NTNU
Norwegian University of
Science and Technology

Comparative Analysis of Bitcoin and Ethereum

Marit Rudlang

Master of Science in Communication Technology
Submission date: June 2017
Supervisor: Colin Alexander Boyd, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

Comparative Analysis of Bitcoin and Ethereum

Marit Rudlang

Submission date:	June 2017
Responsible professor:	Colin Boyd, IIK
Supervisor:	Chris Carr, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Comparative Analysis of Bitcoin and Ethereum
Student:	Marit Rudlang

Problem description:

The Bitcoin system was launched in 2009 and is widely recognized as the first successful attempt at a distributed cryptocurrency, with bitcoins being accepted as payment in a growing number of instances. Despite its popularity, central problems remain with the system design. These issues can be classified into three major groups: wastefulness of computational resources, tendency to centralization over time and ambiguity of transaction finalization.

Since its launch several alternatives to Bitcoin have been developed, with many trying to combat these issues. First among the alternatives, in terms of market value and popularity, is the Ethereum system. The developers claim that it provides a wider scope of functionality and higher levels of effectiveness compared to Bitcoin, all the while maintaining the same levels of security.

This thesis will provide a comparative analysis of Bitcoin and Ethereum with a focus on the three aforementioned issues. Our goal is to determine if Ethereum suffers from – or will suffer from – the same problems, and attempts to offer some insight into the future of this technology in general. The study will use the published technical descriptions of both systems as well as statistics from the live blockchains of each where appropriate.

Responsible professor:	Colin Boyd, IIK
Supervisor:	Chris Carr, IIK

Abstract

Since Bitcoin was launched in 2009, several new cryptocurrencies have been initiated with variations to Bitcoin's original design. Although Bitcoin still remains the most prominent actor in the market, some technical problems have been raised to the design of the protocol. The objective of this thesis is to determine whether the newer cryptocurrencies handle the technical problems of Bitcoin, or if they also suffer from the same issues. Instead of evaluating several cryptocurrencies for this comparison, the cryptocurrency Ethereum has been chosen as a proxy for the others. Ethereum was started in 2014, is widely backed in the community and is second in line to Bitcoin when it comes to market capitalization.

As a basis for the comparative analysis a rigorous study of the Bitcoin and Ethereum protocols have been performed, and parallel descriptions of the systems have been devised. Three technical problem have shaped the focus of the analysis: computational waste, concentration of power and ambiguity of transactions. Real world statistical data has been gathered and synthesized to enlighten the findings in the comparison. The main result of the comparison is that both systems suffer from the same problems to a certain degree, due to the fact that they utilize the same consensus mechanism. However, Ethereum utilizes several newer techniques to try and reduce the severity of these problems compared to Bitcoin, with varying degrees of success.

Sammendrag

Bitcoin ble startet i 2009, siden da har flere nye kryptovalutaer har blitt lansert i markedet med variasjoner til Bitcoin sitt originale design. Flere tekniske problemer har blitt senere påpekt ved Bitcoin protokollen. Til tross for disse problemene forblir Bitcoin den mest fremtredende aktøren på markedet. Formålet med denne oppgaven er gjennomføre en sammenlignende analyse for å avgjøre om disse nye kryptovalutaene håndterer de kjente tekniske problemene ved Bitcoin – eller om de også stever med de samme utfordringene. I stedet for å vurdere flere ulike kryptovalutaer, brukes kryptovalutaen Ethereum som en stedfortreder for de andre. Ethereum ble startet opp i 2014, og har siden det mottatt mye støtte fra miljøet, i tillegg til å være nest etter Bitcoin i markedsverdi.

Det har blitt gjennomført et grundig studie av både Bitcoin og Ethereums protokoller som en del av denne oppgaven, paralelle beskrivelser av systemene er presentert som et grunnlag for sammenligningen. Tre veldefinerte tekniske problemer for Bitcoin har formet fokuset for analysen: bortkastet beregningskraft, konsentrasjon av makt og tvetydighet i transaksjoner. Tilgjengelig data fra de aktuelle systemene har blitt samlet inn og komponert for å belyse funnene fra sammenligningen. Hovedresultatet fra oppgaven er at begge systemene lider av de samme tekniske problemene i varierende grad, grunnet at de begge anvender den samme typen konsensusmekanisme. Ethereum bruker imidlertid flere nye teknikker for å forsøke å redusere effekten av problemene sammenlignet med Bitcoin, med varierende grad av suksess.

Preface

This thesis is submitted at the Department of Information Security and Communication Technology at Norwegian University of Science and Technology (NTNU). The thesis constitutes the final project for the MSc program in Communication Technology with specialization in Information Security. The duration of the study has been 20 weeks, performed in the Spring of 2017.

I would like to thank my supervisor Chris Carr and responsible professor Colin Boyd for our weekly meetings which has been a source of consistency and support throughout the duration of the thesis. They have both always been generous with constructive feedback and valuable advice for the thesis.

I would also like to thank my father and brother for valuable insights and discussion when proof-reading, and a special thanks to Tuva Dybedokken for staying with me until the very end and reading every word in this thesis.

Finally, I would like to thank my good friends in the Drama office, for good times and loving support these last five years.

Trondheim, 12th of June 2017. Marit Rudlang

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Traditionally, financial systems are based on physical money and digital credit. In the world of online transactions the only way of exchanging value is by utilizing trusted third parties, such as banks or intermediate payment systems – for instance, PayPal [1] – to relay the transaction. A weakness of these kinds of online systems is that you have to trust the mediating third party to act in your interest. Even fiat currencies, i.e. the physical money system, have their weaknesses – trust needs to be placed in the institution issuing such currencies, that they will not act in ways that may cause unreasonable levels of inflation or financial crises.

Cryptocurrencies provide an alternate solution to the existing systems. By utilizing a peer-to-peer transaction system – where identities and ownership can be validated using cryptographic means – the users of the system do not have to rely on trust in third parties to exchange value online. Most cryptocurrencies mint their own coin to create value within the system. This is an essential part of their functionality that decouples the value in the cryptocurrency from any fiat currency that relies on governments or organizations for issuance. It also means that cryptocurrencies can be exchanged globally – independently of what currency is native to a region.

When Bitcoin [Nak08] emerged in 2009 it was just a small group of early adopters that saw the potential in the technology and made use of it [NBF$^+$16]. Since then a growing ecosystem of different cryptocurrencies has emerged. It becomes increasingly interesting to notice the differences and challenges of the several systems as they grow in popularity and value.

1.1 Previous work

Although Bitcoin is considered the first *successful* attempt at creating a digital currency, it is not the first attempt. The idea has been around since the early

[1] www.paypal.com

nineties, with several variations and implementations. One of the earliest commercial attempts at a digital currency was a company called DigiCash, founded by Chaum [NBF+16]. DigiCash uses protocols building on a cryptographic scheme Chaum had developed – and patented – that allowed user-to-merchant transactions to be verified anonymously. However, transactions between users were not possible. DigiCash was implemented by some merchants and banks [Cha92], but did not gain enough traction commercially to survive in the market in the long run [NBF+16].

Two other notable systems are Dai's b-money [Dai98] and Back's Hashcash [B+02]. These systems are notable because they introduce key components of the Bitcoin system, and are referenced by Nakamoto in the original Bitcoin paper. The main ideas from these systems form the foundation of the *blockchain* and the *consensus algorithm* for Bitcoin.

Dai[Dai98] discusses the general idea of a digital cash system – b-money – where transactions are being broadcast to everyone, and the users of the system are responsible for verification and keeping track of balances in a distributed manner. The system Dai describes is a general idea – not an actual implementation – but this might have sparked the idea of Bitcoin's blockchain, as it is referenced by Nakamoto in the original Bitcoin paper.

Back [B+02] introduced Hashcash, and implemented a computational proof of work – called a cost function – for a service. The original implementation of this idea was a throttling mechanism for e-mail spam. A similar idea had been presented earlier by Dwork and Naor [DN92]. By including a computationally costly proof of work in an email, the receiver of the mail could be reasonably sure that the email had not been generated on a large scale – like spam emails. Other uses of this cost function were also discussed, for example, as a tool for mitigating SYN-flooding attacks on web servers; a common denial of service attack targeting the TCP protocol [SKR+11]. This proof of work principle is used as a key component in Bitcoin, and as we will see later, is vital for the system functionality.

1.2 Research Objective

Over the years, Bitcoin has been thoroughly reviewed, and technical issues for the system have been identified, Micali [Mic16] summarizes them as three technical problems:

Technical Problem 1: Computational Waste Large amounts of computation and energy is wasted in the validation process of Bitcoin. This is because nodes – i.e. Bitcoin system participants – are investing in expensive hardware to get an advantage when competing to receive rewards for validating the transactions of peers. Combined with the dynamic adjustment of the difficulty of the validation process to fit the expected time of ten minutes between each set of transactions, it creates a situation where unnecessarily large amounts of computation are expended.

