitcoin protocol and how they function.

This section is divided in to three main sections, with the first one being dedicated to explaining some of the base mathematical operations and data structures used by the Bitcoin protocol. In the second section some of the major design concepts of the protocol are shown and explained. The final section introduces the topic of how the Bitcoin network is structured and how some network operations are implemented.

3.1 Underlying mathematics

The Bitcoin protocol makes use of several lower level mathematical operations and data structures the reader should have some knowledge about before attempting to understand the actual protocol.

Some of the operations described here are widely used in other protocols of various types as well as Bitcoin. But as these topics fall outside the main focus of this thesis, a complete description will not be provided here. However, these topics have been the focus of various other technical papers which the reader can explore.

3.1.1 Hashing algorithms

A hash function can be defined as method to map an arbitrary input to a digital output of fixed size. [Wiki:HashFunction] This can be visualized in a simple example:

$$\text{hash}(\text{John}) = 70 \text{ F3}$$
$$\text{hash}(\text{Lisa}) = 15 \text{ A2}$$
$$\text{hash}(\text{James}) = 34 \text{ CD}$$
$$\text{hash}(\text{abc123}) = \text{B7 C2}$$

Here, four input values (also known as keys) are 'digested' and a four digit hexadecimal output value is returned. As hashing algorithms are deterministic, the same input will always provide the same output. In this example, whenever the key 'John' is digested, the value '70F3' is returned.

Hashing algorithms are widely used today in computer science for a variety of purposes, as each purpose has varying requirements, a very large number of hashing algorithms have been implemented to handle different use cases and data types efficiently. It is also important to note that, in general, the mathematical algorithm behind these systems is widely known, as in several cases multiple users and systems need access to the exact same algorithm.

Two common applications for hashing algorithms are given here:

Checksums

In order to ensure data has not been corrupted during certain operations, such as data transfer, a method is needed to ensure that every byte received is exactly the same as the bytes which were sent. In some cases, a single flipped bit can corrupt an entire file, and there isn't always an easy method to determine which bit was altered.

To solve this problem, checksum algorithms are used. In short, these are hashing algorithms used to hash a certain amount of data, such as a file, before and after an operation and verify that the calculated value remains the same.

$$\text{checksum} = \text{hash}(\text{data})$$

Password storage

In numerous computer related services users are required to supply authentication credentials in the form of a user-name and password. This information needs to be stored somewhere to allow the server to determine if the data a certain user has entered is valid or not. The naive approach would be to simply store the user-name and password in a database and run a simple comparison when a login request was received.

This approach has numerous issues. To solve them, cryptographic hashing algorithms can be used to achieve the same results, but with a much higher degree of security.

There are several ways to implement such a system, but a common approach involves storing three values in the database, 1) the user-name for the user. 2) A random field of a certain length, designated 'salt'. 3) The hash value obtained when the users password is concatenated with the salt value and hashed.

$$data = hash(password || salt)$$

When a client machine sends a user-name and password in an attempt to login, the server will retrieve the salt value stored for that user-name and re-calculate the hash value using the provided password. If the calculated hash matches the hash value stored in the database, the login attempt is deemed successful, otherwise the provided user-name or the provided password is incorrect.

$$if (hash(password_{user} || salt_{db}) == hash_{db}) \{ login_success \}$$

For the scope of this thesis, the type of hashing algorithms discussed in this thesis fall under the category of 'Cryptographic Hashing Algorithms'. These are a sub-category of hashing algorithms which also have the following properties: [Wiki:CryptoHash]

One way / Non reversible

This means that the operation cannot be reversed. Hashing algorithms with this property prevent an attacker from easily recover the original input data based solely on the algorithm definition and the output hash data. In an ideal algorithm, the only way to determine the input value which generated a provided hash value would be to a 'brute-force' approach where every possible input value would be hashed and compared to the provided value. The computational requirements for this form of attack increase exponentially as the number of possible output values increases.

Collision resistance

Collisions occur when distinct input values generate an equal output value.

$$\text{hash}(\text{abcde})=71 \text{ BA}$$

$$\text{hash}(12345)=71 \text{ BA}$$

Given the fact that hashing algorithms can process virtually infinite amounts of data and will always return a fixed length output, collisions are unavoidable. But in several applications it is vital to reduce the probability of collisions as much as possible. This is often done by increasing the output length, thus increasing the number of possible hashes.

This will help ensure that it is virtually impossible to manufacture distinct input values which generate equal hash values. For example, a hashing algorithm with an output length of 256 bits will have $2^{256}=1.157 \times 10^{77}$ possible unique output values.

Non-continuous

In some applications of hashing algorithms it is required for small changes in the input value to also cause small changes in the output value. This is useful when searching for similar data, for example when using hashes to search for similar images. In these cases it is important for the algorithm to be as continuous as possible. However, cryptographic hashing algorithms must not be continuous, as this would decrease the complexity of generating distinct keys with equal hash values. In these algorithms, the smallest change will cause a drastically different hash value. This has been named the avalanche effect.

$$\text{hash}(\text{abcd})=\text{E1 FF}$$

$$\text{hash}(\text{abcb})=\text{01 5A}$$

At the time of writing the standard cryptographic hashing algorithm is the SHA-2 family published by the National Institute for Standards and Technology in the United States of America. The family includes several algorithms with various output lengths, including 128, 256 and 512 bits. [Wiki:SHA]

For the purposes of this thesis, another hashing algorithm family is also relevant, RIPEMD (RACE Integrity Primitives Evaluation Message Digest). This algorithm was first developed by a group of researchers in Belgium in 1996, and while the original algorithm had questionable security,

development has continued and there are no known attacks against the current version of the algorithms. [Wiki:RIPEMD]

Like the SHA family, RIPEMD comes with a variety of algorithms with various lengths, including 128, 160 and 256 bits.

The Bitcoin protocol includes algorithms from both these families, making use of the SHA-256 and the RIPEMD-160 algorithms in different circumstances.

In order to help prevent a single point of failure, some systems, including Bitcoin, will chain hashing algorithms together so that the output value from one algorithm will be used as the input for the next algorithm. Like so: $output = hash_B(hash_A(input))$

When this system is implemented with two different algorithms, it prevents the generated hashes from being vulnerable if a method to reverse or otherwise manipulate one of the algorithms is discovered. As both algorithms would need to be broken to allow abuse.

This process can also be employed using the same algorithm. In some circumstances the Bitcoin protocol uses this process with two rounds of SHA-256. This specific implementation is sometimes referred to as double-SHA or SHA².

3.1.2 Public key cryptography

Public key cryptography is widely used in numerous computer systems today in an attempt to ensure data confidentiality and integrity. Unlike symmetric key cryptography, where the same key is used to encrypt and decrypt the data, public key encryption algorithms generate a pair of mathematically linked keys, where one can decrypt what the other encrypts.

These keys are generally known as public and private keys, where the public key is usually only used to encrypt data and the corresponding private key is used to decrypt it.

As their names suggest, the public key can be freely made available to the public, while the private key should be kept secret and never shared with any other parties.

For a public key cryptography scheme to be successful, the following properties are required: [Jayanthi, 2015]

- It should be infeasible to calculate one key based on the other. For example, it should not be possible to calculate the private key using the public key.
- Both the private and public keys should be easy to generate.
- It should be easy to encrypt and decrypt data as long as the correct keys are used. For example, it should be easy for Alice to encrypt a message using Bob's public key, and it should also be easy for Bob to decrypt this message using his private key.
- It should be infeasible for an attacker to recover the original plain-text in spite of having access to the cypher-text and the public key used to encrypt the data.

The reason why these properties are possible comes down to the mathematics used to generate the keys and used to perform the encryption and decryption. Several methods have been proposed one of most well known is known as RSA (**R**on Rivest, Adi **S**hamir and Leonard **A**dleman), first published in 1977. [Wiki:RSA]

While different algorithms are based upon different mathematical problems, the goal of these algorithms is to present a problem which is easy to solve in one direction and extremely difficult to solve in the opposite direction.

Taking the RSA algorithm for a concrete example. The mathematical problem used by RSA is factorization, and assumes that it is very easy to multiply two given numbers together, but without any other information, it is

extremely difficult to, given a product, determine which two numbers were multiplied together to calculate the given value.

The actual implementation is naturally more complex, but it is based on this simple concept.

The properties achieved by algorithms such as RSA make public key encryption algorithms very well suited for securing communication between two individuals or machines and have been used for this purpose in countless situations and applications.

But aside from securing communication between two parties, the same concept can be used for other purposes, one of the most common among them being cryptographic signatures (or Digital Signatures).

Digital signatures can provide two assurances simultaneously. On one hand, they ensure that the signed data has not been altered since it was signed. This includes accidental corruption as well as intentional changes. On the other hand, signatures also ensure that the data was actually created by the person who claimed to have created it. In other words, it provides a method to prove the authenticity of the data.

The concept behind a cryptographic signature is relatively simple and is based upon public key cryptography algorithms. But instead of following the steps used to encrypt data for secure communication, the following steps are followed instead:

1. The first step is to calculate the hash value for the data to signed. A good cryptographic algorithm should be used to ensure the signature cannot easily be manipulated in such a way to allow the data to be altered without altering the hash value.

$$\text{hash value} = \text{hash}(\text{input})$$

2. Once the hash value is calculated, it is encrypted using the private key of the signer. This means that only the public key can decrypt the data.

$$\text{signature} = \text{encrypt}_{\text{private key}}(\text{hash value})$$

3. The final step is to append the encrypted data to the original data.

$$\text{final data} = (\text{data} \parallel \text{signature})$$

The recipient of the signed data only needs to separate the signature from the actual data, decrypt the signature using the readily available public key, and compare it with the hash value of the actual data. If the two values match, the data is considered authentic.

3.1.3 Proof of Work

The concept of proof-of-work problems was first introduced in 1993 by Cynthia Dwork and Moni Naor [Dwork et al., 1993] in order to create a method to reduce spam and denial of service attacks against servers. The presented concept would achieve this by requiring client machines to solve a complex problem and present the result before requesting any services from the server. The server would then verify the given result, and if valid, it proves the client performed the required work and the server can then process the client's request.

In order for this method to be useful, a certain amount of effort asymmetry is required, in other words, the task assigned to the client must be relatively hard to execute, but the verification process must be a much simpler task in order to avoid overloading the server during verification. To create a system with this characteristic, hash algorithms are often used.

The Bitcoin network uses a proof of work algorithm known as hashcash [Nakamoto, 2008] where the goal is to produce an input value which, when hashed, generates a value smaller than the pre-determined target value, usually designated by T . Due to the properties of hashing algorithms, the only way to solve this task is by continuously generating random input values, hashing them and verifying if the output fulfils the given requirements. Meaning that the process is highly probabilistic, similar to a lottery of sorts.

$$T=0500$$

$$\text{hash}(\text{random data})=7420$$

...

$$\text{hash}(\text{FNGiFUs})=0342$$

To verify if the presented solution is valid, other systems on the network only need to run one hash calculation on the presented data and verify if the output value is in fact smaller than the target.

$$\text{if}(\text{hash}(\text{data}) \leq T) \{ \text{valid} \}$$

As a simple example, assuming the hashing algorithm in use returns a decimal value with 4 digits and the target has been set to 500. A valid hash could be: 0342, while an invalid one could be: 7420. With this simple example there are 10.000 possible hash values, with 500 of them being valid. The same principle applies to the proof-of-work employed by the Bitcoin protocol, the numbers involved are simply much larger.

The general formula used to calculate the probability of a random SHA-256 hash being accepted, given a certain target value T , is as follows:

$$\text{Prob}[H \leq T] = \frac{T}{2^{256}}$$

Through simple mathematical operations, the previous formula can be transformed into one which returns the average number of attempts needed to achieve a valid hash.

$$\frac{1}{\text{Prob}[H \leq T]} = \frac{2^{256}}{T}$$

Using the previous example, the average number of attempts required to find a valid hash can be calculated: $\frac{10^4}{500} = 20$. If it assumed that each hash calculation takes 1 minute to compute, the current target value will allow for a valid hash to be found, on average, every 20 minutes. The exact same principle applies to the hashcash proof-of-work used in the Bitcoin protocol.

3.1.4 Merkle Trees

Merkle trees, also known as hash trees, were first developed by Ralph Merkle in 1979 and allow an efficient and secure method to verify the contents of large data structures.

Data in a merkle tree is organized as a binary tree so that each non-leaf node has two children. Each non-leaf node will then contain the hash value of the combined data of its two child nodes.

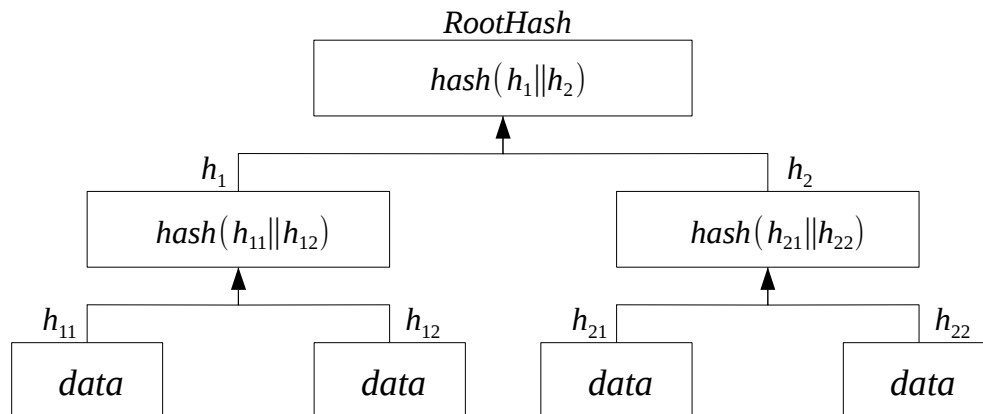


Figure 1: Merkle Tree Structure

In figure 1, all the leaf nodes ($h_{11}, h_{12}, h_{21}, h_{22}$) contain actual data, while the others only contain the hash values of the two nodes below them.

In the cases where a node only has one child, its data is duplicated in order for the algorithm to function correctly.

Merkle trees can be used for several purposes as they provide performance benefits in certain situations. One such situation is when trying to determine if a certain node is already included in the tree. The time complexity for searching a merkle tree is in the $O(\log n / \log t)$ [Berman et al., 2005], while other structures such as lists can have a complexity of $O(n)$.

This type of data structure is also of great benefit in situations where data integrity checks are required. Instead of calculating a hash value for a large file, and using that to determine if the file is damaged, merkle trees allow for a more efficient integrity check by splitting the file into blocks and building a tree based on those blocks. Each block can then be verified individually for alterations. If a block is found to have been modified, only that block will need to be replaced, instead of the entire file.

This structure allows for reliable file downloads from unreliable sources. As long as the root hash is obtained from a trusted source, the rest of the hash tree can be downloaded from any other source. Any alteration in one of the hashes will result in a distinct root hash. When the hash tree has been successfully downloaded, the data blocks can be downloaded and compared with the corresponding hashes.

This type of data structure is widely used in systems ranging from file systems (IPFS, ZFS), to peer-to-peer networks (Bittorrent) and version control software (Git) to ensure data integrity.

3.2 Design concepts

The goal of this sub-section is to describe how some of the most important concepts of the protocol are implemented and how these components function.

These topics include data structures such as transactions and blocks, the concept of Bitcoin mining as well as an introduction to how Bitcoins can be claimed and spent with the use of scripts.

3.2.1 Transactions

In the Bitcoin system there are two types of transactions, coinbase transactions and regular transactions. Both are based on the same architecture and work in the same way, but are used for different purposes. While regular transactions are used to transfer a certain amount of BTCs between two users, coinbase transactions are used to generate new currency and introduce it into the system.

3.2.1.1. Regular Transactions

Regular transactions are built upon previous transactions, creating a chain. Each transaction will use the outputs of previous transactions as inputs. Transactions in their raw form are only a collection of bytes, but can be represented by this structure:

FIELD		TYPE (SIZE)	DESCRIPTION
nVersion		int (4 bytes)	Transaction version. Currently 1.
#vim		VarInt (1-9 bytes)	Number of entries in the list of input transactions.
vin[]	hash	uint256 (32 bytes)	Double SHA-256 hash of the previous transaction. Used as the unique ID.
	n	uint (4 bytes)	Index of the desired transaction output within the transaction specified in the <i>hash</i> field.

FIELD		TYPE (SIZE)	DESCRIPTION
	scriptSigLen	VarInt (1-9 bytes)	Length (in bytes) of the <i>scriptSig</i> field.
	scriptSig	CScript (Variable)	Response to the challenge script provided in the previous transaction.
	nSequence	uint (4 bytes)	Transaction input sequence number.
#vout		VarInt (1-9 bytes)	Number of entries in the array of output transactions.
vout[]	nValue	int64_t (8 bytes)	Number of BTCs in transaction. Measured in 10^{-8} units.
	scriptPubkeyLen	VarInt (1-9 bytes)	Length of the <i>scriptPubkey</i> field.
	scriptPubkey	CScript (Variable)	Script specifying the conditions which need to be satisfied in order to allow spending the associated BTCs.
nLockTime		uint (4 bytes)	Timestamp past which the transaction can be placed in a block.

Table 3: Transaction data structure [Okupski, 2014]

Each transaction will include a list of inputs and outputs. The inputs of one transaction are simply outputs from previous transactions. Thus creating a chain of transactions where each one will depend on previous transactions. This makes it more and more infeasible for any past transaction to be modified as more and more transactions are stacked on to it, as all following transactions would also have to be modified to ensure they remained valid.

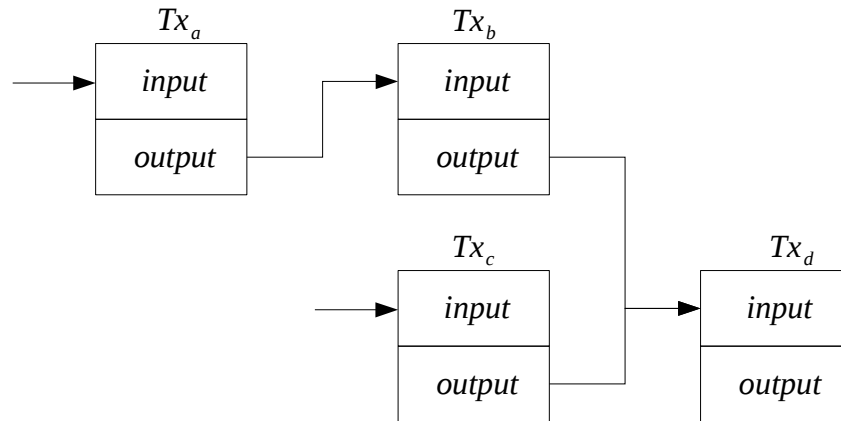


Figure 2: Simple Transaction Chain

vin[]

Each entry in the `vin` array represents an output from a previous transaction. To uniquely identify the desired transaction, the $(hash, n)$ pair is used. The `hash` field is a unique identifier or ID which identifies a single transaction, and `n` is the index of the output within that transaction.

In order to claim the Bitcoins in the chosen transaction, the `scriptSig` field is required to contain a valid response to the `scriptPubKey` field in the selected output.

vout[]

This array represents the outputs of the transaction. There must be at least one output for each transaction.

The Bitcoin protocol does not allow for partial spending, meaning that when a input transaction is claimed, the entire value is used. To determine how many funds should be sent to each output, the `nValue` field is used. The number of BTCs to transfer is set in increments of 10^{-8} , thus allowing for very small amounts of currency to be transferred.

The purpose of the `scriptPubKey` field is to determine who can claim the BTCs in each of the outputs of the transaction. This field is often also called the recipient address.

nLockTime

The nLockTime field exists to allow setting a timestamp or block number after which the transaction will be locked and not allow further modifications. The data is structured as follows:

VALUE	DESCRIPTION
0	Transaction not locked.
<500,000,000	Block number after which the transaction is unlocked.
≥500,000,000	UNIX timestamp after which the transaction is unlocked.

Table 4: nLockTime Values [Okupski, 2014]

Another important aspect of transactions are transaction fees, where a certain amount of the value being transferred is awarded to the miner who generates the block containing the transaction. The amount of value set as a transaction fee is defined by the user who created the transaction, with the possibility of no fee being set. However, the miners creating the block can choose which transactions are included in the blocks they create.

This means that transaction fees are an incentive to have your transaction included in the generated block. It is envisioned that transaction fees will be a more attractive form of earning Bitcoins as the block rewards decrease over time.

3.2.1.2. Coinbase Transactions

While coinbase transactions use the same data structure as regular transactions, some fields hold special values.

#vin

In the case of coinbase transactions, as new value is being created, no input transactions are required, therefore, a single constant transaction is used. Meaning #vin is always set to '1'.

vin[]

A single constant input transaction is provided, but it is not a previous transaction. Instead, the transaction identifier pair is set to a predetermined default value: $(hash, n) = (0, 2^{32} - 1)$

scriptSigLen

In the case of coinbase transactions, this field is often renamed to *coinbaseLen*, but still stores the length of the next field.

scriptSig

This field is also renamed to *coinbase* and holds the block height (number of blocks since the genesis block in the current block-chain) and other arbitrary data used to help miners solve the proof-of-work.

nValue

The purpose of the *nValue* field remains the same, but the actual value must correspond to the mining subsidy, determined by an algorithm which starts at 50 BTCs per block and gets halved every 210.000 blocks. At the time of writing this thesis, the mining subsidy is set at 25 BTCs per block and will be reduced by half approximately every four years.

3.2.1.3. Restrictions

There are also several restrictions which must be observed, both for regular transactions and coinbase transactions, which include:

- The total size of the transaction data must not exceed 10.000 bytes.
- The *scriptSig* data of each input transaction must not exceed 500 bytes. (This restriction may be updated to a maximum of 1650 bytes in a future release of the Bitcoin client software.)
- Dust transactions are not allowed. A transaction is categorized as a dust transaction when at least one output transaction spends over one third of its value in transaction fees.

As well as the previous restrictions, coinbase transactions have a special condition which only allow the value to be spend after 100 blocks. [Btc:DevGuide] This temporarily prevents the miner from spending the Bitcoins earned from a block which later becomes stale. A block is labelled as stale when which is attached to a block-chain which is no longer the longest chain available, and so the chain is abandoned.

3.2.2 Script

Script is a stack-based turing-incomplete language which was designed specially for use in the Bitcoin protocol. This essentially means that it is a simple script language which does not allow for the computation of complex problems, only the problems it was designed for.

The language is based upon a collection of `OP_CODES` which simply are reserved words which control how the script will be interpreted. It is customary for these reserved words to begin with `OP_` and do not contain spaces or lower-case letters. The functionality of most op-codes can be determined by their name, but a full list of codes can be found on the Bitcoin wikipedia page on scripts (<https://en.bitcoin.it/wiki/script>). The full details of this script language will not be explored further in this thesis, but a brief explanation of some of the opcodes may be provided.

In the Bitcoin protocol, scripts are used in the process of claiming received Bitcoins and are included as part of transactions on the network. Bitcoin uses these scripts for two main purposes, and as such can be divided in to two groups, challenge scripts and response scripts.

The first type of scripts are included in each output of a transaction and will include a challenge which the recipient will need to complete in order to use those BTCs. It essentially determines under which conditions the BTCs included in the transaction can be claimed. The second type of script, response scripts, are included as an input to transactions and will contain the solution or response to the challenge scripts included in the transaction they are associated to. [Okupski, 2014]

In essence, including these small scripts in transactions allows users to control who can claim and spend the BTCs being transferred. This feature is required due to the lack of any centralized entity with the power to restrict users from using BTCs belonging to other users. Instead, Bitcoin was designed to leave that responsibility to other users on the network. It is up to them to ensure that the person attempting to spend certain Bitcoins has the right to do so, and ignore the transaction otherwise.

The use of scripts allows this system to achieve this goal by setting a challenge that will be easy for the desired recipient to solve, while being infeasibly complex for any other user to solve and still be easy for any other user to verify the solution provided.

While the Script language itself has enough functionality to allow users to create very intricate conditions under which the associated Bitcoins can be claimed, most of the functionality has been disabled to prevent exploits. Only a small number of standard script templates are currently accepted by the other nodes in the network. While this may seem like a severe limitation, the available templates are sufficient to allow users to set adequate conditions for claiming the associated Bitcoins.

3.2.2.1. Script Templates

At the time of writing this thesis there are only 5 script templates which will be relayed by nodes in the network. An overview of each one will be provided here:

NAME	DESCRIPTION
Pay-to-Pubkey	Obsolete template where the entire public key of the recipient is included in the data.
Pay-to-PubkeyHash	Instead of the entire public key being included in the script, it's hash value is calculated and used instead.
Pay-to-ScriptHash	The hash value of the response script is included instead of the actual script data.
Multisig	Script template where the recipient must prove the ownership of more than one public key, or provide the one or more scripts which produce the provided hash values.
Nulldata	Template which does not included any restrictions. Mainly used for including arbitrary data.

Table 5: Script Templates

Pay-to-Pubkey (P2PK)

With this form of transaction, the sender transfers funds directly to the owner of a public key. This means that in order to claim the transferred Bitcoins, the user must prove ownership of this public key, this is done by providing a digital signature in the response script. [Okupski, 2014]

This script template is still accepted, but is considered obsolete, as the more recent Pay-to-PubkeyHash is more compact and considered more secure. The main difference between the two is that while the P2PK script includes the complete public key data of the recipient, the P2PKH utilizes a hash of the public key, thus requiring a smaller number of bytes. [BtcWiki:Script]

A simple example of a challenge script following this template could be as follows:

scriptPubKey: <pubKey> OP_CHECKSIG

<pubKey> is the complete public key of the recipient and the opcode OP_CHECKSIG instructs the interpreter that the signature contained in the corresponding response script must be verified against the provided public key. The corresponding response script simply includes a digital signature.

scriptSig: <sig>

<sig> represents the signature obtained after hashing the transaction and encrypting the output data with the users private key. For response script to be valid, the private key used should be the pair of the required public key.

Pay-to-PubkeyHash (P2PKH)

As an improvement to the original Pay-to-Pubkey script template, the P2PKH type uses a hash of the public key instead of the complete key. By doing this transactions can be made smaller or can allow for more complex transactions while using the same number of bytes. As well as this, transactions are more secure against any future break in the public key algorithm used, as the hashing algorithm would also have to be broken to allow an attacker to obtain the private key based solely on the transaction data.

Various algorithms can be used to hash the public key data, as long as it is made clear which algorithm was used, so that other users can use the same algorithm. This is done with the use of specific opcodes for each hashing algorithm. Some examples of such opcodes are:

- **OP_SHA256**

Data is hashed using the SHA-256 algorithm.

- **OP_HASH160**

Two hash algorithms are used in sequence, firstly, the SHA-256 algorithm is used, then the output hash is hashed again using the RIPEMD_160 algorithm.

- **OP_HASH256**

This opcode instructs the interpreter to use double SHA-256, in other words, the data is hashed with SHA-256 and then the output is hashed again using the same algorithm.

The following example is a challenge script based on this template:

```
scriptPubKey:  OP_DUP  OP_HASH160  <pubKeyHash>  OP_EQUALVERIFY  
OP_CHECKSIG
```

Where <pubKeyHash> is the hash of the public key of the recipient of the transaction.

In this example, OP_DUP serves the purpose of duplicating the top item on the stack, while OP_EQUALVERIFY performs a direct equality comparison followed by a check to determine if the top item on the stack is TRUE.

The corresponding response script may look like this:

```
scriptSig: <sig> <pubKey>
```

<sig> corresponds to the signature used to prove the ownership of <pubKey>.

Pay-to-ScriptHash (P2SH)

P2SH scripts are in some ways similar to the P2PKH scripts, but instead of being used to send BTCs to the owner of a certain public key, Bitcoins are instead sent to the owner of a specific script, also known as Pay-to-ScriptHash addresses.

With this type of script the sender will typically generate a challenge script of some form, and instead of attaching it to the transaction as would be done in a P2PKH script, the hash value for the entire script is calculated and attached to the transaction. For the transaction to be claimed, the recipient will have to provide the same script as the sender used to generate the hash value, as well as any data required for the script to be evaluated as true. [BtcWiki:P2SH]

This method can ensure the same amount of security as P2PKH scripts, as the provided script must also return true. This means that if the `OP_CHECKSIG` opcode is used, the signature to verify the public key must also be provided.

An example of what a challenge script of this type may look like is this:

```
scriptPubKey: OP_HASH160 <scriptHash> OP_EQUAL
```

Here, the script instructs the interpreter that the desired script has been hashed using the SHA-256 algorithm followed by a round of `RIPEND_160`. `<scriptHash>` represents the hash of the desired script and `OP_EQUAL` instructs the interpreter that the when hashed, the script provided in the response script must match the provided hash value.

The corresponding response script will follow this pattern:

```
scriptSig: <optionalData> {<serializedScript>}
```

Here the `<serializedScript>` is provided by the user redeeming the transaction, as well as any additional data that may be required to ensure that the script returns true when evaluated.

A more concrete example may be:

```
scriptSig: <signature> {<pubkey> OP_CHECKSIG}
```

This response script includes the sub-script (`<pubkey> OP_CHECKSIG`) which when hashed returns the same value as the value provided in the challenge script, as well as a digital signature (`<signature>`) which will be used by `OP_CHECKSIG` to ensure the redeemer actually owns the public key which was provided. If the hash of the sub-script does not match the hash provided in the challenge script, or if the sub-script is invalid or returns false, the transaction is rejected.

The use of this type of challenge script allows for the recipient to develop complex logic for claiming the transaction without the sender requiring prior knowledge of it. The recipient can simply provide the hash of its complex script to the sender in the form of an address.