Technical Problem 2: Concentration of Power The validation is concentrated in a few centrally organized groups, and not spread out in the distributed manner as it was intended. For the validation process to be profitable for nodes in the peer-to-peer network regarding costs related to the process, nodes join together and split rewards received from successful validation. In a group like this, the cost of specialized validation equipment and power expenses is split between the nodes in the group, and any reward for a successful validation is shared with the others. This defeats the original purpose of Bitcoin, which is to be distributed.

Technical Problem 3: Ambiguity Because of the underlying architecture of Bitcoin, transactions take time to process. Knowing exactly when a transaction can be trusted or not is subject to variance, causing the validity of the transactions to remain ambiguous for a period of time after they have been relayed to the network.

The main research objective of this thesis is to determine if the later cryptocurrencies have avoided the known problems of Bitcoin, by performing a comparative analysis of Bitcoin and Ethereum.

Newer cryptocurrencies have the advantage of the knowledge of the problems of Bitcoin before development; comparing these cryptocurrencies to Bitcoin therefore seems like the natural choice when reviewing how well these challenges are being met and handled by the newcomers. Rather than try to answer this by examining multiple instances of cryptocurrencies the main focus has been put on Ethereum.

There are several reasons for choosing Ethereum as a proxy for the other cryptocurrencies. Firstly, the currency is relatively recent, Ethereum was launched the summer of 2015, the developers of the system will likely be aware of the current research in the field. Secondly, the system is explicitly aimed at fixing the shortcomings of Bitcoin [Whaa], which means that we can assume they have had these problems in mind when designing the protocol. Lastly, although Bitcoin is still the biggest of the cryptocurrencies in terms of adoption and value, Ethereum comes next in line as the second most popular alternative and is widely backed in the community. To give a quantifiable measure of the two systems, the website *coinmarketcap.com* values the current (30. May 2017) Bitcoin market capitalization as being over 32 billion US dollars. Ethereum comes next with over 20 billion US dollars [Cry]. However, it should be noted that these values are highly volatile and subject to change on a daily basis.

1.3 Methodology and Outline

As a methodology for this thesis consists of deriving parallel technical descriptions for Bitcoin and Ethereum to detail both systems in a scientifically rigorous manner to accurately compare and contrast them. Importantly, while Bitcoin has recently been described well in the literature, information on Ethereum is dispersed over online wikis, blogs, and forums, subject to edits and changes by the different authors sporadically – making the task of detailing Ethereum non-trivial.

Furthermore, the technical problems of Bitcoin have been elaborated in terms of the protocol detail, and Ethereum has been analyzed to evaluate whether it exhibits the same problems. The two descriptions are placed in contrast to each other, highlighting their similarities and differences. The comparative analysis will also include data available from the live systems when this is appropriate. This data has been gathered and synthesized to enlighten the discussion and to determine if the findings from the comparison align with the data.

An outline of the contents of the thesis is included below.

Chapter 2: Theory Cryptographic primitives, data structures and other technical aspects that are essential to both Bitcoin and Ethereum.

Chapter 3: Bitcoin An in-depth description of the Bitcoin protocol and technical features.

Chapter 4: Ethereum An in-depth description of the Ethereum protocol and technical details as a contrast to Bitcoin functionality.

Chapter 5: Comparison and Discussion From what we have learned in Chapter 3 and 4; a comparative analysis of Ethereum and Bitcoin in terms of the technical problems discussed, enlightened by the statistical data gathered.

Chapter 6: Conclusion Summary and concluding remarks.

Chapter 2 Theory

This chapter discusses fundamental cryptographic primitives and schemes, as well as specific data structures and their properties that are used both within Bitcoin and Ethereum.

2.1 Cryptographic Primitives

Cryptographic primitives are the building blocks of any secure information system. For distributed cryptocurrencies, the cryptographic primitives are essential for the functionality of the currency. A cryptographic primitive is an algorithm – usually following a standard established by a trusted institution – that performs a cryptographic function. This section describes the functionality of the relevant cryptographic primitives for Bitcoin and Ethereum.

2.1.1 Hash Functions

Hash functions, in general, take data of any length as input and transform the data according to a deterministic mathematical function h. Depending on what kind of properties that are desirable for the output, there exist a variety of different families of hash functions to suit most applications. For instance, a hash function to be used for indexing data in an array will need to produce a fixed length output from the input data that can be used indicate where the data is stored. Utilizing hash functions for indexing is a typical operation, and such hash functions can be constructed simply. A cryptographic hash function, on the other hand, has higher requirements.

Cryptographic Hash Functions

Stinson [Sti05] formally defines a hash function $h : X \mapsto Y$ like so: Let $x \in X$, and define $y = h(x)$. For a cryptographic hash function it should only be possible to calculate y by applying function h to x.

A secure hash function must be resistant to the following three properties, i.e. the following three problems should be difficult to solve for the given hash function h [Sti05].

Preimage Resistance For hash function h and an element $y \in Y$ it should be infeasible to find $x \in X$ such that $h(x) = y$. This property is also known as the *one-way property.*

Second Preimage Resistance For hash function h and an element $y \in Y$ it should be infeasible to find $x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$.

Collision Resistance For hash function h it should be infeasible to find $x, x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$.

In addition to these three, other typical properties for cryptographic hash functions are listed next. These are – as we will see later – especially interesting in the context of distributed cryptocurrencies.

Uniform hashing distribution The hashing distribution should be uniformly distributed. The distribution of hash values should be approximately the same as selecting a random value from the output space. Hash functions that have this property can be utilized as pseudo-random number generators because the output will be difficult to distinguish from random.

Efficiency Computing $h(x)$ should be a computationally efficient operation. This is true for most cryptographic hashes, but we will see that for some scenarios, hash functions that are deliberately slow are more appropriate.

Output size The size of the output must be of such a length that it is computationally infeasible to brute force a pre-image of a hash. That is if the output size is too small a list of possible input/output pairs can be generated to find pre-images, which breaks the first requirement for cryptographic hashes.

Designing secure hash functions is far from easy, and through the years there has been a slow evolution of cryptographic hashing functions as flaws or weaknesses in previous functions have been found [HS05]. Because of the difficulty in designing secure hash functions, there are several standards of hash functions. The National Institute of Standards and Technology (NIST) is a measurement laboratory founded in the US [NIS]. NIST, among other things, is responsible for the standardization of cryptographic algorithms – including the most commonly used cryptographic hash algorithms. Table 2.1 gives a general overview of this evolution up till today. The first four algorithms were developed by NSA, while SHA-3, was the winner of an

Table 2.1: Evolution of Standardized Hash Functions

Family	**Year**	**Functions**	**Output size**	**Deprecated**
MDx [Riv92]	1990's	MD2, MD4, MD5	128-bit	*Deprecated*
SHA (SHA-0) [PUB95]	1993	SHA-0	160-bit	*Deprecated*
SHA-1 [PUB95]	1995	SHA-1 (fix on SHA-0)	160-bit	*Deprecated*
SHA-2 [Dan13]	2001	SHA-256, SHA-512 (plus truncated versions)	224-bit, 512-bit (plus truncated versions)	
SHA3 [PUB14]	2012	SHA3-224, SHA3-256, SHA3-384, SHA3-512	224-bit, 256-bit, 384-bit, 512-bit	

international competition – the hash function was originally named Keccak. SHA3 is the standardized version of Keccak, and they differ slightly in implementation.

2.1.2 Hash Chains

The general idea of a hash chain is that by consecutively applying a one-way hash function to some data, the same data can be used as a secret over several passes without revealing the content. An example of a hash chain of size four looks like this:

$$h(h(h(h(\text{data})))) = h^4(\text{data})$$

Practical applications of hash chains are Lamport's one-time password [Lam81]. This scheme allows the same shared secret to be reused consecutively by using is as a root in a hash chain. Once the chain has been generated a new one-time password is determined by starting at the end of the chain, and moving backward. Once a password has been used it is removed from the chain and is never used again, in this way no information is disclosed by knowing previous passwords. Additionally, in 1997 Rivest and Shamir [RS97] proposed a micropayment system utilizing hash chains to verify payments.