Multisig

The multisig script type allows the sender to require the recipient to prove ownership of an arbitrary number of public keys. To do so, the challenge script will include a list of n public keys and a number m , which represents the minimum number of public keys the redeemer must own. The redeemer will then need to provide at least m signatures in the same order as the provided public keys.

Multisig scripts can either be based on P2PKH, where public keys are provided, or based on P2SH, where hashes of scripts are provided instead. For the first type, the number of signatures is limited to $1 \leq m \leq n \leq 3$. This means that at most, there can request the redeemer to prove ownership of three public keys. On the other hand, P2SH multisig scripts are only bound by the maximum script size (currently 500 bytes) and the maximum size of the serialized script (currently 520 bytes). This means that it is possible to create higher requirements, such as 4-of-5 scripts, where the Bitcoins can only be claimed when the recipient can provide four valid scripts out of the five script hashes provided. [Okupski, 2014]

Multisig scripts are useful in situations where Bitcoins are owned by more than one individual, and it is not desired to allow any single user full access to the stored BTCs, instead requiring more than one individual to approve the transfer for it to be accepted.

An example of the challenge script can be seen here:

```
scriptPubKey: m <pubKey 1> ... <pubKey n> n OP_CHECKMULTISIG
```

The script begins with the minimum number of keys the recipient must own (m) followed by an ordered list of the respective public keys. Following the last public key, the total number of keys is given (n) and the opcode `OP_CHECKMULTISIG`, which will instruct the interpreter to verify the signatures provided in the response script to ensure that all the signatures provided are valid, and at least (m) were given.

The corresponding response script may follow this pattern:

```
scriptSig: OP_0 <signature 1> ... <signature m>
```

The opcode `OP_0` does not provide any functionality to the script, but it is required due to the stack based architecture of the interpreter, an extra element is popped off the stack while running the script, meaning that the response script must be padded with zero data.

The signatures are evaluated in order, meaning that they must appear in the same order as the public keys in the corresponding challenge script.

Nulldata

Nulldata scripts are unique from the previous scripts as they do not include any requirements in the challenge script and thus do not require any data in the corresponding response script. The goal of these scripts is, unlike the other scripts, not to transfer funds, instead, their purpose is to allow users to include arbitrary data in the transaction.

An example nulldata challenge script is as follows:

```
scriptPubKey: OP_RETURN [arbitrary_data]
```

The opcode OP_RETURN instructs the interpreter to simply return true and evaluate the script as valid. The following data is then ignored by the interpreter and does not need to follow the Script language. This means that a user may use the [arbitrary_data] field to add a small amount (up to 40 bytes) of data. The goal of this will vary from case to case, but may include a small message to the redeemer of the transaction or for anyone else browsing the block-chain.

Transactions with this type of script also do not follow the dust rule, allowing the number of BTCs being transferred to be set to 0. As this type of script does not cost anything to create, there would be the possibility of users attempting to flood the block-chain with useless transactions consisting only of arbitrary data. To avoid this, only one output script of this type is allowed per transaction.

As the script always evaluates as true, there is no need for a response script, and can be left empty.

3.2.2.2. Hash Types

NAME	DESCRIPTION
SIGHASH_ALL	With this type of script, the entire transaction data is concatenated and hashed.
SIGHASH_SINGLE	The transaction data is hashed along with a single output.
SIGHASH_NONE	None of the outputs are included in the hash calculation.
SIGHASH_ANYONECANPAY	Modifier which also removes all other inputs from the hash calculation.

Table 6: Hash Types

Transactions may include scripts which require the redeemer to prove ownership of a give public key by providing a digital signature. This is an effective way to ensure ownership of the corresponding private key as it is

infeasible to generate a valid signature without the private key, and it is easy for any user to verify any given signature.

These signatures are based on hashes of the data being signed, and as with scripts, there is more than one type of signature which can be used. Each type of signature will include different data in the hash calculation. A brief description of the various types will be given here:

SIGHASH_ALL

This is the default behaviour, where the entire transaction is used to generate the signature for the current input.

When using this type of hash, two temporary modifications are made to the data before performing the hash calculation:

All of the opcodes are removed from the response script of the current input transaction and a new length is calculated for the `scriptSigLen` field.

In order to avoid generating signatures which are dependent on previous signatures, the response script field (`scriptSig`) of all other input transactions are temporarily replaced with empty scripts before generating the signature, essentially removing them from the signature calculation process. [Okupski, 2014]

The original data is then restored before restarting the hashing process for the next input transaction.

$$\text{hash value}_{\text{vin}[x]} = \text{hash}(nVersion || \# \text{vin} || \sum_{i=0}^{\# \text{vin}} \text{vin}_{[i]} || \# \text{vout} || \sum_{j=0}^{\# \text{vout}} \text{vout}_{[j]} || nLocktime)$$

SIGHASH_SINGLE

With this option, only one of the outputs is included when generating the signature. However, all the other data in the transaction is included.

As with the previous signature type, some temporary changes are required to ensure the signatures are not dependent on each other. In this case, as well as removing the response scripts on other inputs, the `nSequence` value for the current input is set to 0 and the `#vout` value is set to the index of the current input + 1.

$$\text{hash value}_{\text{vin}[x]} = \text{hash}(nVersion || \# \text{vin} || \sum_{i=0}^{\# \text{vin}} \text{vin}_{[i]} || x + 1 || \text{vout}_{[y]} || nLocktime)$$

SIGHASH_NONE

This option does not include any of the outputs while generating the signature, and as such, these can be changed by other parties. As the previous option, all other data is included, and consequently cannot be altered.

As removing all output data and removing the response script from the other inputs, this type of script also sets the `#vout` field to 0.

$$\text{hash value}_{\text{vin}[x]} = \text{hash}(nVersion || \#vin || \sum_{i=0}^{\#vin} \text{vin}[i] || 0 || nLocktime)$$

SIGHASH_ANYONECANPAY

Unlike the previous types, `SIGHASH_ANYONECANPAY` is a modifier which can be added along side one of the previous hash types. When used, the hash will only include the current input in the signature, instead of the default behaviour where all inputs are always included. The coverage of the signature with regard to the outputs remains unaffected and is determined by the behaviour defined by the selected signature type.

As well as the temporary changes applied by the main signature type in use, this modifier also adds two more modifications. Setting the `#vin` value to 1, and removing all other inputs, leaving only the current input data to be hashed. [BtcWiki:Script]

$$\text{hash value}_{\text{vin}[x]} = \text{hash}(nVersion || 1 || \text{vin}[x] || \#vout || \sum_{j=0}^{\#vout} \text{vout}[j] || nLocktime)$$

3.2.2.3. Signature Creation

Once the signature type has been selected for the current transaction, the actual signature data is calculated in three steps. Firstly, the signature type is appended to the modified transaction data. Then, the data is encrypted using the private key (see section 3.1.2 for more on public key cryptography) and finally, the last byte of the signature type is appended to the encrypted data, this is done so the recipient can easily determine which data was included in the signature without needing to decrypt the data. The validity of the signature can then be easily confirmed by any other user by attempting to use the public key to decrypt the signature data.

$$signature = sign(data_{transaction} || sigtype) || sigtype[i_{last\ byte}]$$

The use of these scripts can lead to the appearance of some interesting cases, such as transactions with scripts with invalid public keys which cannot be claimed by anyone, thus removing Bitcoins from circulation, or transactions without any restrictions which can be claimed by anyone. However, some of these scripts are considered non-standard and will not be propagated in the network.

3.2.3 Addresses

Once challenge and response scripts are understood, understanding addresses becomes easier, as addresses are based upon the public key hashes used in the P2PKH scripts or script hashes used in P2SH scripts.

Addresses are case sensitive and exact, this means that a user must take care when entering an address to ensure it is correct. Some of the characters in the address are used as a checksum, which allow for a fast and easy way to check if the address is valid. However, if the address happens to be well formed, just not owned by anyone, or the owner no longer has access to that address, the BTCs will be lost.

Here is an example of a Bitcoin address with a length of 34 characters:
[BtcWiki:Address]

3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy

To avoid visual ambiguity, the base-58 encoder will remove the following characters: 'I', 'l', '0', 'O'. This helps prevent some errors when addresses are manually introduced.

Due to the properties of the hashing algorithms used to generate addresses, there is a virtually endless supply of unique addresses which can be used. These addresses can be generated by anyone at any point in time, with or without an internet connection. The hashing algorithms also ensure that addresses do not contain any personal information, meaning they can be shared freely without being directly tied to their owner or other addresses.

To generate an address, the following steps are required:

- 1. Get version number.**

For PubKeyHash address 1 byte of 0 (0x00) is used, while for ScriptHash addresses 1 byte of 5 (0x05) is used. These values later get converted by the encoder to '1' and '3' respectively.

- 2. AddrHash**

The AddrHash is a combination of the version number and the hash of the public key or script, depending on the type of address. Two hashing algorithms are used to generate the hash value, SHA-256 followed by RIPEMD-160.

$$\begin{aligned} \text{AddrHash} &= \text{RIPEMD}_{160}(\text{SHA}_{256}(\text{script})) \\ \text{AddrHash} &= \text{RIPEMD}_{160}(\text{SHA}_{256}(\text{pub key})) \end{aligned}$$

As the final hash algorithm used is RIPEMD-160, the final hash length will be 20 bytes.

3. Checksum

In order to easily verify if there were any typos in an address, a checksum is included in the address data. For this, the AddrHash is hashed twice using the SHA-256 algorithm, with only the first 4 bytes of the output being used.

$$checksum = SHA_{256}(SHA_{256}(AddrHash))_{[0-4]}$$

4. **Bitcoin Address.** The last step in generating the address is concatenating the checksum to the AddrHash and performing the base-58 encoding.

$$address = encode_{58}(AddrHash || checksum)$$

The final structure of the address will be as follows:

$$[version_{1\text{ byte}} || hash_{20\text{ bytes}} || checksum_{4\text{ bytes}}]$$

This is known as the binary Bitcoin address and always has a length of 25 bytes. However, the encoder will convert this binary data in to human readable characters, and the resulting string will be slightly longer, usually 34 characters.

When a user wishes to send BTCs to another user, this address can be used in the `scriptPubKey` field.

The name address can cause some confusion for some users, as they can be mistakenly confused with accounts numbers. This is not the case, unlike accounts or account numbers, addresses are designed to only be used once, after which they should be discarded. Not doing so can lead to several problems, including a reduction in privacy. As the number of transactions a single address is involved in increases, the easier it becomes to attempt to associate that address with a single user.

3.2.4 Blocks

A block can be seen as a container used to group various transactions together.

Each block can be divided in to two large components, the header and the payload. While the header has a constant size of 80 bytes, the payload can vary from block to block depending on how many transactions are included in the block, up to a limit of 1MB. In a similar fashion to transactions, when the header is hashed, it is used as a unique identifier for a block, known as the block ID.

An overview of the data structure of a Bitcoin block can be seen in table 7.

	FIELD NAME	TYPE (SIZE)	DESCRIPTION
Header	nVersion	int (4 bytes)	Block version in use. Currently this is set to 2. No other values are allowed.
	HashPrevBlock	uint256 (32 bytes)	Hash value of the header from the previous block in the chain. Calculated using SHA ² .
	HashMerkleRoot	uint256 (32 bytes)	Hash value from the root of the merkle tree which contains all of the transactions included in the block.
	nTime	uint (4 bytes)	UNIX-timestamp of the block creation time.
	nBits	uint (4 bytes)	Target value used to solve the proof-of-work stored in a compact format.
	nNonce	uint (4 bytes)	Random number used to allow solving the proof-of-work.
Payload	#vtx	VarInt (1-9 bytes)	Number of transaction entries in the vtx[] field.
	vtx[]	Transaction (variable)	Array of all the transactions included in the block.

Table 7: Block structure [Okupski, 2014]

nVersion

There are currently only two block versions, 1 and 2. However, since BIP0034 was introduced, only version 2 can be used. Any other values will cause the block to be considered invalid and will not be propagated in the network.

HashPrevBlock

As each block contains a reference to the previous block, in the form of a hash value of the block header, blocks are chained together in a fixed order. This is the basic structure of the block-chain.

Given two blocks A and B, where the `HashPrevBlock` field of Block B contains the hash of the header of block A. It is proved that block B was created after block A. As well as this, it is proved that any transaction included in block A was created before block B. This ensures a level of temporal consistency.

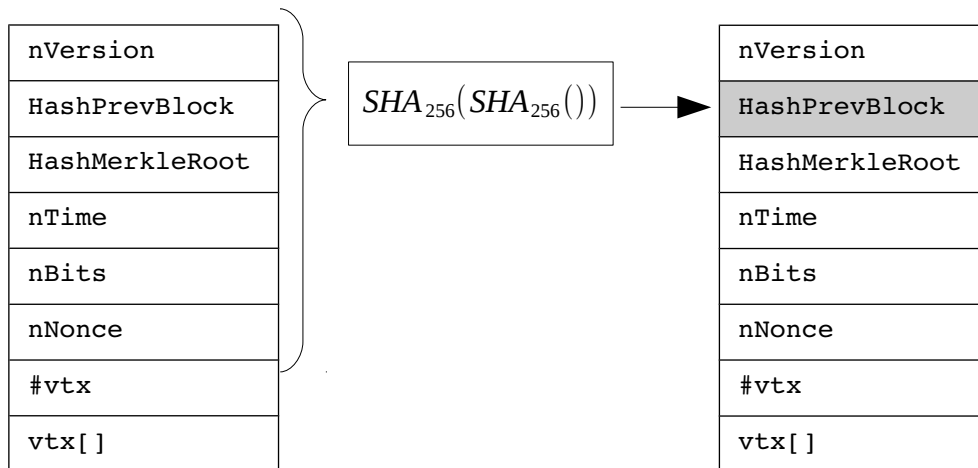


Figure 3: Block Chaining

To create the value used in the `HashPrevBlock` field, the SHA^2 algorithm is used on all the data contained in the block header. Data stored in the payload is not included, as it may later be removed to save space by taking advantage of the Merkle Tree structure. The algorithm to calculate the hash for a given block may look like this:

$$block_{ID} = SHA_{256}^2(nVersion || HashPrevBlock || HashMerkleRoot || nTime || nBits || nNonce)$$

HashMerkleRoot

In order to save space as the number of transactions increases, and consequently, the size of each block, a merkle tree structure (see section 3.1.4) was introduced in the Bitcoin protocol to allow old blocks to be compacted by removing old transactions.

By including the root hash in the block header, transaction data can be removed without altering the block ID (hash). This allows certain nodes to be more lightweight and only download the actual transaction data when needed.

Another advantage of this type of data structure is the ease of maintaining integrity. Whenever data in one of the transactions is altered, accidentally or intentionally, its hash will also be altered. This will then immediately cause the root hash to be invalid. Thus, a single hash value is enough to ensure the integrity of all the transactions contained in the block.

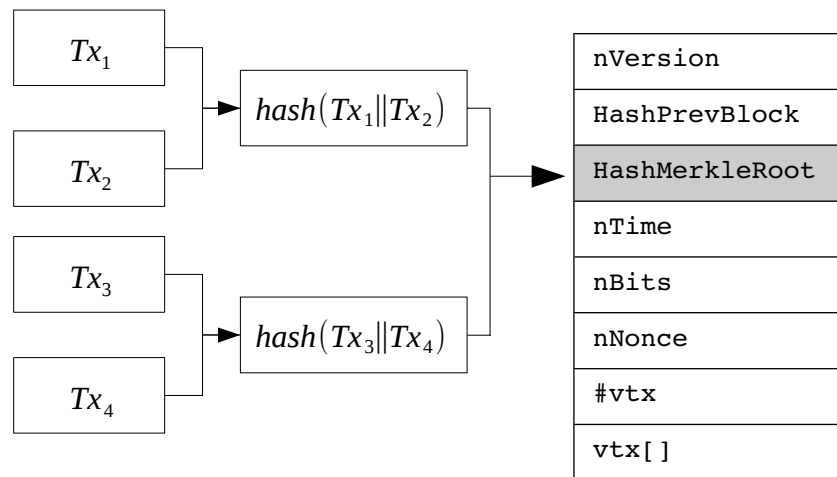


Figure 4: Merkle Tree Root Hash

nTime

This field contains an approximation of the block creation time stored as a UNIX-timestamp. However, as the value is fixed at the start of the process of generating the block, there will inevitably be a discrepancy between the time stored here and the actual time when the block was successfully mined and propagated through the network.

nBits

The nBits field stores a compact representation of the target value used to solve the proof of work (see section 3.1.3). While the actual target value is 256 bits long, the value is compacted to only occupy 32 bits. This is done using the following steps: [Moore, 2013]

1. Convert to base 256. Values in this base can be represented by two HEX digits. For example, $1.000_{10} \rightarrow 0308_{256}$
2. if the most significant digit is greater than 127 (0x7f) prepend 0 (0x00) to the original number. A possible example may be:
 $40.000_{10} \rightarrow 9C 40_{256} \rightarrow 00 9C 40$
3. The first byte of the output will represent the number of digits in the entire original value. If 0x00 was added in the previous step, this is also included in the count. For example, $len(00 9C 40_{256})=0x03$
4. The final step is to append the three most significant digits to the digit calculated in the previous step. The result will be a four digit approximation of the original value. $9C 40_{256} \rightarrow 03 00 9C 40$

To recover the original 256 bit representation of the target value, the following formula is used:

$$N = h_1 h_2 h_3 \times 2^{8 \times (h_0 - 3)}$$

h_0 , h_1 , h_2 and h_3 represent the four components of the compact value. For example, if the compact representation is 1d00ffff, the value is split into groups of two HEX digits, with each pair being assigned to a h_i variable from left to right.

nNonce

The nNonce field only contains 4 random bytes used to add randomness to the mining process. This allows miners to easily adjust one field to completely change the resulting hash, with the goal of selecting a value for the nNonce field which when along with all the other data in the block header allows for a valid solution to the proof-of-work problem.

#vtx

The purpose of this field is to only show the number of values stored in the vtx[] array, i.e. the number of transactions stored in the block.

vtx[]

The main purpose of block in the Bitcoin protocol is to group transactions together in to groups. And this is the field where all the transactions included in the block are stored as an array.

The number of transactions in the array varies from block to block, as well as the number of bytes this field requires. But the Bitcoin protocol currently restricts the maximum size of blocks and any block over 1MB is considered invalid. [BtcWiki:BlockSize]

3.2.5 Mining

Perhaps one of the most important aspects of blocks in the Bitcoin protocol is the way they are generated. Every Bitcoin user has the option to attempt to generate blocks, thus becoming a miner. Figure 5 shows how various users 'compete' to create a new block using the various transactions which have been created and waiting to be included in new blocks.

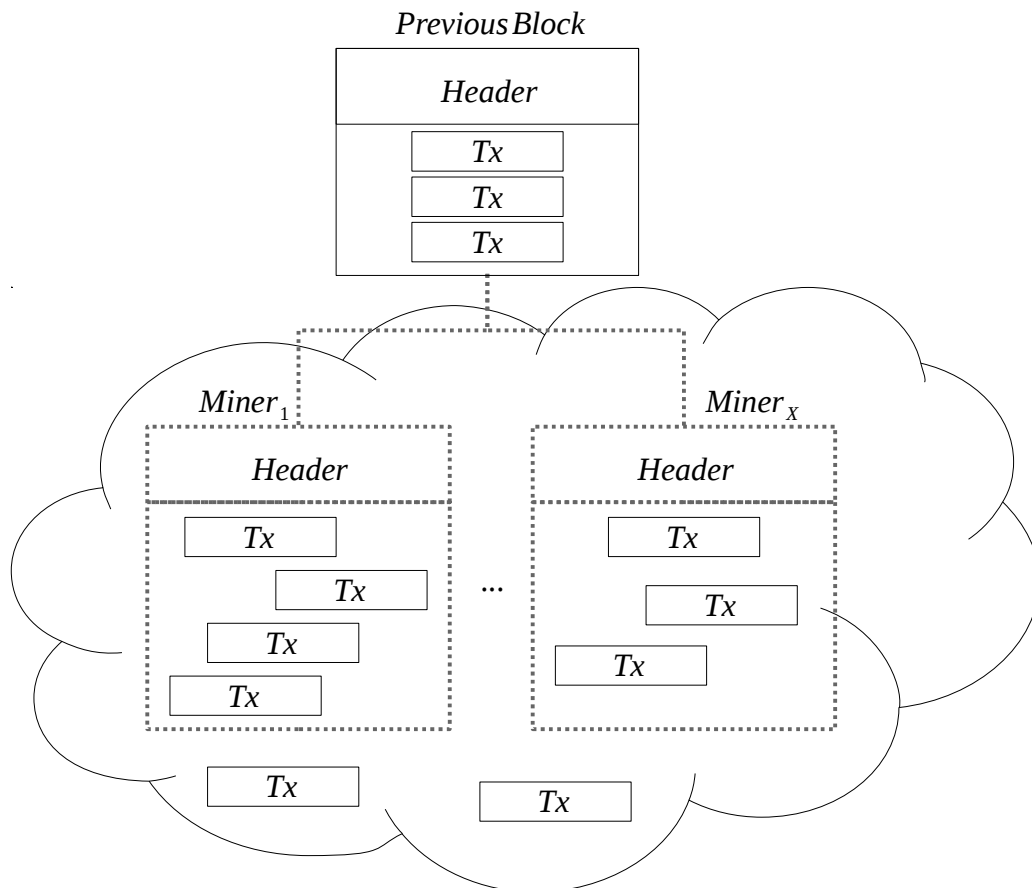


Figure 5: Block Mining

The process of solving a proof-of-work and generating a valid block is called Bitcoin mining and is currently the only way new Bitcoins can be generated and introduced into circulation.

The incentive for becoming a miner is in the form of BTCs. Whenever a valid block is created, the miner is awarded a sum of BTCs as well as receiving the transaction fees from all the transactions included in the generated block.

Difficulty

To help avoid users dealing with the large numbers involved in the proof-of-work problem on a regular basis, the Bitcoin protocol calculates a smaller value to represent the current effort required to solve a proof-of-work. This value is known as the difficulty.

In order to calculate the difficulty value, a constant value is used as the largest target possible, resulting in the lowest difficulty. In Bitcoin this value is known as the 'pool difficulty' or 'pdiff' and was set during the implementation of the Bitcoin protocol as a 256 binary value where the first 32 bits are set to '0' and the remaining bits set to '1'. While this number is smaller than the total number of possible hashes, the decimal representation of this number is still 68 digits long. [BtcWiki:Difficulty] The formula used to calculate the difficulty is as follows:

$$\text{Difficulty} = \frac{\text{pdiff}}{T}$$

As the difficulty value decreases, the less effort is required to solve the proof-of-work, with the lowest possible difficulty being '1', when the target is equal to the maximum target 'pdiff'. On the other hand, the highest possible difficulty is reached when the target value decreases to '1'. With a target value this small, solving the proof-of-work would require generating random data which when hashed returned exactly '0'.

To ensure the block generation time remains somewhat constant, the target value is automatically adjusted by the network on regular intervals. The goal being to keep the block generation time at around 10 minutes per block. During the protocol implementation it was determined that adjustments would occur every 2016 blocks. If the goal of 10 minutes per block is kept, the time required to generate the 2016 blocks would be exactly 2 weeks.

This means that if more than two weeks have passed since the previous adjustment, the difficulty is reduced to allow faster block creation, otherwise the difficulty is increased in order to slow down block creation. The formal formula governing this operation is described below. [Okupski, 2014]

$$T_{\text{new}} = \frac{t_{\text{sum}}}{14 \times 24 \times 60 \times 60 \text{ s}} \times T$$

This formula will adjust the target value T based on the ratio between the total amount of time which has passed since the last adjustment (t_{sum}), in seconds, and the number of seconds contained in two weeks.

To calculate the t_{sum} value, two blocks are taken, the previous block, and the block 2015 blocks before it. The timestamps are then subtracted, resulting in the number of seconds which have passed between these two blocks.

In order to avoid the target value fluctuating huge amounts between adjustments, the maximum change that can be made in each adjustment is a factor of 4. To do this, two simple checks are implemented, if the t_{sum} value is above 8 weeks (4.838.400 seconds), the t_{sum} value is set to 4.838.400 seconds. On the other hand, if the t_{sum} value is lower than half a week (302.400 seconds), the value is set to 302.400 seconds. [Moore, 2013]

Priority

When mining nodes collect transactions to include in each new block, a calculation can be made to determine which transactions to include first, i.e. to calculate the priority of each transaction.

To calculate the priority of a given transaction, the following formula is used, where $vin[i]$ represents the input transaction of index i .

$$Tx_{\text{priority}} = \frac{\sum_{i=0}^{\# \text{vin}} (value_{vin[i]} \times age_{vin[i]})}{length}$$

value : the amount of BTCs contained in the used input transaction measured in satoshis.

age : the number of blocks created since the transaction was included in the chain.

length : the size of the complete transaction data in bytes.

Procedure

For a user to start attempting to start mining a block, the following procedure is followed: [Okupski, 2014]

1. Collect transactions

The mining software will collect all broadcast transactions and select the ones the miner wishes to include in the block. The first transaction in the block will be the coinbase transaction, created by the miner which claims and spends the block reward, as well as the transaction fees for all other transactions included in the block.

2. Verify transactions

All the transactions to be included in the block will be verified. This includes verifying that each input to a transaction includes a response script that allows the user to claim those BTCs. If any of these checks fail, the transaction is not included.

3. Select block chain

The miner will select the latest block on the longest known block-chain to be the parent of the current block. Several block-chains can exist concurrently, and each miner should select the chain with the most number of blocks, or more precisely, the chain with the most expended computational effort. The header of this block will then be hashed and included in the header of the new block.

4. Solve proof-of-work

The final step is to solve the proof-of-work problem for the current block (section 3.1.3). Once that is done, the block can be broadcast to all other nodes and included in the block-chain.

If at any point during the process, another block is published, the miner will first validate the received block and ensure that a) the proof-of-work solution is valid, and b) that all the included transactions are also valid. If these checks pass, the miner will restart the procedure and start mining a new block.

In order for the proof-of-work to be solved, the miner has access to two sources of 'randomness' in the block header. The first, and most obvious, is the `nNonce` field. The sole purpose of this field is to allow miners to increment the value and generate a completely new hash value each time they do. However, in some cases, a miner may iterate over every possible 4 byte value which can be stored in this field without obtaining a valid solution to the problem. When this happens, the second source of 'randomness' used.

This is the `MerkleRootHash` field and is used by altering the coinbase transaction included in the block.

As discussed in section 3.3.1, a coinbase transaction is a special type of transaction which assigns new BTCs to the creator of the block. In essence, this is a transaction created by the miner assigning BTCs to himself. This means that this is often the only transaction which the miner can alter during the mining process without invalidating it.

A typical change which can be made to the content of the coinbase transaction is to alter the data stored in the `coinbase` field (named `scriptSig` in regular transactions). This will in turn change the hash of the transaction, resulting in a cascading change in the merkle tree hashes going all the way up to the root hash stored in the `MerkleRootHash` field. The mining process can then continue by iterating over the possible values for the `nNonce` field with the new coinbase transaction in place.

3.3 The Bitcoin network

The Bitcoin protocol is built upon a decentralized network of nodes, organized into a peer-to-peer structure. This means that there is no central server to organize the nodes or to store data. Instead of having a central agency to control the nodes and maintain security, the Bitcoin protocol relies a set of pre-determined rules and protocols which all (or most) of the network nodes abide by and enforce while exchanging transactions and blocks. As long as most of the nodes on the network are honest, the whole system remains operating as intended.

In this section, more information will be given on how architecture of the network and how nodes find and communicate with each other.

As the popularity of the protocol has increased, some alternative protocols for network communication have been implemented. For example, some mining groups have designed their own network for sharing blocks at a higher speed [Corallo, 2013], and some wallet software alternatives have dedicated servers for storing transaction information [Git:electrum]. However, these aspects of the protocol will not be discussed in this thesis, as they are not part of the original or default behaviour.

3.3.1 Block chain

Blocks are a central part to the Bitcoin protocol, and as each block contains a reference to the block which came before it, a chain of blocks is created leading all the way back to the genesis block. This data structure is called the block chain.

The purpose of the block chain is to be a public ledger of all the transactions performed on the Bitcoin network. This data structure is unique from that employed by fiat currencies and allows any user to browse the entire block chain and view any transaction ever performed.

However, as the process involved in creating a block is probabilistic by nature it is possible for two miners to generate valid blocks at approximately the same time. As each block is propagated through the network it is entirely possible for a significant number of nodes to receive one block before the other. This will result in a fork. [Okupski, 2014]

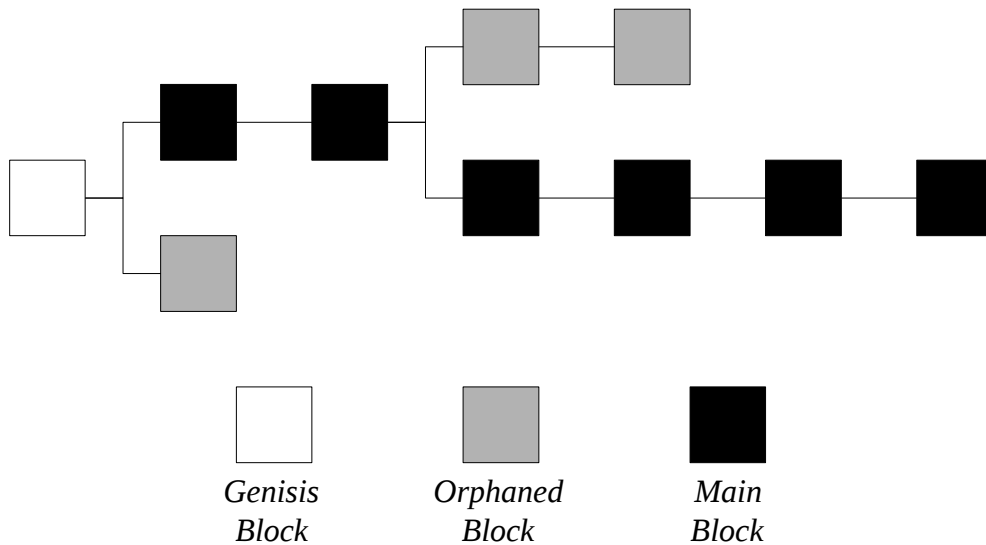


Figure 6: Normal Block-Chain Fork

This happens because once a node receives the first block, it will reject the second one. As the second block will have the same parent, and most likely, include one or more of the same transactions. But, each node will still keep the second block it receives as long as other verifications pass.