An adapted form of hash chains can also be used as a secure timestamp server for data, Haber and Stornetta [HS91] discusses this use of hash chains. In secure timestamp servers, more data can be added to the hash chain over time, creating an immutable structure of the previous data. Figure 2.1 shows how such a use can be structured, where T denotes a timestamp, H a hash value. By evaluating the hash value in the chain assurance that the previous timestamps have not been tampered with is provided, because changing a timestamp would require finding collisions for all

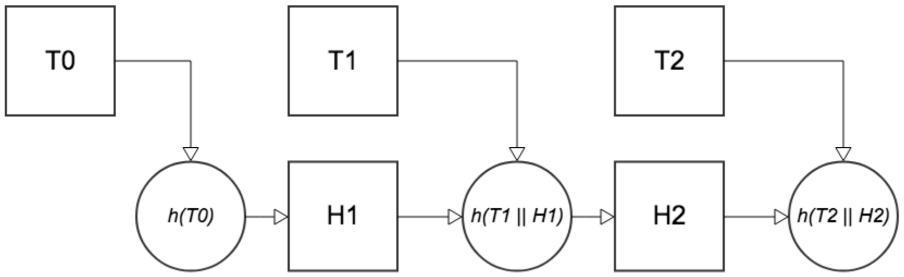


Figure 2.1: An example of a hash chain with timestamping.

the subsequent hashes. Other data could also be included along with the timestamps, creating assurance of the creation time and integrity of the data.

.

2.1.3 Digital Signatures

In the world of cryptography there are two paradigms: *symmetric* and *asymmetric key cryptography* [DH76]. Symmetric key cryptography is based on the idea that the communicating parties have a shared secret, a *key*, that can be used to secure communication. Symmetric key cryptography is efficient but based on trust between the communicating parties that neither will disclose the key to others. Asymmetric cryptography removes this need for trust in the other party by providing keys in pairs: *public* and *private keys.* The private key is secret and only known to the owner of the key, while the public key is publicly announced to anyone. The keys are related to each other in such a way that the public key can encrypt, the private key can decrypt – the public key alone will not enable decryption. Asymmetric cryptography is commonly called public key cryptography.

Public key cryptography is based on different domains of mathematical functions, the second column in Table 2.2 lists some of these. Typically, the functions are based on a mathematical problem that is easy to solve when given all the relevant information, but computationally hard to solve with only parts of it. In terms of cryptography the principal part of that information will usually constitute the secret private key.

Digital signatures have their foundation in the public key cryptography: Public keys can be used authenticate digital signatures, while private keys can be used to digitally sign outgoing messages. A good digital signature will provide assurance of a message's authenticity and integrity for anyone holding the sender's public key, without disclosing any information about the related private key. Several different digital signature algorithms exist. NIST specifies the DSS [Gal13], a collection of three standardised digital signature algorithms. Table 2.2 gives an overview of these, their relevant parameters and mathematical basis. All digital signature schemes define three algorithms:

Key generation algorithm The key generation algorithm is the initial phase. This step typically takes a random number of a fixed size – the private key – and performs a mathematical function on the random number to produce the public key.

Signature algorithm The signing algorithm takes as input the message to be signed and the signer's private key and outputs a signature that can be appended to the original message.

Verification algorithm The verification algorithm takes three variables as input: a message, the signature appended to the message and the sender's public key. The algorithm outputs *true*, if both the signature matches the sender's public key and the message has not been tampered with, *false* otherwise.

The security in a digital signature relies on the hardness of the underlying mathematical problem. In addition to the details of the digital signature algorithms, DSS defines rigid guidelines on the size of the domain parameters and keys for implementations. Because signatures are based on different mathematical problems, the recommended parameter length differs between the different algorithms. For instance Elliptic Curve Digital Signature Algorithm (ECDSA) has a significantly smaller key size than Digital Signature Algorithm (DSA) for the same level of security [BR12].

2.2 Data Structures

A data structure is a method of organizing data, usually with the goal of structuring them such that operations can be performed efficiently on the data. There exists a variety of different data structures – designed for different purposes – where each has its advantages and disadvantages. The data structure most suited to your system depends on what kind of operations you wish to perform on the data subsequently. Primitive data structures are for instance arrays, trees, stacks, and queues. Some specific data structures are used in Bitcoin and Ethereum; the next section will give the reader an overview of these.

Table 2.2: Overview of the three algorithms specified in the DSS [Gal13]

Algorithm	Mathematical basis	Domain parameters	Private key	Public key	Signature
Digital Signature Algorithm (DSA)	Discrete logarithm problem	Per message secret k, primes p and q, generator g	x	y	(r, s)
Rivest-Shamir-Adleman-DSA (RSA-DSA)	Integer factorization problem	Modulus n	n, d	n, e	s
Elliptic Curve DSA (ECDSA)	Discrete logarithm problem for elliptical curves	per message secret k, curve parameters, base point G, order n of base point G, cofactor h	d	Q	(r, s)

2.2.1 Merkle Trees

A Merkle tree [Mer82] is a binary tree, where the leaf nodes are hashes of original data, and all parent nodes are hashes of its two children. This is similar to a hash chain, except that for each new hash data is combined with the next node on the same level, halving the number of hashes for each level in the tree. The root of the tree is called the Merkle root or root hash of the tree. The root hash represents a hash of all the data combined in the tree. Figure 2.2 shows how each parent node is assembled and hashed from the data in the child nodes. In the figure $h(x)$ denotes a hash function. The yellow fields show the original values as leaf nodes, the blue box – the root of the tree – is called the *root hash*, and represents the hash of all the values stored in the tree.

The Merkle tree is efficient when checking the integrity of data in the leaf nodes. By storing the data in a tree structure, a validator only needs to check one branch of the tree to validate the data in the leaf node. This means that for any dataset of size n the lookup time for data is logarithmic.

.

2.2.2 Patricia Tree

A Patricia tree is a binary radix tree [Mor68]. The tree is constructed by dividing data according to common prefix and storing each prefix only once. A binary Patricia tree means that each parent node may have at most have two children. Data stored in a Patricia tree is read by accumulating data from traversing the branches of the tree from beginning to the end node. Figure 2.3 shows a simple implementation of a Patricia tree, where each new node adds to the final value. The tree is constructed from four data values, which can be seen in the box to the left. The yellow nodes

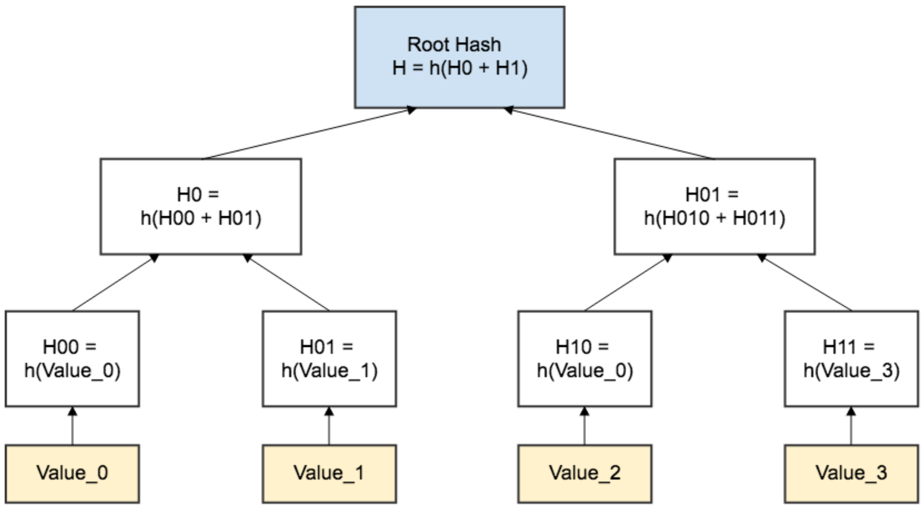


Figure 2.2: A block diagram showing the structure of a Merkle tree.

show how the tree can be traversed to find the data value "ape". The dollar sign denotes a terminating node.

2.2.3 Merkle Patricia Trees

The Merkle Patricia Tree is a data structure that is used extensively in Ethereum. The structure is a combination of the functionality of a classic Merkle tree such as the one described in Section 2.2.1, and a Patricia tree in Section 2.2.2. This results in a data structure with the same properties for integrity checking data stored in the structure that is gained with the Merkle tree, but with optimizations for storage, insertion, and deletion from the Patricia tree.

The Merkle Patricia Tree is specified by the developers of Ethereum [Woo14] [Pat]. The Design Rationale document [Des] states that the design is influenced by original thoughts from the Bitcoin forum [Ult]. The properties of the Merkle Patricia Tree are listed below:

Logarithmic time Lookups, updates, deletions, and insertions are done in logarithmic time.