Forks of this nature are usually resolved as the next blocks are generated. As miners continue to work on each branch, the first miner to generate a successful block will propagate it through the network and thus extend the chain it considers to be the longest. As other nodes receive this block they drop any work on other chains, which become stale as no more nodes attempt to extend them.

As forks become stale, the transactions included in them are re-added to the transaction pool in order to be re-validated and added to a new block, this ensures that all valid transactions are included in the longest block chain and no data is lost due to forks.

Another form of fork can occur when different nodes are running different versions of the client software. When new validation rules are implemented, there will be a period of time where two versions of nodes are running concurrently. One set of nodes will be using the old rules, while other nodes will be using the new rules. It is entirely possible that the nodes running with the updated rules accept a certain block as valid, while nodes using the old rules consider the same block invalid. When this occurs, the block chain will also fork.

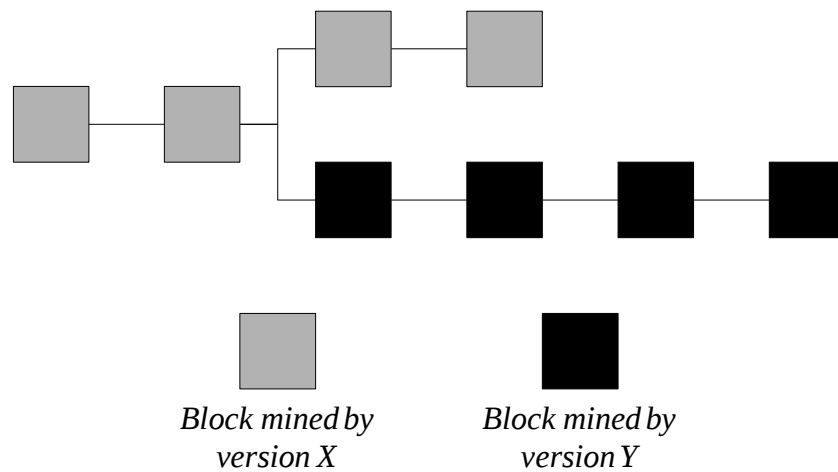


Figure 7: Hard Block-Chain Fork

This type of fork is resolved as the number of nodes running the new version increases. As the number of nodes running with the old rules decreases, the amount of effort expended attempting to extend that branch decreases and will eventually become stale. When planning updates to the Bitcoin protocol, hard forks are a consequence which should be carefully considered.

3.3.2 Network communication

As the various nodes are distributed on a peer-to-peer network without any central entity, nodes need a method to discover new peers and communicate with them.

To achieve this goal, when a Bitcoin client starts up, it will make use of several hard-coded DNS seeds. The response from these servers will include a collection of DNS A records (DNS A records → host addresses) of peers which accept new incoming connections.

The DNS seeds are maintained by the Bitcoin community, and while some of the stored addresses are updated manually and may point to nodes which have since become inactive, others are added automatically by scanning the network for nodes running on the default port (8333 or 18333 for the Bitcoin network and test network respectively).

As soon as a successful connection is made to a valid node, addresses of other nodes can be shared between nodes using `addr` (address) messages. This allows for a true decentralized network without relying exclusively on the DNS seeds.

Once a node has been found communication begins with a `version` message, which includes relevant information about the current node. The receiver will then respond with its own `version` message to complete the communication setup.

More information on the structure of the version message can be found on the Bitcoin developer reference [Dev:Version].

Once communication between two nodes has been successfully established with the use of version messages, an initial sync is required. The purpose of the sync is to download the current version of the block chain. As various forks of the block chain can exist concurrently, the node attempting the sync will attempt to request data from as many nodes as possible, and consider the longest received chain to be the valid chain.

This synchronization is important and must be done before any new transactions or blocks can be verified by the node.

Up until version 0.9.3 a simple method often named 'Blocks-first' was used. However, in more recent versions this method has been replaced with a newer method named 'Headers-first'. This new method allows for improved parallelization as it splits the process of downloading a block in to downloading the header and downloading the block data. This means that a

node can download the payload of one block while continuing to download the headers for other blocks.

When a miner creates a new block it must be broadcast to other nodes in order to be validated and included in the block chain. To achieve this, the miner can use one of two methods.

1. Unsolicited block push

The miner sends the new block to all of the nodes it is currently connected to. As all of these nodes do not have the fresh block, they will accept it, and if it is deemed valid, add it to their block chain.

2. Standard block relay

With this method the miner acts as a standard relay node by sending a `inv` message (inventory message) to all of the known nodes with a reference to the fresh block. These nodes will then reply with a message requesting the block.

In both cases, the receiving node will verify the block to ensure it is valid before propagating it further in the network. If it is deemed invalid, the block is simply ignored.

A similar method is used when broadcasting transactions. But in this case, the sender will always use an `inv` message to send a reference to the new transaction and cause other nodes to request the new transaction data.

Other nodes will then verify the received transaction data to ensure it is valid or not. If it is deemed valid, it may continue to distribute the transaction to other nodes. Some nodes may also include the transaction in a memory pool with the purpose of attempting to include it in a future block.

3.3.3 Bitcoin mining pools

As the difficulty of the proof-of-work problems increases with the increase of computational power available on the network, it has become more and more infeasible for home computers to mine blocks and earn rewards. In order to address this, a method was devised to allow multiple computers to work together to solve the proof-of-work problem. These systems have been named mining pools.

The general principle behind this system is simple and similar to lottery pools. The problem is divided among the participating nodes, and when the problem is solved, earning the Bitcoin reward, the BTCs are distributed among the participating nodes. In essence, each node is forfeiting a portion of the reward in exchange for a higher probability of winning, or in this case, solving the proof-of-work and mining a block.

In practice, several methods have been developed to implement this idea, with different reward schemes and network communication structure. While the Bitcoin protocol is decentralized, mining pools can implement other network structures, allowing some to be peer-to-peer based and others to rely on a central server to manage the various nodes. A simple example of a mining pool with a central server is shown in Figure 8.

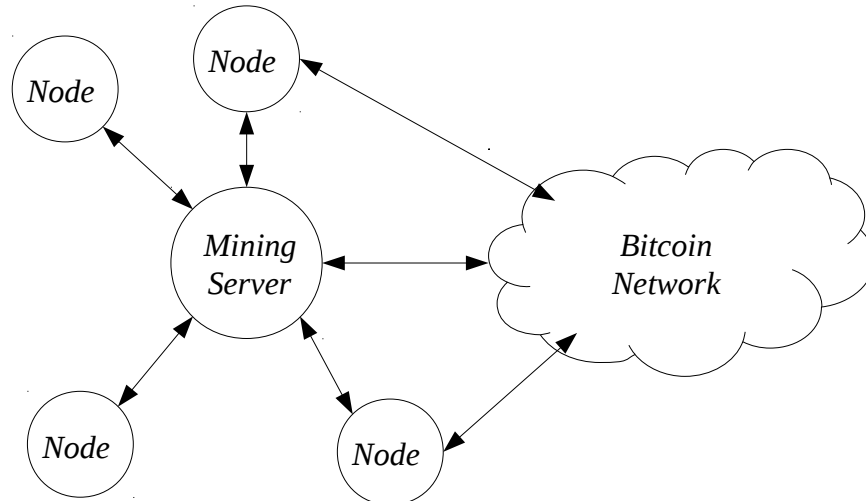


Figure 8: Mining Pool Structure

The first step is to create a new target value (T_{pool}) higher than the current mining target. This means it is easier to generate hash values that are below the pool target than values below the mining target.

$$1 \leq T_{network} \leq T_{pool} \leq 2^{256}$$

Now that a target value for the mining pool has been established, all participating nodes will attempt to use the block header to generate a hash value smaller than the T_{pool} and propagate it in the mining pool. While this value may not be a valid solution for the proof-of-work, it shows the mining pool that the node has expended a certain amount of effort, or a share of the work. The incentive being a higher proportion of the reward is awarded to nodes which have completed more shares.

Due to the probabilistic nature of the algorithms involved, eventually a hash value will be generated which satisfies the pool target as well as the current mining target. When this occurs, a valid block is generated and the reward can be claimed and shared among the participating nodes.

So while each node does not actively attempt to solve the actual proof-of-work problem, and instead works on an easier problem, it is probable that one of the nodes will eventually generate a hash which solves both problems simultaneously. It would be possible to have all nodes attempting to solve the global proof-of-work, but when a solution was found, there would be no fair method to distribute the rewards, as there is no measure of how much effort each node has expended.

Mining Farms

Another solution to the increase of computing power available on the network are Bitcoin farms. This approach is similar in the sense that several computers work together to solve the proof-of-work problem, however, unlike with mining pools, mining farms are generally owned by a single user or entity. This means that any rewards earned do not need to be shared among several users and evidence of expended effort is generally not necessary. The result is that each node is attempting to solve the proof-of-work with the $T_{network}$ value and the block can be published as soon as any of the machines finds a solution.

A typical mining farm may include anywhere from two to several thousand computers on a network working on generating the next block. In several farms specialized hardware is employed to further increase the hashing speed and thus improve the probability of generating a valid block.

Farms can also implement specialized software solutions to manage the various machines. This is done to prevent multiple machines from using exactly the same inputs to solve the proof-of-work.

4 Analysis

While the previous section discussed the internal workings of the protocol and give a more detailed look at how each of the components operates, the goal of this section is to analyse the complete system for some security and privacy issues and determine if these issues can be solved or if they are a serious threat to the operation of the protocol.

This section is divided in to two main sections. Firstly, some of the technical issues are analysed and discussed. With the second section being dedicated to exposing some of the effects the Bitcoin protocol has had on society in the form of legal and economic changes.

4.1 Technical Analysis

In this section some technical aspects of the Bitcoin protocol will be discussed and analysed for possible issues in an attempt to determine if each one is a major concern and if any possible counter measures are available.

The issues discussed in this section include the anonymity of Bitcoin addresses with some information on services which aim to increase it further, as well as some possible attacks which could be employed against the network.

4.1.1 Address anonymity

Bitcoin, as well as other similar services, are often advertised as an anonymous currency. However, this claim may appear to be false at first glance, as the entire history of every transaction ever made is freely available to the public in the form of the block chain. So how can users have any assurance that information about the transactions they make can be kept private?

This confusion is raised when people attempt to compare cryptocurrencies such as Bitcoin with more traditional banking services, where each person has a unique bank account number. This account number does not change and uniquely identifies each individual or organization. This allows the bank to trace any transaction to the individual who created it, and can share that information with any other entities. Having a public ledger of transactions in this kind of system would destroy any user privacy.

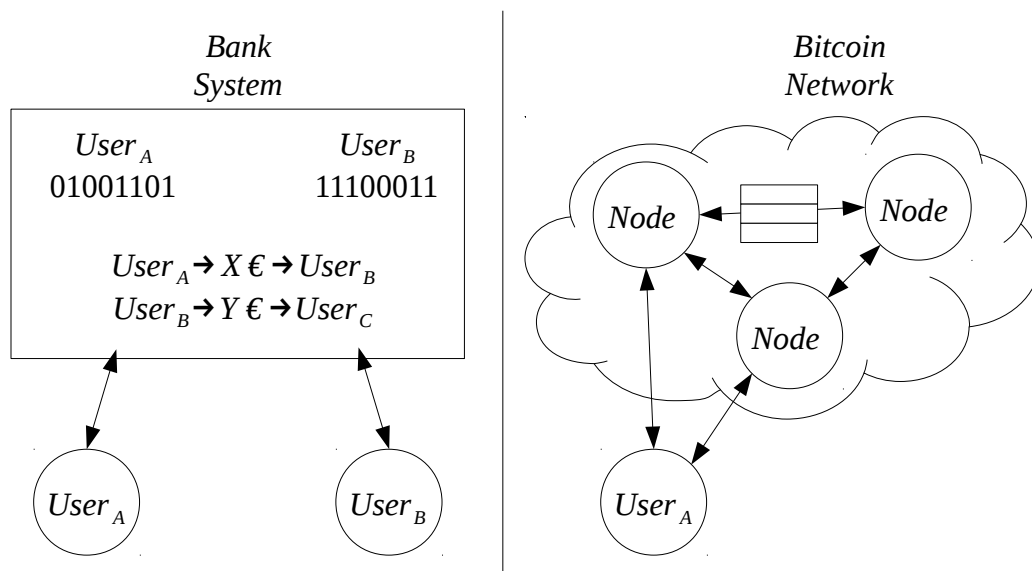


Figure 9: Bank System vs. Bitcoin Network

Figure 9 is a simple comparison between the structure of the Bitcoin network and that of a simple bank. In the bank system a unique ID is stored for each user and a record of all the transactions which have been ever been made with a clear connection to the users involved. On the other hand, the Bitcoin protocol does not assign unique identifiers to any of its users.

It may be counter intuitive, but Bitcoin users do not actually own any currency. Moreover, BTCs are not transferred from one user to another.

Instead, Bitcoins are locked in such a way as that only the desired recipient can claim them. This feature is provided by the use of public key cryptography, where anyone claiming the BTCs must prove ownership the private key linked to the provided public key.

This means that users can, and should, create multiple key pairs (or addresses) and do not use any address more than once. This technique makes associating any address to a single user much more difficult, as each address will only appear once in the entire block chain.

Static address

Some institutions accept Bitcoin donations or payments, and in this case, it is much simpler to generate a single address and allow all users to use it to transfer funds to this address instead of generating a new one for each donation/payment.