Bounded tree depth Unlike the regular Merkle Tree there is an upper limit to the depth of the Merkle Patricia tree that is independent of the values stored

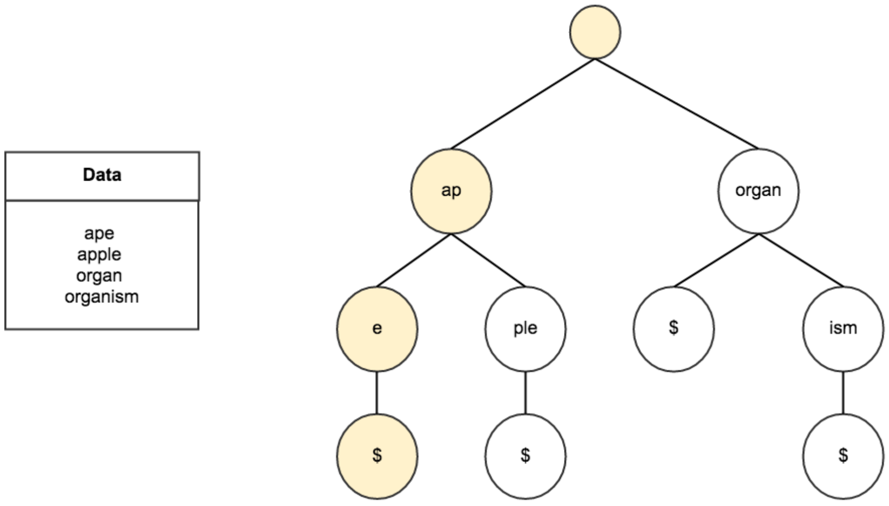


Figure 2.3: A simple Patricia tree.

in the tree.

Unique root-mapping The root of the tree is uniquely mapped to the data stored in the tree, i.e., changing any value in the tree influences the value of the root. Unlike the Merkle Tree, the order of insertion is irrelevant to the computation of the root.

All data added to the Merkle Patricia Tree is hashed and indexed in the format [key, value]. The key is the hash of the data, and the value is the actual data. The keys become the traversal keys – that works similarly to the Patricia tree – the data values are stored wholly in the leaves of the tree. This works by the use of three types of nodes:

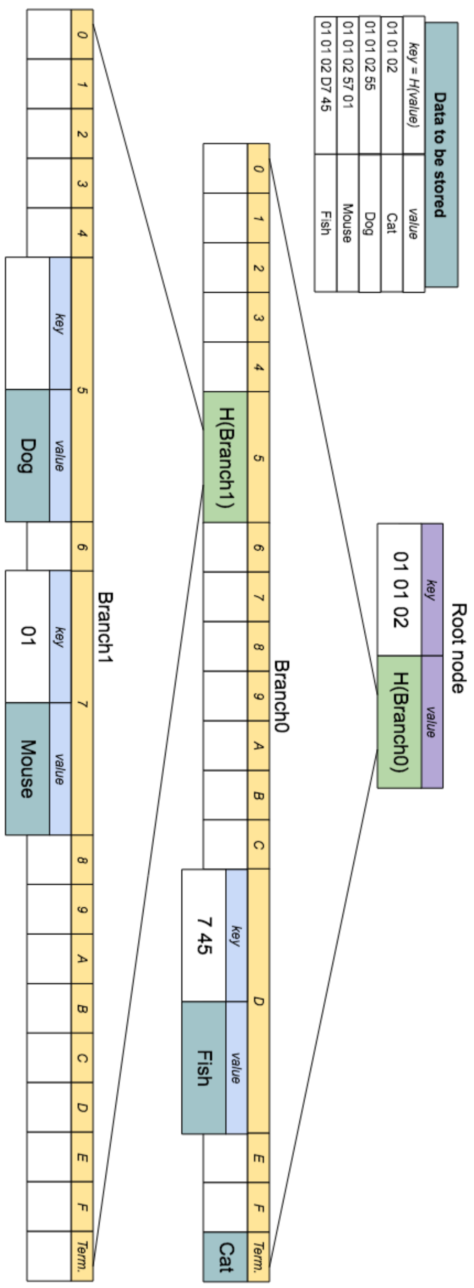
The [key, value] node is the standard leaf node, and stores the referenced data. The value is the data, and the key is the remainder of the traversal key (if there is a remainder). The traversal key is the hash of the data value stored in the leaf node. The hash provides integrity for the data value and prevents tampering with the data in the tree as long as the root hash remains intact.

An extension node is an intermediary node that holds intermediary key fragments that are shared by more than one child leaf node. An extension node is always followed up by a branch node. The extension node is also in the format [key, value] where the key represents the intermediary fragment the key traverses, and the value holds a hash of the child branch node. This hash has two functions: firstly, to reference the next node to look up, and secondly to provide integrity for the child node.

The branch node is an array of size 17, where each position in the array references one hex encoded character in the key of a branching leaf node. The 17th position references a terminating leaf node value. If the branch has a child that is not a leaf node, i.e. it branches to either an extension node or another branch node – as is often the case – the hash of the child is stored in the array. This hash provides the same two properties as the hashes in the extension node: referencing the child and providing integrity for the child.

Figure 2.4 visualizes the concept with four data values. The blue nodes are regular [key, value] nodes, the purple node is an extension node, and the yellow nodes are branch nodes. For the sake of illustration, the keys have been set to simple hex values that illustrate the properties of the different node types. Note that in a real Merkle Patricia Tree the keys are created by cryptographically hashing the data values that are stored, and this is essential to the security of the tree.

The root hash will always be an accumulation of the hashes of all the extension and branch node hashes. The extension and branch nodes only exist on the basis of the leaf nodes which contain the actual data values, and they are constructed from the keys of the data values. Since the keys are hashes of the data values the value in a leaf node cannot be changed without altering the entire structure and corresponding root node. As such the hashes work as cryptographic integrity checks, and as hash pointers to other nodes.



Data to be stored

key = H(value)	*value*
01 01 02	Cat
01 01 02 55	Dog
01 01 02 57 01	Mouse
01 01 02 D7 45	Fish

Figure 2.4: A Merkle Patricia tree containing four data values.

Chapter 3 Bitcoin

Bitcoin's original intent – as it is described by Nakamoto [Nak08] – was to be a peer-to-peer *electronic cash* system, in other words a system where nodes can exchange value in a distributed manner without relying on trusted third parties to forward transactions. Today we know Bitcoin as a *decentralized cryptocurrency.* No central authority distributes the currency or validates the transactions. That responsibility falls solely to the users of the system. To operate correctly, thousands of users need to agree on all transactions happening everywhere at the same time. This can be a challenge, but Bitcoin is living proof that it is practically possible. This chapter will start by defining some terms used in Bitcoin, followed by a detailed description of the functional aspects of Bitcoin. Finally, a discussion about the current state of the Bitcoin system and security considerations.

3.1 Definitions

We begin this chapter by discussing some of the terms used to describe the original intents of Bitcoin: value, trust, and distributed consensus.

3.1.1 What is Value?

Bitcoin has its own coin that is *valuable*, the currency is created as a part of the system and is only tradeable within the system. The Merriam-Webster dictionary defines value:

Definition 3.1. Value *"a fair return or equivalent in goods, services, or money for something exchanged."*

For something to have value, there needs to be a consensus that it is possible for it to be exchanged in return for goods, services or money. If someone is willing to trade for the item in question, it has value.

In itself the currency minted in Bitcoin is useless, but the moment someone is willing to trade a coin in the system in return of some good, service or money in the physical world, the coins get real world value. To illustrate this fact, the first Bitcoin transaction in return for an actual service was a payment for pizza in 2010. The pizza was bought for the neat price of 10,000 bitcoins [1] – a value roughly translating to 22 million USD at the current exchange rate.

The main purpose of the Bitcoin protocol is to enforce the general rules of exchanging value, that we recognize from the physical world. They can be summarized like so:

Authenticated transactions You can only spend value that is yours.

No Double Spends You can only spend value once - once it is exchanged, it belongs to someone else.

The Bitcoin system is distributed, decentralized and does not rely on any trusted third party to function – it is sometimes referred to as *trustless* because of this property [NBF+16].

3.1.2 What is Trust?

The Merriam-Webster dictionary defines trust as

Definition 3.2. Trust

(a) "Assured reliance on the character, ability, strength, or truth of someone or something.

(b) One in which confidence is placed."

In Bitcoin confidence is placed in the mathematical problems underlying the cryptography, allowing nodes with no preconditions to interact in a trustworthy manner.

3.1.3 Distributed Consensus Problem

For a cryptocurrency to be distributed it has to find a solution to the distributed consensus problem. The distributed consensus problem is a classic problem from the field of computer science. Narayan et al. [NBF+16] defines it like so:

[1] https://bitcointalk.org/index.php?topic=137.0

Definition 3.3. Distributed Consensus Problem: *"There are n nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:*

- *It must terminate with all honest nodes in agreement on the value.*
- *The value must have been generated by an honest node."*

The distributed consensus problem is not a trivial problem to solve. In fact, two known impossibility proofs: The Byzantine Generals problem [LSP82] and Fisher-Lynch-Paterson [FLP85], prove the impossibility of this problem for certain models.