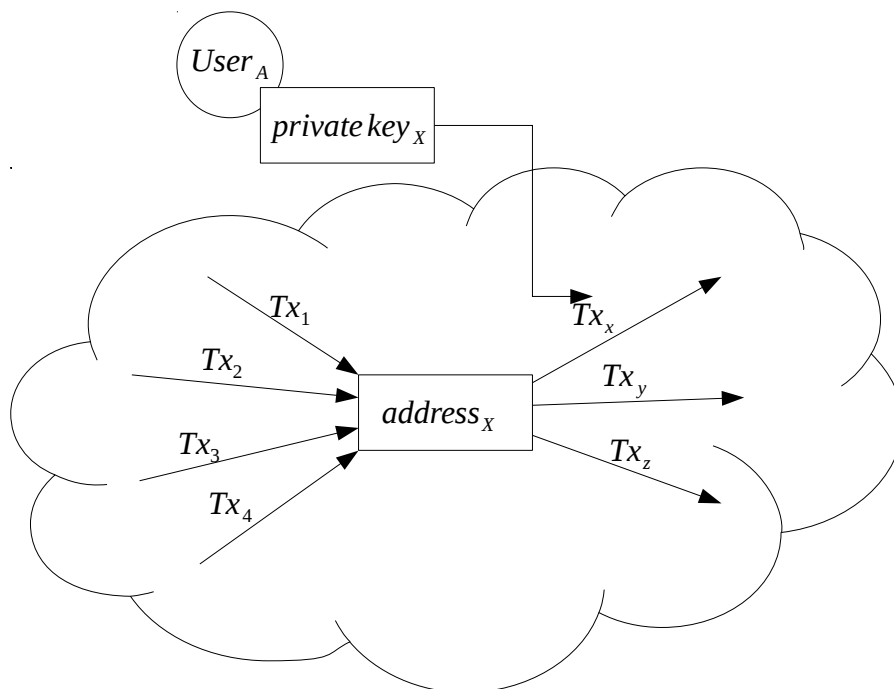


Figure 10: Static Address

A simple example is shown in Figure 10, where $User_A$ uses their private key to claim all the BTCs sent to $address_x$.

This can cause issues where the destination address is known to belong to certain organization, so anyone inspecting the block chain can find every transaction to ever use that address as an output. However, this is not

a major concern for users who transfer funds to static addresses, as the addresses used to send the funds should still be unique.

Exchange services

When users wish to exchange fiat currency, or another cryptocurrency, for Bitcoins, they will usually resort to a currency exchange service. These services allow users to buy and sell Bitcoins in exchange for other currencies and are very important for the Bitcoin market.

However, they will often require users to provide personal information which will allow the service to uniquely identify a single individual.

The exchange service will then have a direct link between an individual and the address which received, or sent, the BTCs involved in the exchange, giving any adversary with access to this information a good starting point when attempting to trace the activities on a certain individual.

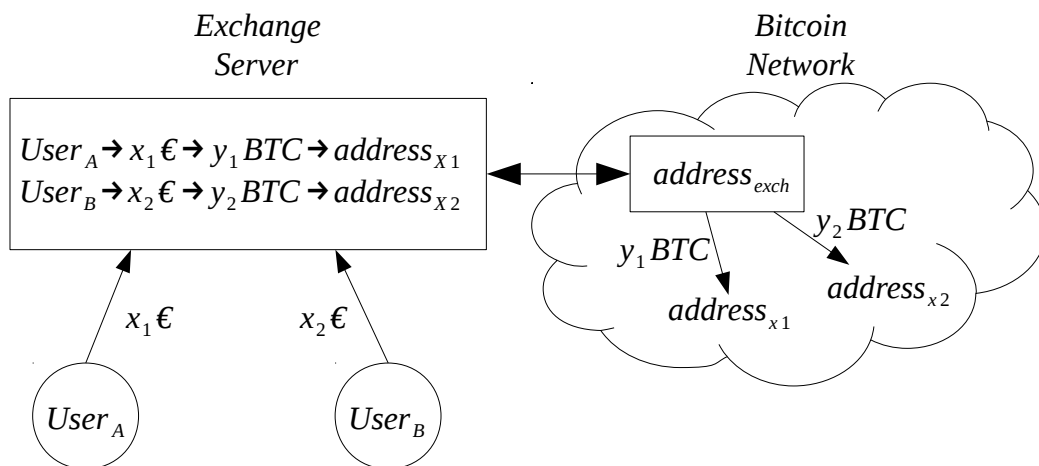


Figure 11: Bitcoin Exchange Server

Figure 11 shows a sample exchange service where $User_A$ and $User_B$ purchase Bitcoins in exchange for a certain amount of Euros. The Exchange server then uses an address it controls to send the desired amount of Bitcoins to the addresses provided by the users.

The server will then have a record of transactions which allows linking each unique user to a certain address, thus reducing user privacy when using Bitcoin.

Mixing services

To help increase privacy even further, services were created with the goal of anonymizing transactions even further. These services have been named mixing services and are used to help prevent any user from following a certain amount of Bitcoins through the block chain.

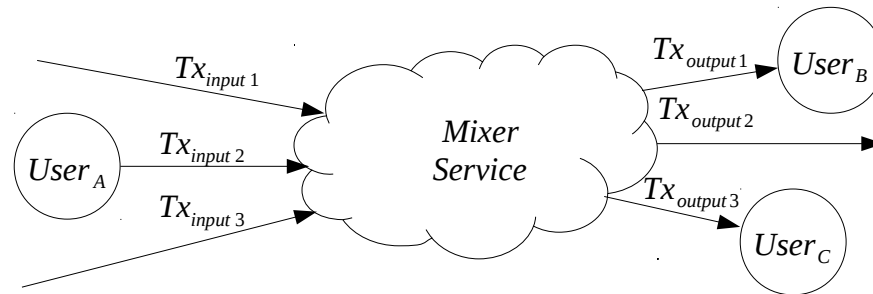


Figure 12: Mixing Server

Figure 12 provides a simple abstract example of how a mixer service operates. Without access to the data stored by the service, it is not possible to determine if the transaction $Tx_{input 2}$ made by $User_A$ was transferred to $User_B$ or $User_C$.

Several services of this kind are currently in operation, with each implementing slightly different rules and protocols and each will charge different amounts of BTCs for the service, usually in the form of a percentage of the performed transactions. But the general principle stays the same. Instead of transferring BTCs directly to the recipient, users will instead transfer the funds to an address controlled by the mixer service. Once the payment has been confirmed, the service can then join and split several transactions over time and finally create a new transfer from another, unrelated address, to the final recipient.

This works well because the origin of the transfer is irrelevant from the recipients perspective, therefore it makes no difference if the payment is made by the actual user receiving the goods or services or by another random Bitcoin user. This process helps prevent any attacker drawing conclusions on where any of the individual final payments actually came from.

However, mixing services are not perfect, as some services will keep records of the transactions made, allowing anyone with full access to the service data the ability to track the transactions. This means that if the service is hacked or cooperating with other entities the provided anonymity may be compromised.

Another issue with mixing services is that not all services operate in the same way, with some services using more complex algorithms when attempting to conceal links between transactions. This results in different services providing different levels of privacy to their users.

A study by Malte Möser found that while two of the tested services (Bitcoin Fog and Blockchain.org) did provide adequate privacy, with no discernable link between the original and final transactions being found, another mixing service by the name of BitLaundry did not adequately obfuscate the connections between the input and output transactions. This allowed the transactions to be traced even when using the mixing service. [Möser, 2013]

4.1.2 Double spending

Double spending describes the act of creating two or more valid transactions claiming the same Bitcoins. This can be an issue with the decentralized nature of Bitcoin as some nodes may receive one of the transactions before the other, thus creating two different versions of events. [BtcWiki:DoubleSpend]

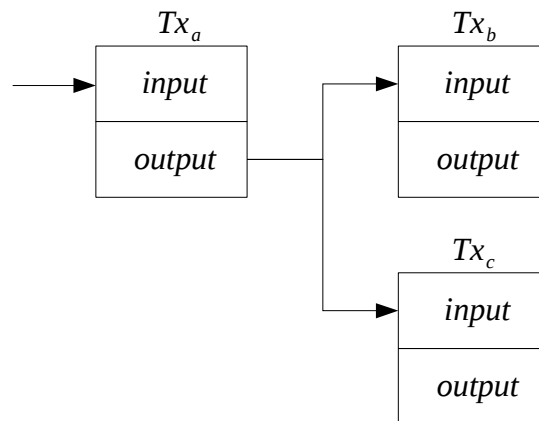


Figure 13: Double Spending Transaction

Figure 13 shows a simple double spending attempt where two transactions are created using the output of the same transaction. This is not possible, as the protocol does not allow for partial spending. The entire value of a transaction must be spent in a single transaction. In a real attack Tx_b could be a valid transaction purchasing a good or service from a merchant, while Tx_c could be another valid transaction sending the same BTCs to another address controlled by the sender.

In a traditional banking service this problem is more easily solved, as only one version of the transaction history can exist. This means each transaction can be verified for double spending before being processed.

The Bitcoin protocol on the other hand allows for various different block chains to be in existence at any given time, and it is possible for different chains to have a different transaction history. As with other aspects of the protocol, the responsibility of protecting the network against double spending is placed on the various nodes on the network.

While various methods are available for performing a double spending attack, a common attack method involves creating two distinct transactions attempting to claim BTCs from the same previous transaction. Typically, the recipient address of one will be a merchant, while the second address will be controlled by the sender. To avoid this situation, the protocol requires that once one of the transactions is included in a block, the second transaction cannot be included in the same block or any subsequent blocks. This can be easily enforced as it is easy for the miner to verify if any of the input transactions have already been claimed.

However, it is still theoretically possible for the transactions to be included in different blocks, creating a fork, either naturally or by the attacker. But this issue will be resolved when one of the forks becomes stale.

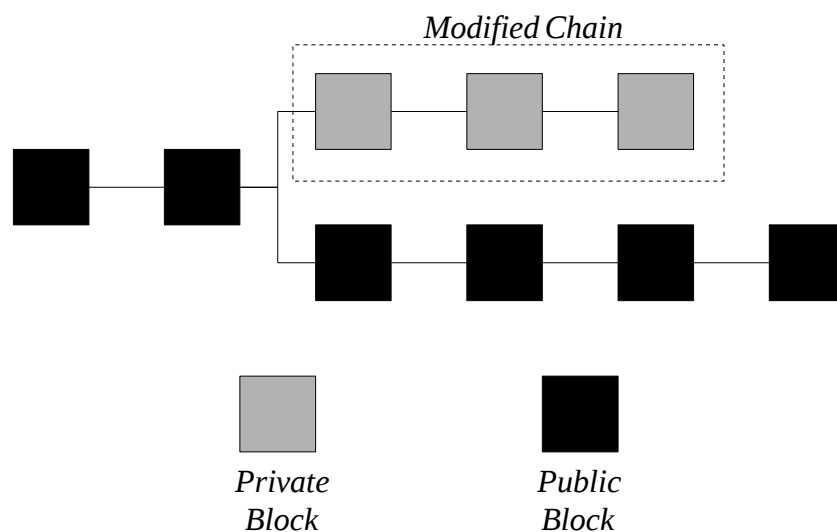


Figure 14: Double Spending Attack

Figure 14 shows a simple example how a double spending attack can be attempted. In this case the attacker has successfully mined a new private block chain with the transaction which would invalidate the payment transaction. The rest of the network continues to work on a block chain which includes the valid payment transaction. The attacker will continue generating blocks on the modified chain until it is longer than the main block chain, then releasing his version of the chain. As other nodes adopt the new chain, the payment transaction will effectively be reversed.

For services receiving payments in the form of Bitcoins, double spending attacks can cause problems as the merchant will not be able to spend the received BTCs if the received payment is later considered invalid.

While it not possible to reduce the risk of this form of attack down to 0%, there are some precautions a service can take to decrease the risk and increase the attack cost. The most important precaution merchants should implement is to delay accepting a transaction until several blocks have been built on top of it. In this context, those blocks are often referred to as confirmations. [Karame et al., 2012]

Each new confirmation decreases the probability of the transaction, and all subsequent blocks, being discarded and replaced by another chain. Typically, it is advised to wait for at least 6 confirmations before assuming the payment is complete. As long as the majority of the network is controlled by honest nodes, it becomes less and less likely an attacker could create a new chain without the payment transaction to replace the current chain.

Another precaution merchants can employ is to restrict the client software to only connect to well connected nodes, i.e. nodes which are connected to a large number of other nodes, and reject incoming connections. This will help prevent a malicious entity feeding blocks directly to the merchants client software and ensure that any received blocks have also been received and validated by a large portion of the network.

4.1.3 Botnet farming

Due to the nature of the Bitcoin protocol there is an incentive to control greater and greater computing power in order to have an increased hashing rate. In other words, to be able to calculate more hash values in the same span of time. There are several ways legitimate to achieve this, such as mining pools or mining farms. However, another, more nefarious method has also been used, botnets.

A botnet is a group of computers which have been infected by malicious code and can be remotely controlled by an attacker. The type of infection will dictate the amount of control the attacker has over the infected machine, but in some cases machines can be used to mine Bitcoins without the knowledge or consent of the owner. [Wiki:Botnet] This means an attacker in control of botnet could mine large amounts of BTCs without having any hardware or electricity expenses. [Bradbury, 2013] As well as this, a sufficiently large botnet could have the power to perform double spending attacks. However, this form of attack will only have any chance at success if the attacker can consistently control over 50% of the computational power on the network. Otherwise, honest nodes will generate legitimate blocks at a higher rate, thus causing the 'fake' chain to eventually become stale.

While there is no security mechanism embedded in the Bitcoin protocol to detect or prevent botnets from mining bitcoins, as the power of legitimate mining pools and mining farms increases, the required computing power required by the botnet in order to be effective also increases. This reduces the incentive for botnets attempting to mine Bitcoins instead of other, perhaps more lucrative, malicious activities.

Botnets have some inherent limitations when attempting to mine bitcoins. Firstly, infected machines are not running 24 hours a day, with many only being available for a few hours a day. Secondly, as other applications may be running at the same time, the full potential of the infected machine cannot be used for mining. As well as this, the malicious software will generally only use the machines CPU to solve the proof-of-work problems, instead of the more efficient GPU. These limitations severely increase the number of machines needed to compete with legitimate mining pools or farms.

Another issue botnets have is that a complex network architecture is required in order to hide the operation from other users and the authorities. As the simplest implementation is to have a central server to control all the infected machines, each infected machine will require a method to find and

connect to this central server, which may be in the form of a hard-coded IP address or host-name. This means that when the malicious code is analysed, this address can be found and efforts can be made to shut down the server, thus disabling the botnet.

It is estimated that in 2013 the total network hash rate was over $1PH/s$ (10^{15} hash calculations a second) and the hash rate for the largest known botnet at the time, ZeroAccess, with 1.9 million machines was only $2.85TH/s$. i.e. 0.285% of the total hash rate. [Bradbury, 2013] And as of November 2015, the global hash rate is estimated at over $450PH/s$, decreasing the effectiveness of botnet farming even further. [blockchain:hashrate]

4.1.4 Flooding

A flood, or transaction flood, is a form of denial of service attack directed at the Bitcoin network. The principle behind this attack is relatively simple, create as many valid transactions as possible between two addresses the attacker controls and broadcast them to the network. This will flood the network with new unconfirmed transactions to be included in new blocks.

As the number of unconfirmed transactions in the memory pool increases, the amount of time required for a new, legitimate, transaction to be included in a block increases. This is caused by the size limit applied to blocks and the constant rate at which blocks are created. This can result in legitimate transactions requiring several hours to achieve a single confirmation, i.e. being included in a block. [BtcWiki:Flood]

The Bitcoin protocol does include some measures to decrease the incentive to perform this attack as much as possible. These measures include:

Transaction fees

Mining nodes have the option to ignore transactions if they do not include a transaction fee, meaning that for this attack to be most effective, a transaction fee must be spent, creating a monetary cost for the attacker.

Transaction priority

As well as the monetary cost caused by the transaction fees, miners also prioritize transactions based on the amount of time since the Bitcoins included in the inputs were last spent. This results in a lower priority for transactions which are repeatedly spending the same Bitcoins over and over.

This reduces the effect of the attack by allowing more legitimate transactions to be included in blocks before the flood transactions.

As well as these protocol implementations, the incentive for this form of attack is reduced further by not providing the attacker with any kind of benefit, other than that, if successful, the attacker may be responsible for temporarily incapacitating the Bitcoin network.

However, even with these measures, this form of attack has been performed in the past, and is often referred to as a stress test, as it can be used to determine how well the network can handle the increased traffic. One notable example occurred in July 2015, when an unknown entity began a decentralized flood attack causing thousands of unconfirmed transactions

being distributed through the network. At one point as many as 80,000 unconfirmed transactions were waiting to be included in blocks.

In this instance, thousands of transactions were also created directed at some charities and organizations with very small amounts of BTCs, resulting in around 30BTCs being donated to various institutions. This is unusual, as the attackers will often attempt to reduce the amount of BTCs spent as much as possible.

The attack seems to have concluded in July and at this time the motive is not known. Some speculate that it may be connected to the on-going debate on the proposed increase in block size, this attack would then be an effort to show the issues with limiting the block size to 1MB. [BtcWiki:JulyFlood]

4.1.5 Software Errors

As the Bitcoin protocol relies so heavily on software to generate currency, ensure the network is operating as designed and correctly transferring funds, software errors or bugs are a major concern. These may be completely accidental, or intentional 'back-doors' implemented in the code by malicious developers.

The open-source nature of Bitcoin protects users, to a certain extent, against both types of software errors by allowing any developer to examine the entire source-code and search for possible bugs or back-doors. However, being open-source also allows any malicious developer to clone the Bitcoin client and create a malicious version of the client which does have an error or a back-door known to the developer. This could then allow the creator to exploit machines running his version of the software.

While this will always be a threat to individual users, in order for it to be a significant threat to entire network, any malicious version of the software would need to be adopted by a large proportion of Bitcoin users. This is unlikely as the official software solutions are free and easily obtainable from the official sources and can also be compiled from the source-code.

On the other hand, numerous accidental errors have been discovered in the Bitcoin client code, ranging from memory overflows to transaction opcode exploits. [BtcWiki:Vulnerabilities] These bugs range in the effect they could have on the network as a whole, as some allow for billions of Bitcoins to be generated without performing the required work, while others are less serious and only increase the effectiveness of potential DoS attacks.

However, whenever such bugs have been discovered, the code has been promptly fixed by the developer community and rapidly distributed to a large portion of the user base. And while many bugs have been discovered, there is no evidence that any have been exploited on the Bitcoin network.

While the most dangerous security threats are located the code that communicates with other nodes and validates transactions and blocks, other serious vulnerabilities have been found in the wallet component. This part of the software is used to aggregate all the addresses the user creates along with the associated private keys needed to claim the received Bitcoins. This makes it an attractive target for any malicious party.

One of the main threats this component has faced stemmed from the lack of encryption in initial versions, this meant sensitive data was stored as plain-text and greatly facilitated data theft. This has since been fixed by adding an encryption option, but users may still need to opt-in to this feature. [BtcWiki:Weaknesses]

4.1.6 Comparison

Figure 15 shows a comparison of how each of the items explored in the previous sections affects each type of Bitcoin user.

<i>User \ Threat</i>	<i>Address Anonymity</i>	<i>Double Spending</i>	<i>Botnet Farming</i>	<i>Flooding</i>	<i>Software Errors</i>
<i>Miner</i>	!	✓	!	!	!
<i>Exchange Service</i>	!	!	✓	!	!
<i>Merchant</i>	!	!	✓	!	!
<i>User</i>	!	✓	✓	!	!



 *Not a threat*
 *Possible threat*

Figure 15: Technical Analysis Comparison

In this context miners are a subset of Bitcoin users who generate blocks. Exchange services represent the various services which allow user to exchange their Bitcoins for fiat currencies and vice versa. Merchants represent users who sell goods or services in exchange for BTCs. And finally, the user category represent everyday users who purchase goods and services with Bitcoins.

While it is expected that everyday users of Bitcoin will fall into different categories at different moments in time, for the purposes of this comparison that is not considered. It is assumed that each user occupies a single category and does not perform any actions associated with any other category.

Address Anonymity is mainly an issue to Bitcoin users, as a lack of anonymity could allow their purchasing habits and donations to be tracked by third parties. But this issue can also pose a risk to other categories of users, as an analysis of the block chain would allow an attacker to determine how many funds are currently associated with a certain address, and if owner of the address is known, it is possible to determine how many funds a miner, merchant or exchange service has available. This would allow an attacker to target a specific user or organization with a large amount of funds in order to attempt a theft of some kind.

The double spending attack is a significant threat mainly to merchants and exchange services and affects both in a similar way. This form of attack introduces a risk of a merchant selling goods or services in the belief that payment has been received, when in fact, it is later reversed.

This affects exchange services in a similar way when in the process of exchanging Bitcoins to another currency. The service may believe the Bitcoins are already in their possession and proceed with the payment. If the attack is successful, the service would lose both the Bitcoins and the currency that was traded for them.

However, the risk of this form of attack is mitigated by following the recommended precautions. The effectiveness of the attack is also reduced by the increase of computing power available on the network, as any attacker would need to control a larger amount of computing power to compete with honest nodes.

For the other categories of user, double spending is not such a significant threat, as users and miners do not receive payments, meaning that this form of attack is not applicable to them.

Botnet farms are a low risk issue for all types of user, with the highest affected group being miners. This is only due to an increase in 'competition' when mining Bitcoins, as a significant botnet would increase the network hashrate, leading to an increase in difficulty and reducing the effectiveness of the miner's machine and reducing profits. However, in reality, botnet farms are not a significant threat in this context. This is, in a large part, due to the inherent ineffectiveness of infected machines when farming bitcoins.

Of course there are other threats caused by botnets to the owners of the infected machines, including the installation of other malicious software or the theft of personal data and possibly Bitcoins. However, this form of attack is directed against the computer user, not the Bitcoin protocol, so it is not relevant in this context.

While it does not produce any profit to the attacker, the most effective form of attack discussed in this thesis is known as transaction flooding. This form of attack affects all categories of Bitcoin users.

Miners are affected by an increase in the required memory to store the pending transactions as well as an increase in the size of the blocks being transferred within the network.

All other types of user are mostly affected by a slow down of the entire system. Leading to an increase in transaction processing times and an increased risk of transactions being simply lost. However, in most cases this is merely an inconvenience and not a form of attack which causes significant damage.

Finally, software errors are perhaps the most significant threat against all users of the Bitcoin protocol. While there is no evidence that any bugs or back-doors have been exploited in the Bitcoin network, several bugs have been found (and corrected) since the creation of Bitcoin.

Depending on the nature of the particular bug, any category of user may be affected in a variety of ways. Exchange services and merchants may be severely affected if an exploit allows for altering payments after confirmation. While other bugs, such as a vulnerability in the wallet software, may be a bigger threat users and miners.

The active community of developers around Bitcoin have done a good job in finding and correcting this kind of vulnerabilities before they can be exploited by any malicious party. But it is possible that with a decrease in interest in Bitcoin the security provided by the openness of the protocol will also be reduced, as less developers will be actively inspecting and fixing the code.

4.2 Social Analysis

While the goal of this thesis is mainly a technical analysis, the popularity of Bitcoin has caused some important socio-economic repercussions in modern society.

This section is focussed on providing a short overview of some of the changes and social issues raised by Bitcoin and how they have affected society during the, relatively brief, history of Bitcoin. These issues are shown 'as is' merely as an effort to expose the reader to some of these issues, and not to determine if they are positive or negative or to find possible solutions.

4.2.1 Bitcoin Economics

Bitcoin is still in the early stages of economic development, and as such has not reached the popularity level of other services such as Paypal with respect to paying for goods and services. This is largely due to often being associated with illegal operations such as black markets and money laundering, and more recently, Bitcoin is also being associated with funding terrorism. [Fox, 2015]

While the number of users who have adopted the Bitcoin protocol is still growing, the currency can experience a large amount of volatility and large economic bubbles. This is easily demonstrated by the change in value the currency has experienced in the last few years, rising from around \$200/BTC to over \$1000/BTC over the course of a month in late 2013, and since returning to around \$300/BTC at the time of writing this thesis.

In some ways, Bitcoin is similar to a stock market, as large fluctuations in value can be caused by changes in how the public perceives Bitcoin. With value rising when the public have more trust in the system, and dropping when that trust is broken.

Another cause of these large fluctuations is that the currency is still in it's early stages, and some users still control very large amounts of currency. This means that choices made by these users will have a large effect on the overall currency. [Barker, 2014]

However, unlike fiat currencies, which can fluctuate in value as the inflation rate is changed or as the regulatory body introduces new currency into the system, Bitcoin does not allow for the arbitrary creation of new currency. A fixed number of Bitcoins will be mined, after which no more currency units can be generated, guaranteeing the scarcity of the currency. Even monetary systems based on precious metals cannot provide this, as a new stockpile of the resource may be found and reduce the it's value.

Some economists expect that as the currency matures and its adoption rate increases, resulting in increased trust, the level of volatility will decrease and the currency will stabilize. But other economists claim that Bitcoin is based on an old idea which has been dismissed by current economists and is no longer in use in today's society. As Bitcoins are in limited supply, the deflationary nature of Bitcoin will always cause large fluctuations in price and not allow for any regulatory body to control these fluctuations, leading to the currency to ultimately fail. [AP, 2013]

4.2.2 Bitcoin Legality

Bitcoin has often been associated with illegal activities due to the attractiveness of an anonymous currency to individuals or organizations partaking in illegal activities. And while the vast majority of Bitcoin users only using the system for legitimate activities, the privacy provided by Bitcoin has caused it to be used in black markets around the world as well as money laundering operations.

A notable example of this was a online site named Silk Road, which provided users the ability to purchase illegal narcotics in exchange for Bitcoins. The site has since been shut down by federal law enforcement. [Orsini, 2013]

Bitcoin has become an attractive currency for laundering funds and the trade of illegal goods due to the provided privacy and the lack of the same level of regulation experienced in other currencies. However, even with the anonymity provided by the Bitcoin protocol, as all transactions are recorded in a public ledger, law enforcement agencies could use statistical analysis tools to detect malicious activity and track down individual users. [Brito, 2011]

Due to the decentralized nature of Bitcoin, no one country can control and regulate the currency, causing government agencies tasked with preventing money laundering or black market sales to be concerned about this form of currency. This has lead to several different countries taking different stances on the legality of trading in Bitcoins. Several countries have introduced legal framework to attempt to regulate or restrict the the use of the currency, while others allow an unregulated use of the currency and no laws regarding Bitcoin have been put in place. A few countries, such as Bangladesh and Thailand, have attempted to completely ban the use of Bitcoin and consider any use of the system to be illegal. [Wiki:BtcLegality] [CoinDesk:Legality]

Another issue government agencies face when attempting to regulate the use of the Bitcoin currency is how to apply current tax laws to the new currency.

As with other regulatory laws, taxation laws also vary dramatically from nation to nation. With some countries not requiring users to declare their Bitcoin holdings for tax purposes, and others, such as the United States, where the Internal Revenue Service (IRS) has officially defined Bitcoin not a

currency, but as an asset, and thus requires users to declare their Bitcoin holdings and profits when paying taxes.

This legally means that each Bitcoin user in the United States must declare and pay taxes on any Bitcoin transaction and Bitcoin miners will also be required to pay a self-employment tax. However, many Bitcoin users are not very concerned about this new law, as they expect it will not be possible for the IRS to enforce such a law, and users will develop new methods in which to use Bitcoins without informing the IRS of their profits. [Doherty, 2014]

5 Conclusion

During this thesis several aspects of the Bitcoin protocol have been discussed, as well as some of the supporting protocols and data structures needed for the successful operation of the protocol.

At the end of this thesis, it is concluded that Bitcoin has been a very important development in the area of cryptocurrencies, being at the source of hundreds of new cryptocurrencies, some with improved security and privacy measures. And the open source nature of the Bitcoin protocol allows for improvements to be proposed, implemented and verified by members of the community. This ultimately results in a more secure and responsive protocol which can be improved over time to respond to new threats and social changes.

This thesis has approached some perceived vulnerabilities of the Bitcoin protocol, and while not all threats were discussed, it was determined that the protocol has sufficient security to remain in operation for the foreseeable future, assuming the community around Bitcoin remains active. And while it is not possible to determine how many people are actively using Bitcoin on a regular basis due to the pseudo-anonymous nature of the protocol, it is possible to determine the total number of transactions being made. And this value has been steadily increasing since the implementation of Bitcoin, which can be seen as an indication that the Bitcoin community is still growing. [BTC:TransactionNumber]

So while it is impossible to predict how important Bitcoin will become in the following years, it will always remain an important part of digital currency history as the first decentralized cryptocurrency to achieve large scale adoption and spawn hundreds of derivatives.

5.1 Was the goal reached?

The goal of this thesis was to provide the reader with a resource which would allow a better understanding of what the Bitcoin protocol is and how it functions, as well as the several components required for it to behave in a secure and stable enough manner to be used as a currency.

This goal has mostly been achieved. Chapter 2 contains information allowing the reader to understand what a cryptocurrency is, as well as historical information related to how Bitcoin came in to existence. Following this, chapter 3 contains a large amount of information which should help the reader gain a better understanding of the Bitcoin protocol and how its several components operate, as well as including information on the lower level protocols used by the Bitcoin protocol which the reader must understand in order to fully understand the protocol. And lastly, chapter 4 provides the reader with an analysis of some of the perceived weaknesses of the Bitcoin cryptocurrency and some theoretical attacks which may be used.

However, there were some topics related to the internal operation of the Bitcoin protocol which were not discussed, as disclosed in the introduction of the thesis, and more information could have been provided on the technical operation of network communication protocols used. While chapter 4 does include a sub-chapter related to network communication, this topic is sufficiently large to become the topic of a single thesis and only the most important information was provided.

The goal of the thesis also included discussing some of the social and legal impacts the introduction and widespread use of Bitcoin has introduced, and while this was discussed in section 4.2, this topic was not explored in any detail. This is mostly due to the background of the author which does not include any education in such topics. This decision was also influenced by the technical nature of the other chapters in the thesis.