3.2 What is Bitcoin?

The Bitcoin system itself is effectively a set of rules – a protocol – that all the nodes in a peer-to-peer network have to abide by. The rules of the protocol are enforced by using techniques from cryptography. These rules allow the users of the system to validate the legitimacy and ownership of the cash that is being spent in transactions without having to trust the source of the transaction data. If anyone tries to deviate from the protocol by including transactions that are illegal – e.g. double spending – the other users in the network can detect illegal transactions by checking the cryptographic proofs. If the checks return false, the transaction will be discarded, and it will be as if it never happened. As long as more than half of the users that are validating and agreeing on the transactions are acting by the rules, one can be assured that the system is operating as desired [Nak08].

The original white paper [Nak08] by Satoshi Nakamoto describes the functionality of the system on a high level. However, there is a score of subtleties of the system that Nakamoto's paper does not cover. The reference point for users that want to take part in the network is, therefore, a canonical implementation of a computer program called *bitcoind* – or the bitcoin daemon. Interestingly, the name Satoshi Nakamoto is a pseudonym for the unknown actor who designed Bitcoin. The true identity of Nakamoto remains unknown to the public to this day [NBF^{+}16]. For the first few years after deploying Bitcoin, Nakamoto was still participating in mailing lists and forums discussing the functionality and future of Bitcoin. However, after the system became increasingly popular, Nakamoto went underground and has not been a part of the development of the system since – the reasons for this remain unknown.

3.2.1 bitcoind

The daemon program *bitcoind* runs as a background process. Each instance of this daemon is called a Bitcoin node and is a participant in the network. As mentioned, the first version of bitcoind was created by Nakamoto in 2009 [NBF$^+$16]. Over the years – after Nakamoto disappeared – the task of updating and maintaining the daemon has fallen to the Bitcoin Core developers [Bita]. Documentation for the technical details are available through a wiki page [Bitb] which is maintained by the community.

The community can take part in the process of improving Bitcoin through a so-called Bitcoin Improvement Proposals (BIPs). By describing functionality or ideas for Bitcoin in BIPs they can be included in new versions of the protocol. However, it is up to the nodes in the system if they want to accept these changes or not – should a change be introduced that a significant part of the network does not want to accept, the system could *fork* causing different versions of the system to exist at the same time, and potentially having severe consequences for the system. Because of this risk, there are two classes of updates for the protocol:

Soft Fork A soft fork is a change in the protocol that enforces stricter rules than the ones already in existence. Anyone who still abides by the old rules will therefore still accept the data that have been produced in accordance with the protocol, but data created by people that have not updated their version may not be accepted by everyone.

Hard Fork A hard fork is a change in the protocol that changes some rule completely. That means that any data produced in accordance with the new protocol would be illegal in the old protocol, and vice versa.

Because of the risk of network splits, changes to the protocol are not always easy to implement because they require a majority to agree to the change. Changes that require hard forks are especially difficult because they effectively split the network until everyone has updated their program. An unforeseen effect of this is that the implementation of Bitcoin that is in use today still has a lot of quirks from the original implementation that are hard to fix, because of the risk related to the process of forking the blockchain. An example of this is a bug in one of the scripts, CHECKMULTISIG, that unexpectedly removes a data value every time it executes – making it so programmers have to add a dummy value into the script in the correct place for it to run correctly. The Bitcoin wiki [2] contains an extensive list of "protocol housekeeping" and bug fixes that are not changed for this reason.

[2]https://en.bitcoin.it/wiki/Hardfork_Wishlist

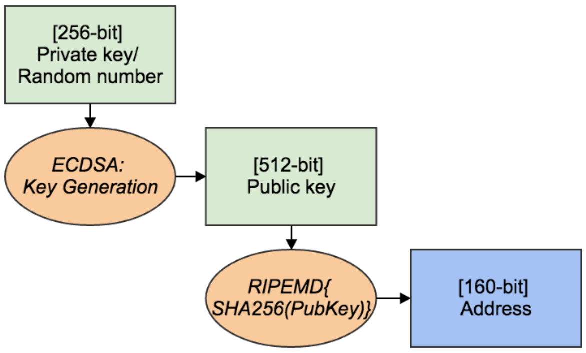


Figure 3.1: Diagram showing the relation between keys and addresses in Bitcoin.

3.3 Addresses and Keys

In Bitcoin, value is related to an *address*. A new address can be generated by anyone by executing the ECDSA (see Section 2.1.3) key generation, producing a public and private key pair. Figure 3.1 shows this process. In the figure, the squares represent data, and the circles represent actions. The green boxes are keys; the blue boxes are addresses. The ECDSA domain parameters are the same for all ECDSA signatures. The curve used in ECDSA is called Secp256k1 and is not a NIST standard, but defined by Standards for Efficient Cryptography (SEC) [Qu99]. The address is a number that is derived from the ECDSA public key by hashing the key twice, using first SHA256 and then *RIPEMD*. RIPEMD is another European standard hashing algorithm, producing a 160-bit output [DBP96]. By holding the private key that is related to an address you control the value. The address can then be published in order for other users to transfer value to your address in exchange for goods and services, much in the same way as when paying for it with regular cash.

The design rationale for choosing these specific cryptographic primitives in the Bitcoin implementation are not known, but because they are fundamental to the security of the system, we can assume they have not been chosen lightly. Possible justifications to utilizing hashing twice may be in the case of future vulnerabilities becoming evident in the hashing functions, and attempting to mitigate some information leakage [GCR16]. By utilizing two hashing functions, the possibility of both breaking at the same time is less probable. Additionally, the RIPEMD hashing function output is of size 160-bit, requiring less storage than SHA256. An influencing reason for choosing a non-NISTs standard for the curve parameters may be that NIST has earlier been accused of designing an intentional trap-door in one of their cryptographic standards [BLN16].

3.4 Transactions

To transfer value between two addresses a transaction must be generated. Each transaction has at least one input and one output. An input is a reference to an output from a previous transaction that *links a value to an address.* Section 3.4.1 discusses the details of this referencing scheme. In Bitcoin, there is no store of value that is yours, like you would have an account of money in your bank. Rather, the value you own is the sum of all Unspent Transaction Output (UTXO) to addresses that you control the private keys to. Nakamoto [Nak08] defines UTXO, i.e. electronic coin, as follows:

Definition 3.4. *"An electronic coin is a chain of digital signatures."*

The UTXO can, therefore, be thought of as your coins – that you are free to spend. To spend the coins, you simply have to construct a transaction where you reference the coins from a previous transaction that were sent to an address you control.

A well-formed transaction references an UTXO, as an input, cryptographically signs it, and outputs one or more new UTXOs that are now associated with the address(es) of the receiver. The sum of values in the input has to be larger than, or equal to, the sum of outputs e.g. you cannot spend more coins than you own. The transaction format is specified in the Bitcoin protocol, and has the following fields:

Version Number The only available version number is currently 1.

Input Count The total number of inputs, must be at least 1.

List of Inputs A list input items with their associated data.

Output Count The total number of outputs, must be at least one.

List of Outputs A list output items with their associated data.

Block Lock Time The earliest time the transaction can be processed by the network.

One transaction will spend the entire amount that is stored in the referenced input addresses. Most of the time the amount that you want to spend is not the same value you have in one address, therefore more addresses can be combined as inputs, and the outputs can be split up so that some output is returned to an address you control – in the same way you receive change when spending physical cash.

3.4.1 Inputs, Outputs and Scripts

Bitcoin specifies a stack based scripting language, *Script*, that is used when processing a transaction. The scripting language has a variety of arithmetic and logic instructions, as well as cryptographic functions for hashing and signing. Scripts are a part of the transaction inputs and outputs and specify the cryptographic details for the transaction. The whole execution process is shown in Figure 3.2 for reference.

A transaction input is a fulfillment of the conditions specified in the UTXO that is being claimed in the transaction, and is added to the list of inputs in the transaction see the input list of transaction 2 in Figure 3.2 for reference. Each item in the input list section of the transaction contains the following fields:

Previous Transaction Hash A double SHA256 hash of the previous transaction that output the value to be spent in this transaction. This is a pointer to the transaction that redeems the UTXO that is to be spent in this transaction. In Figure 3.2 the upper dotted line shows this reference.

Previous Transaction Output Index The index of the output in the transaction output list of the transaction referenced in the first field. In Figure 3.2 the lower dotted line shows this reference.

Transaction Input Script Length The length of the following script to be used in this transaction.

Script The input script that is executed in this transaction.

Sequence Number Not currently used except to disable block lock time in a transaction [Seq].

The data received up to this point is actually enough to redeem the value in the transaction. However, the transaction also needs to specify the conditions under which the UTXO from this transaction can be spent in the future. The conditions reference to where the output section of the transaction becomes relevant. Each item in the output list of the transaction contains the following fields:

Value The number of coins that this output references.

Transaction Output Script Length The length of the following script.

Transaction Output Script The output script that is to be executed when this output is referenced in the next transaction e.g. when the receiver of this output wants to spend the value. Transaction 1 in Figure 3.2 is the previous UTXO referenced, and is referenced with the input of Transaction 2. Transaction 2 also needs to include new outputs defining the conditions for the next time the same coins will be spent.

Transaction Execution

When a transaction executes, it combines the script of the previous UTXO with an input script on the execution stack. Figure 3.2 shows this process in the yellow box. The execution script is assembled from a previous transaction in the blockchain – Transaction 1 in the figure, and the new input script in the new transaction – Transaction 2 in the figure, the blue boxes in Figure 3.2. If the execution of the combined script runs without error, the transaction is considered valid. The resulting transaction output in Transaction 2 can then be redeemed by the receiver at some later time, according to the specifics in the new output script. The green box in the figure references some other UTXO that also has been consumed in this transaction, i.e. more than one UTXO was spent in this transaction.

Bitcoin Scripts

Bitcoin scripts can be versatile because of the general scripting language. However, the versatile functionality is rarely utilized in real transactions. Of all executed in Bitcoin, 99.9% [NBF+16] are of the same type: *scriptSig* and *scriptPubKey*. The script, scriptPubKey is an output script saying that anyone who can sign the transaction using the private key related to the address specified in the script, can spend this specific UTXO. The script, scriptSig is simply the corresponding digital signature to that address. During transaction execution the two scripts are combined (see Figure 3.2). For combined script to run to completion the signature in the input script must match the public key specified in the output script.

The only real limitation of Script compared to other programming languages is that it does not include any instructions for creating loops in the code. This limitation is intentional, to prevent malicious users from creating infinite loops in the scripts when other nodes execute them.

Transaction Validation

After a transaction is formed, it is relayed to the other nodes in the network for validation in a peer-to-peer manner. Each new transaction has to be linked to a previous transaction where the value of the previous transaction stays the same, but the ownership of the value is changed. Effectively, this means that transactions are linked together in a chain of transactions, from the value that was created to the latest transaction made. How the value is created, i.e. how the coin is minted, will be discussed in section 3.5.2.

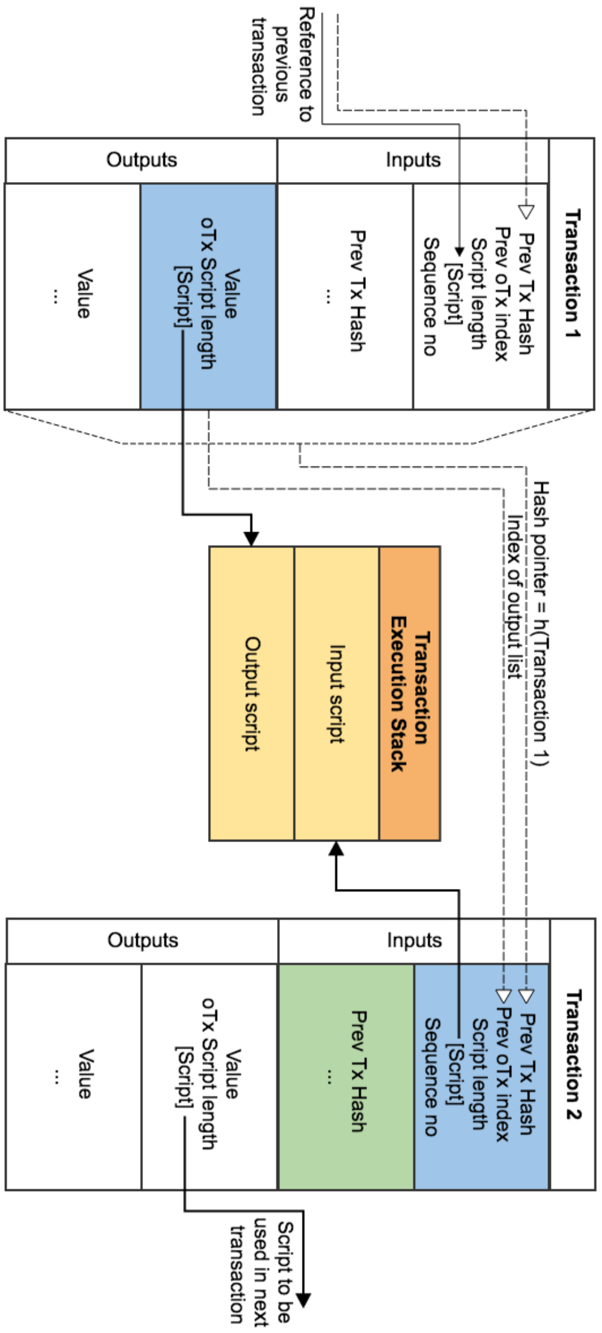


Figure 3.2: Diagram showing the execution of a script in Bitcoin.

To check if a transaction is valid, the digital signatures related to all previous transactions involving the value you wish to check need to be validated. In other words, all the scripts from the inception of the value to the script you wish to validate, need to execute successfully.

Additionally, the validation process needs to check that the transaction has not already been spent in some previous transaction e.g. that one UTXO has been signed for in more than one transaction. This is *double spending* and means that you are spending the same value in different transactions. In the analogy of physical cash, for instance, this means that you would spend some cash buying groceries and then spend the same cash again buying new clothes. This is of course illegal, as it would cause the money to lose its value.

For any node to be able to check if a coin has been spent already, implies that all nodes need to keep track of all transactions that has happened in the network. Since transactions can be relayed from any node at any time and any position in the network, this is not trivial – and is an instance of the decentralized consensus problem from section 3.1.3. Bitcoin solves this problem by utilizing a *blockchain*.

3.5 The Blockchain

This section describes the dynamics of the Bitcoin blockchain: how it is assembled, validated and agreed upon.

The blockchain is a data structure. Bitcoin uses this data structure to create an ordered *immutable ledger* of transactions, so that all nodes can agree on which transactions happened at what time and in what order. The blockchain is assembled from blocks of structured data, with a header and body format. The blocks are chained together by using hashes in the header as pointers to previous blocks. Figure 3.3 shows this structure. The blockchain is immutable because changing data in an early block would require a chain reaction of changes in the pointers in all following block hash pointers.

3.5.1 Bitcoin Block Structure

Figure 3.4 shows the general structure of a Bitcoin block. In the figure, the green boxes constitute the block header. There are nine fields in the block in addition to the transaction list. The fields are described below.

Magic number A hard coded 32-bit preamble for every new block, the value is always 0xD9B4BEF9.

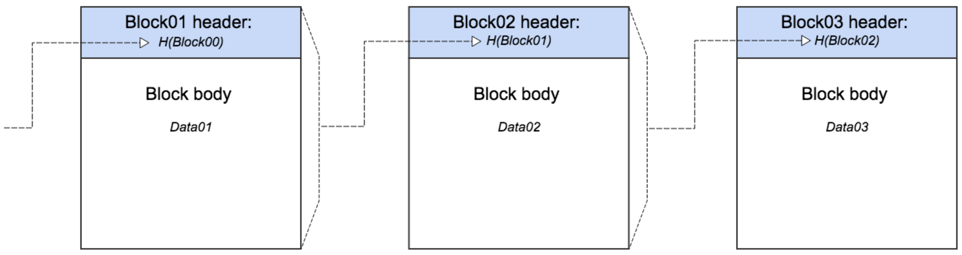


Figure 3.3: General structure of a blockchain.

Block size The number of remaining bytes in the block. The maximum block size is hard coded to be 1MB in the current version of Bitcoin.

Version Version number of the blockchain protocol.

Hash of Previous block The header of the previous block is hashed, and the hash is stored in this field. This field is both a pointer to the previous block and a security mechanism of Bitcoin. This will be discussed further in section 3.5.2.

Hash of Merkle Root A Merkle Tree is constructed from the transactions in the Transaction list part of the block, the root of this tree is hashed, and works as a cryptographic proof of the integrity of the transaction list. Merkle trees were discussed in Section 2.2.1 for more details.

Time A time stamp for when this block was assembled.

Bits This field is relevant for the consensus algorithm for Bitcoin. It will be discussed in Section 3.5.2, when Hash of previous block is described.

Nonce A random number. This field is relevant for the consensus algorithm for Bitcoin. It will be discussed further in Section 3.5.2.

Transaction counter The number of transactions in the transaction list.

Transaction list This is the data section of the block. All transactions that have been relayed by the network since the last block will be put in this list.

The blockchain forms an ordering of transactions so that double spends can be detected as long as everyone adheres to the same chain. By listening for transactions incoming on the peer-to-peer network, anyone can assemble a block according to this structure, and validate that the transactions included are well formed and do not double spend.