5.2 Proposed improvements to Bitcoin

Since the initial description and implementation of the Bitcoin protocol several improvements have been suggested by several users of the system and can be viewed on their github page (<https://github.com/bitcoin/bips>). These proposed improvements are labelled BIPs (Bitcoin Improvement Proposal) and many of them have already been implemented in the current version of the code base.

The implementation of these improvements follow a pre-determined workflow going from drafts to finally being implemented. But the fate of these proposals ultimately rests with the consensus of the Bitcoin users. As any implemented improvement will need to be supported by the economic majority to be effective. [BtcWiki:EcoMajority] This means that the users using Bitcoin to trade goods and services can vote on which improvements get accepted or rejected.

Perhaps the most controversial improvement which is being currently debated is the proposal to increase the block size from 1MB to a higher value. This would allow more transactions to be included in each block, thus allowing Bitcoin to increase in capacity as more users start using Bitcoin. In the current state, the protocol can handle the number of transactions being made quite well, with the exception of cases of flooding. But it is expected that as the number of users, and subsequently transactions, increases, the current block size will not be sufficient. This will lead to a longer delay for valid transactions to be included in blocks, with some possibly not being included at all.

On the other side of the debate, users argue that increasing the block size too much will damage the decentralization of Bitcoin, by favouring miners with more capable hardware. As doubling the block size will in practice, double the amount of require data storage space available to miners. As smaller miners are forced to quit mining, more and more of the network CPU power is being controlled by fewer entities, leading to more centralization.

The current proposal is a gradual block size increase which will see the first increase to 8MB be made in early 2016. [Khaosan, 2015]

5.3 Future work

This thesis can be built upon in two main ways, either by providing a more in-depth description and analysis of the network architecture and communication protocols use in the Bitcoin system, as well as providing information on the different modes of operation Bitcoin allows. Alternatively, more information can be provided on the social, economic and legal aspects of the Bitcoin protocol.

A technical description on the network component of Bitcoin could include topics such as a description of all the various messages which the nodes can use, a more detailed description of how the peer-to-peer network is organized and how nodes intercommunicate. Information could also be provided on the differences between the full node and simple payment verification operation modes.

Other topics could include discussing the creation and use of contracts in the Bitcoin context, as well as an overview of different wallet solutions.

On the social, legal and economic side of the thesis, many more improvements could be made as many economists and lawyers find the concept of cryptocurrencies to be a complex issue with many facets.

Table List

Table 1: Glossary Items.....	6
Table 2: Bitcoin History [BtcWiki:History] [Wiki:Bitcoin].....	24
Table 3: Transaction data structure [Okupski, 2014].....	39
Table 4: nLockTime Values [Okupski, 2014].....	41
Table 5: Script Templates.....	44
Table 6: Hash Types.....	49
Table 7: Block structure [Okupski, 2014].....	55

List of Figures

Figure 1: Merkle Tree Structure.....	36
Figure 2: Simple Transaction Chain.....	40
Figure 3: Block Chaining.....	56
Figure 4: Merkle Tree Root Hash.....	57
Figure 5: Block Mining.....	60
Figure 6: Normal Block-Chain Fork.....	66
Figure 7: Hard Block-Chain Fork.....	67
Figure 8: Mining Pool Structure.....	70
Figure 9: Bank System vs. Bitcoin Network.....	73
Figure 10: Static Address.....	74
Figure 11: Bitcoin Exchange Server.....	75
Figure 12: Mixing Server.....	76
Figure 13: Double Spending Transaction.....	78
Figure 14: Double Spending Attack.....	79
Figure 15: Technical Analysis Comparison.....	86

References

[Wiki:DigitalCurrency] : Wikipedia, Digital Currency, 14.08.2015,
https://en.wikipedia.org/wiki/Digital_currency

[Wiki:Cryptocurrency] : Wikipedia, Cryptocurrency, 21.08.2015,
<https://en.wikipedia.org/wiki/Cryptocurrency>

[Economist, 2015] : The Economist, The Magic of Mining, 10.01.2015,
<http://www.economist.com/news/business/21638124-minting-digital-currency-has-become-big-ruthlessly-competitive-business-magic>

[Wiki:Chaum] : Wikipedia, David Chaum, 03.11.2015,
https://en.wikipedia.org/wiki/David_Chaum

[Wei-Dai, 1998] : Wei-Dai, B-Money, 1998, <http://www.weidai.com/bmoney.txt>

[Szabo, 2008] : Nick Szabo, Bit-Gold, 2008,
<http://unenumerated.blogspot.de/2005/12/bit-gold.html>

[Wiki:Szabo] : Wikipedia, Nick Szabo, 30.07.2015,
https://en.wikipedia.org/wiki/Nick_Szabo

[Nakamoto, 2008] : Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008, www.bitcoin.org

[Wiki:ListCryptocurrency] : Wikipedia, List of Cryptocurrencies, 09.08.2015,
https://en.wikipedia.org/wiki/List_of_cryptocurrencies

[Wiki:Litecoin] : Wikipedia, Litecoin, 06.08.2015,
<https://en.wikipedia.org/wiki/Litecoin>

[Wiki:DogeCoin] : Wikipedia, Dogecoin, 25.08.2015,
<https://en.wikipedia.org/wiki/Dogecoin>

[Wiki:PotCoin] : Wikipedia, PotCoin, 19.05.2015,
<https://en.wikipedia.org/wiki/PotCoin>

References

- [Wiki:Coinye] : Wikipedia, Coinye, 09.06.2015,
<https://en.wikipedia.org/wiki/Coinye>
- [Wiki:PeerCoin] : Wikipedia, Peercoin, 27.07.2015,
<https://en.wikipedia.org/wiki/Peercoin>
- [Wiki:Dash] : Wikipedia, Dash (Cryptocurrency), 08.07.2015,
https://en.wikipedia.org/wiki/Dash_%28cryptocurrency%29
- [Wiki:CryptoNote] : Wikipedia, CryptoNote, 26.08.2015,
<https://en.wikipedia.org/wiki/CryptoNote>
- [BtcWiki:Genesis] : Bitcoin.org, Genesis Block, 19.07.2015,
https://en.bitcoin.it/wiki/Genesis_block
- [Wired, 2013] : Wired.com, Take a tour of Robocoin, the world's first bitcoin ATM, 29.10.2013, http://www.wired.com/2013/10/bitcoin_atm_gallery/
- [BBC, 2014] : BBC News, MtGox bitcoin exchange files for bankruptcy, 28.02.2014, <http://www.bbc.com/news/technology-25233230>
- [Reuters, 06.01.2015] : Reuters.com, Bitcoin exchange Bitstamp suspends service after security breach, 06.01.2015,
<http://www.reuters.com/article/2015/01/06/us-bitstamp-cybersecurity-idUSKBN0KF0UH20150106>
- [Reuters, 09.01.2015] : Reuters.com, Bitcoin exchange Bitstamp says to resume trading on Friday, 09.01.2015,
<http://www.reuters.com/article/2015/01/09/bitstamp-cybersecurity-idUSL6N0UO1DC20150109>
- [BtcWiki:History] : Bitcoin.org, History, 2015, <https://en.bitcoin.it/wiki/History>
- [Wiki:Bitcoin] : Wikipedia, Bitcoin, 22.08.2015,
<https://en.wikipedia.org/wiki/Bitcoin>
- [CoinMarket, 2015] : coinmarketcap.com, Crypto-currency Market Capitalizations, 09.11.2015, <http://coinmarketcap.com/#EUR>
- [Evans, 2014] : Jon Evans, Enter the Blockchain: How Bitcoin Can Turn The Cloud Inside Out, 22.04.2014, <http://techcrunch.com/2014/03/22/enter-the-blockchain-how-bitcoin-can-turn-the-cloud-inside-out/>
- [Wiki:Namecoin] : Wikipedia, Namecoin, 23.10.2015,
<https://en.wikipedia.org/wiki/Namecoin>
- [woolci, 2015] : woolci, Which one you choose? PeerTracks or Ujo?, 12.09.2015, <https://bitsharestalk.org/index.php?topic=18402.0>
- [Wiki:HashFunction] : Wikipedia, Hash Function, 07.08.2015,
https://en.wikipedia.org/wiki/Hash_function

References

- [Wiki:CryptoHash] : Wikipedia, Cryptographic Hash Function, 28.08.2015,
https://en.wikipedia.org/wiki/Cryptographic_hash_function
- [Wiki:SHA] : Wikipedia, Secure Hash Algorithm, 06.04.2015,
https://en.wikipedia.org/wiki/Secure_Hash_Algorithm
- [Wiki:RIPEMD] : Wikipedia, RIPEMD, 06.11.2014,
<https://en.wikipedia.org/wiki/RIPEMD>
- [Jayanthi, 2015] : Jayanthi, Public Key Cryptography and PuTTYgen - Program for Generating Private and Public Keys, 2015,
<http://resources.infosecinstitute.com/public-key-cryptography-puttygen-program-generating-private-public-keys/>
- [Wiki:RSA] : Wikipedia, RSA (Cryptosystem), 18.09.2015,
[https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
- [Dwork et al., 1993] : Cynthia Dwork, Moni Naor, Pricing via Processing or Combating Junk Mail, 1993,
<http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp.ps>
- [Berman et al., 2005] : Piotr Berman, Marek Karpinski, Yakov Nekrich, Optimal trade-off for Merkle tree traversal, 28.10.2005,
<http://www.sciencedirect.com/science/article/pii/S0304397506008693>
- [Okupski, 2014] : Krzysztof Okupski, Bitcoin Developer Reference, 15.12.2014,
- [Btc:DevGuide] : bitcoin.org, Developer Guide, 25.10.2015,
<https://bitcoin.org/en/developer-guide>
- [BtcWiki:Script] : bitcoin.org, Script, 25.09.2015,
<https://en.bitcoin.it/wiki/Script>
- [BtcWiki:P2SH] : bitcoin.org, Pay to Script Hash, 27.05.2015,
https://en.bitcoin.it/wiki/Pay_to_script_hash
- [BtcWiki:Address] : Bitcoin.org, Address, 31.08.2015,
<https://en.bitcoin.it/wiki/Address>
- [Moore, 2013] : Chris Moore, StackExchange: How to calculate new bits value, 2013, <http://bitcoin.stackexchange.com/questions/2924/how-to-calculate-new-bits-value>
- [BtcWiki:BlockSize] : bitcoin.org, Block size limit controversy, 11.09.2015,
https://en.bitcoin.it/wiki/Block_size_limit_controversy
- [BtcWiki:Difficulty] : bitcoin.org, Difficulty, 13.07.2015,
<https://en.bitcoin.it/wiki/Difficulty>

References

- [Corallo, 2013] : Matt Corallo, [Bitcoin-development] [ANN] High-speed Bitcoin Relay Network, 05.11.2013, <https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03189.html>
- [Git:electrum] : spesmilo, electrum-server, 23.10.2015, <https://github.com/spesmilo/electrum-server/tree/8acd8eef4995db28b5c46a2986ac1b5a9e0a25ba>
- [Dev:Version] : bitcoin.org, Developer Reference - Version, 21.10.2015, <https://bitcoin.org/en/developer-reference#version>
- [Möser, 2013] : Malte Möser, Anonymity of Bitcoin Transactions, 2013,
- [BtcWiki:DoubleSpend] : bitcoin.org, Double Spending, 11.05.2015, <https://en.bitcoin.it/wiki/Double-spending>
- [Karame et al., 2012] : Ghassan O. Karame, Elli Androulaki, Srdjan Capkun, Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin, 2012, <http://eprint.iacr.org/2012/248.pdf>
- [Wiki:Botnet] : Wikipedia, Botnet, 6.11.2015, <https://en.wikipedia.org/wiki/Botnet>
- [Bradbury, 2013] : Danny Bradbury, Why ZeroAccess botnet stopped bitcoin mining, 02.10.2013, <http://www.coindesk.com/zeroaccess-botnet-stopped-bitcoin-mining/>
- [blockchain:hashrate] : Blockchain.info, Hash Rate, 08.11.2015, <https://blockchain.info/charts/hash-rate>
- [BtcWiki:Flood] : bitcoin.org, Flood Attack, 08.07.2015, https://en.bitcoin.it/wiki/Flood_attack
- [BtcWiki:JulyFlood] : bitcoin.org, July 2015 Flood Attack, 15.07.2015, https://en.bitcoin.it/wiki/July_2015_flood_attack
- [BtcWiki:Vulnerabilities] : bitcoin.org, Common Vulnerabilities and Exposures, 17.08.2015, https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures
- [BtcWiki:Weaknesses] : bitcoin.org, Weaknesses, 08.07.2015, <https://en.bitcoin.it/wiki/Weaknesses>
- [Fox, 2015] : Fox News, ISIS Parks it's cash in Bitcoin, experts say, 25.11.2015, <http://www.foxnews.com/tech/2015/11/25/isis-parks-its-cash-in-bitcoin-experts-say.html?intcmp=hpbt3>
- [Barker, 2014] : Jonathan Todd Barker, Why is Bitcoin's Value so Volatile?, 27.05.2014, <http://www.investopedia.com/articles/investing/052014/why-bitcoins-value-so-volatile.asp>

References

- [AP, 2013] : USA Today, Bitcoin economics: Primer on volatile currency, 11.04.2013, <http://www.usatoday.com/story/tech/2013/04/11/bitcoin-economics/2073517/>
- [Orsini, 2013] : Lauren Orsini, Bitcoin and the black market: The ties that bind, 08.10.2013, <http://readwrite.com/2013/10/08/bitcoin-probably-gets-its-value-from-illicit-use>
- [Brito, 2011] : Jerry Brito, Bitcoin, Silk Road and Lulzsec oh my!, 03.06.2011, <http://techliberation.com/2011/06/03/bitcoin-silk-road-and-lulzsec-oh-my/>
- [Wiki:BtcLegality] : Wikipedia, Legality of Bitcoin by Country, 09.11.2015, https://en.wikipedia.org/wiki/Legality_of_bitcoin_by_country
- [CoinDesk:Legality] : CoinDesk.com, Is Bitcoin Legal?, 19.08.2014, <http://www.coindesk.com/information/is-bitcoin-legal/>
- [Doherty, 2014] : Brian Doherty, Bitcoin and Taxes, 03.04.2014, <https://reason.com/archives/2014/04/03/bitcoin-and-taxes>
- [BTC:TransactionNumber] : blockchain.info, Number of Transactions per Day, 15.11.2015, https://blockchain.info/charts/n-transactions?timespan=2year&showDataPoints=false&daysAverageString=1&show_header=true&scale=0&address=
- [BtcWiki:EcoMajority] : bitcoin.org, Economic Majority, 24.09.2015, https://en.bitcoin.it/wiki/Economic_majority
- [Khaosan, 2015] : Venzen Khaosan, Bitcoin XT Block Size Increase: What is Proposed and how will it Affect the Bitcoin Price?, 23.06.2015, <https://www.cryptocoinsnews.com/bitcoin-xt-block-size-increase-proposed-will-affect-bitcoin-price/>

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____