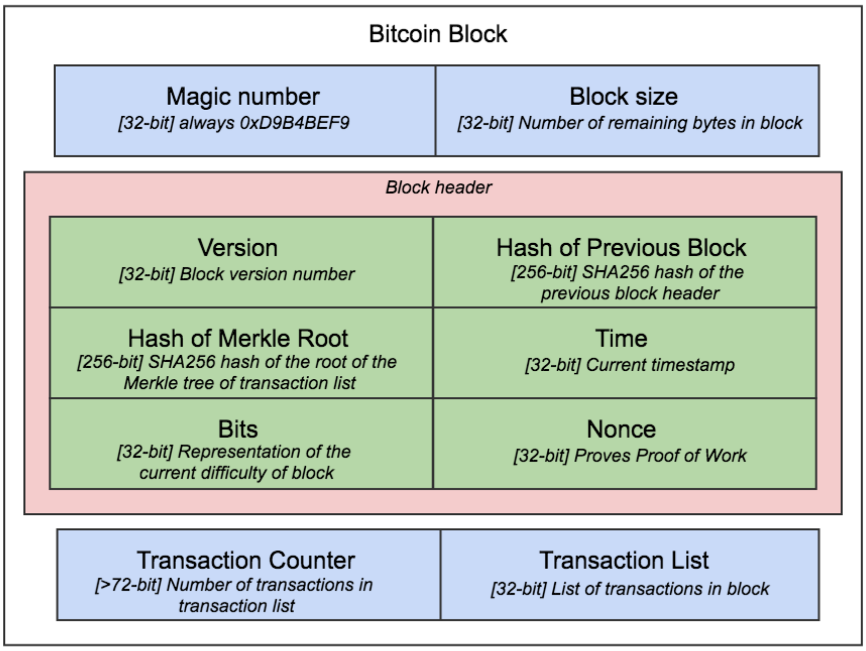

Bitcoin Block
Magic number
[32-bit] always 0xD9B4BEF9
Block size
[32-bit] Number of remaining bytes in block
Block header
Version
[32-bit] Block version number
Hash of Previous Block
[256-bit] SHA256 hash of the previous block header
Hash of Merkle Root
[256-bit] SHA256 hash of the root of the Merkle tree of transaction list
Time
[32-bit] Current timestamp
Bits
[32-bit] Representation of the current difficulty of block
Nonce
[32-bit] Proves Proof of Work
Transaction Counter
[>72-bit] Number of transactions in transaction list
Transaction List
[32-bit] List of transactions in block

Figure 3.4: Block diagram of a Bitcoin block.

Unfortunately, the problem of propagation delays and "which block came first" will occur if blocks could be created at random and relayed to the network by arbitrary nodes. Section 3.5.2 discusses the details of what is known as *mining*, and how this process allows the network to come to consensus on one canonical blockchain, independently of any nodes subjective view of the network state.

3.5.2 Mining

Mining is the process of validating transactions and assembling blocks in Bitcoin. Mining can be done by any node in the network with the necessary amount of computing power and energy. Each new block that is created awards the *miner* – the person who assembled the block – a fresh batch of newly minted coins. This is how new coins are minted in the system (this procedure will be discussed further in Section 3.5.2). The idea behind mining is that some nodes in the system will be *incentivized* to perform the task of validating the transactions in the blockchain in return of monetary rewards.

Proof of Work

For the network to be able to reach a consensus to which block should be appended to the blockchain a method commonly known as *proof of work* is used. The idea is that a block needs to include proof that some level of computational work has been done to create the block to be considered valid. This idea has been around for a while in other types of systems; both Back[B^+02] and Dwork et. al. [DN92] discusses its use in different systems. In Bitcoin the proof of work scheme has two purposes:

1. The time it takes to create a proof of work allows the newest blocks to circulate the network so that all nodes are aware of the most recent state. The algorithm is designed to adjust the difficulty of the proof so that it takes approximately *10 minutes* to create each new block.

2. Computing a proof of work is computationally expensive, thereby making attempts at disrupting the system by flooding the network with false blocks *very* expensive. This is argued as a security property of the system.

Minting Procedure and Incentives

In Section 3.4 we discussed how value is exchanged in Bitcoin as a chain of signatures, from the inception of the coin to the current transaction. However, we omitted the procedure of creating new coins. The process of minting new coins is actually "baked" into the mining process and adds incentive to nodes to volunteer to perform this task.

We know that all transactions have at least one input and one output. The exception to this is the first transaction in a new block – called the *Coinbase* transaction. This transaction has an arbitrary input and a UTXO that can be redeemed with the miners signature. The exact amount of UTXO the miner is rewarded for mining is hard coded into the Bitcoin protocol according to the following model.

The model is a geometrically decreasing series on the block height and a block reward, where the original block reward of 50 coins is halved every 210 000 blocks. Figure 3.5 graphs this function. Figure 3.6 shows the effect of this on the total supply of coins in the system. As is evident from the figure, the total supply of coins tends to an upper cap of 21 million coins.

As is apparent from 3.5, the value of the coinbase transaction is decreasing towards zero over time. To keep the incentive for mining, transaction fees are introduced. Transaction fees are defined as *the value left over from a transaction input after subtracting the outputs.* Transaction fees are voluntary, but so is including

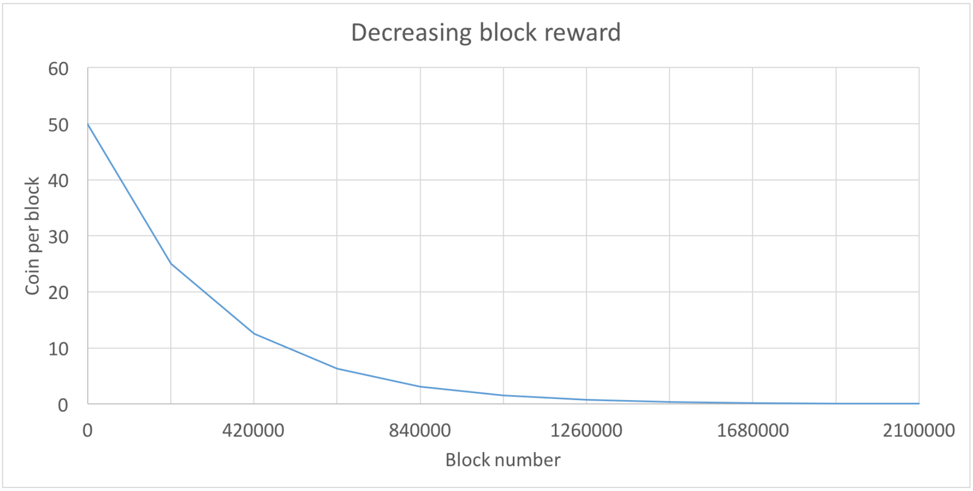


Figure 3.5: Graph of the decreasing block reward as a function of block number.

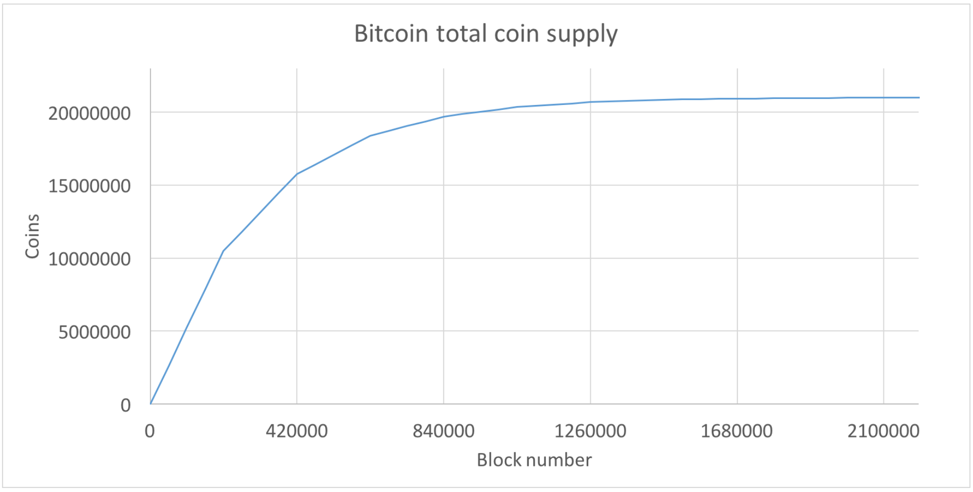


Figure 3.6: Graph of the total supply of coins as a function of block number.

a transaction in a block, a miner might choose to ignore transactions without fees. This process is subject to market supply and demand. As coinbase rewards go down over time, it is assumed that this will balance out and that transactions fees will provide miners with enough incentive to keep mining. However, how this will work out is yet to be known.

Hashcash Algorithm

The mining algorithm Bitcoin uses is called Hashcash and is derived from Back's Hashcash algorithm [B+02]. The idea behind the algorithm is that the miner needs to assemble a block in such a way that the hash of the block header is the solution to a *partial pre-image* of a hash, i.e. the hash digest that is produced by hashing the header needs to be below a well defined numerical value. The hashing algorithm used is the NIST standard SHA256. The header is actually hashed twice to be a valid block, where the difficulty is a numerical value that needs to be bigger than the resulting hash value:

$$\text{difficulty} > h(h(\text{BlockHeader}))$$

From the properties of cryptographical hash functions (see Section 2.1.1), it is infeasible to find such a value – except to try and brute force the output, which is exactly what the miners do. Of course, the block header has a certain structure, so all values cannot be changed at random. Two fields can be changed during mining, without breaking the structure:

1. **32-bit Nonce** This field in the block header, and is typically the first field that is brute forced.
2. **Coinbase data** The input section of the coinbase transaction can hold any arbitrary data. This works as an extra Nonce because changing the data in the coinbase transaction has ripple effects on the Merkle Tree root. Changing the value in the coinbase transaction involves re-computing the Merkle tree – this is typically done after running out of potential Nonce values because it is a more computationally expensive process.

The level of difficulty of the hash is updated every 2016 blocks. This is done by evaluating the timestamps of the previous blocks, comparing them to the goal time of two weeks per 2016 blocks (approximately 10 minutes between each block) and adjusting the difficulty accordingly. Let D_{new} be the new block difficulty, D_{current} the current difficult. T_{current} and $T_{\text{current}-2016}$ denotes the timestamp in the block header of the block of the specified height. T_{target} is the ideal time of 10 minutes between each block that is the target of the system.

$$D_{\text{new}} = D_{\text{current}} * \frac{T_{\text{current}} - T_{\text{current}-2016}}{T_{\text{target}}}$$

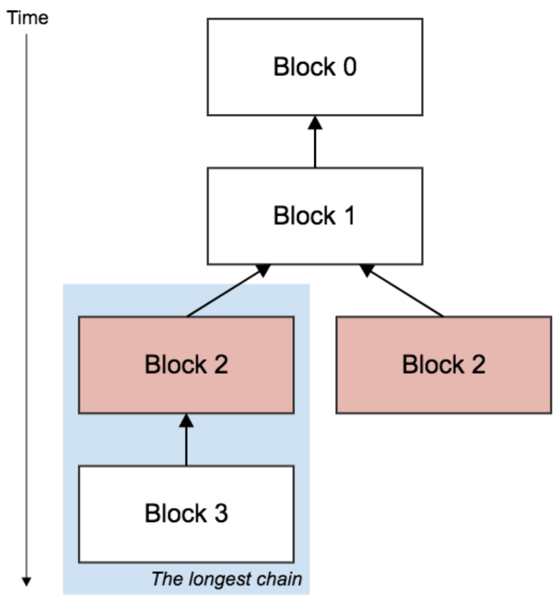


Figure 3.7: Example of a fork in the blockchain.

Nakamoto Consensus

The consensus protocol of Bitcoin has been dubbed *Nakamoto Consensus* after the original idea from Satoshi Nakamoto [Nak08]. The Nakamoto consensus is a strategy of operation that makes it so, if most nodes adhere to it, the biggest reward that can be gained for a node is achieved by staying with this strategy. From game theory this is known as a *Nash equilibrium* [OR94].

The strategy can be summed up in a simple sentence: *The longest chain wins*. Meaning that if a node is presented with more than one block that is not part of the same chain – a fork in the blockchain – the recommended course of action is to stay with the chain that has the most blocks. If any fork has the same amount of blocks, the block with the highest difficulty wins e.g. the block hashes to the lowest value. Figure 3.7 illustrates this process. The two blocks <Block 2> are valid blocks with the same parent, at this point in time a fork has happened. The other miners will have to decide upon one of the two blocks to extend. Once <Block 3> has been mined the longest chain will be decided and the other miners will switch to this chain in the fork.

Because of the probabilistic nature of the Hashcash algorithm, the nodes will converge to one consistent blockchain when following this rule, as the event of two blocks being created within the same window of time consequently diminishes exponentially.

This kind of consensus is called *eventual consensus* [NBF$^+$16] because blocks cannot be trusted before one can be sure that it is a part of the longest chain. There is a possibility of blocks that have been created according to protocol may not be included in the blockchain, because other blocks were created at effectively the same time. The probability of a block not becoming a part of the *main chain* falls exponentially with the number of blocks that are subsequently added on top of the block in question.

The rule of thumb used in Bitcoin is that after six children blocks have been added, the block is considered confirmed. With a block time of 10 minutes, a block is considered confirmed in the blockchain after 60 minutes [Nak08].

Block Validation

When a miner receives a block from a peer, the following points needs to be validated (not necessarily in this order). If any of the points to not hold the block is discarded and is not considered a part of the blockchain.

Transaction structure Transactions are well formed with valid signatures.

Transaction chain No double spends; the UTXO has not been spent already.

Coinbase and transaction fees Coinbase transaction outputs correct value and transaction fees are in line with the transaction inputs.

Proof of Work The block hashes to a value below the difficulty target.

Nakamoto consensus The previous block was the last block in the longest chain.

Blocks that are well formed, but indecisive about which chain is the longest are saved until the tie is broken.

It is important to note that the validity of a block is a subjective matter for the nodes in the system, because the chain of transactions may vary depending on which block they receive first. However, if the nodes follow the Nakamoto consensus protocol, their subjective view of the blockchain will eventually converge to the canonical view of the network.

3.6 Bitcoin at Scale

Since 2009 the Bitcoin community has evolved as more miners and transactors have joined the system. Some of the consequences of this evolution will be discussed in this section.

3.6.1 The Growing Blockchain

As is the nature of the blockchain, the size is continuously expanding to include more blocks. As a measure to control the growth a hard coded maximum limit of the block size is set to be 1MB. Controversy exists around this block size limit [Bloa] – many believe it should not be so stringent as it is today – but as we have learned, making such a change would require a hard fork of the system and is therefore not that easy to achieve. Additionally, in Section 4.6.3 we discuss increasing the block size and the implications this has on the level of security for the consensus protocol.

Even with the hard limit of 1MB per block the size of the blockchain today amounts to over 100GB [3] – and it will keep growing. This is more storage than the average personal computer has to spare, meaning that for most people wanting to run a full validating node e.g. a node with the entire blockchain downloaded, they will have to buy dedicated hardware for storage. This development was predicted by Nakamoto in the design phase of Bitcoin, which is why the original white paper [Nak08] specifies how so-called *Light nodes* could operate, these are further described in Section 3.6.2.

Figure 3.8 shows this development since 2009; the data is collected from blockhain.info, an online blockchain explorer. The reason the size remained near zero the first few years is due to the fact some time passed by before the system picked up a substantial amount of users to generate transactions. Figure 3.9 shows this development. Evidently, on average since late 2016, blocks are reaching the maximum block size.

3.6.2 Light Nodes

A Light Node is a node that does not store the entire blockchain. Light nodes make using Bitcoin more versatile, as they can be implemented in devices with fewer hardware requirements – such as smart phones and tablets. Light nodes allow for transactions to be generated and verified "on-the-fly" by utilizing *Simplified Payment Verification (SPV)*, instead of the standard method described in the previous sections.

An SPV process involves querying full nodes for blocks from the longest chain. When the transaction in question is included in the block, the Merkle tree can be

[3]https://blockchain.info/charts/blocks-size

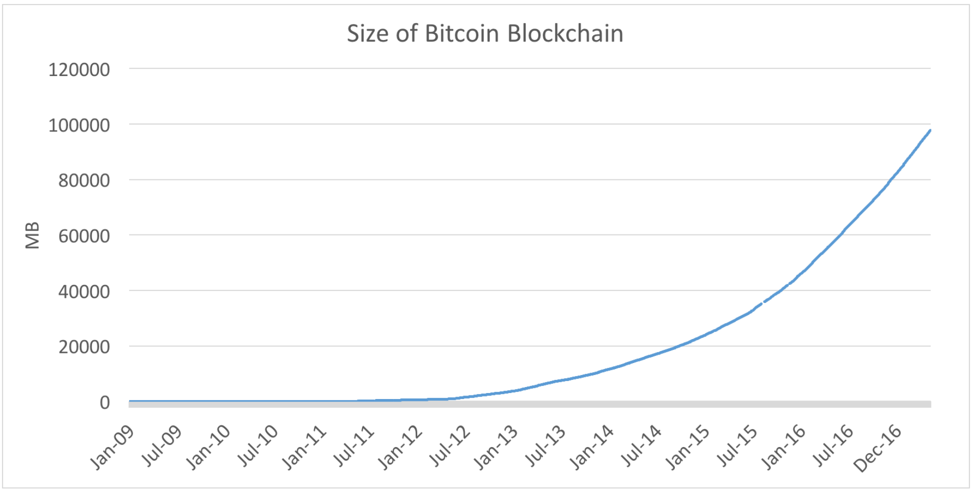


Figure 3.8: Graph of the total blockchain size since the beginning of 2009.

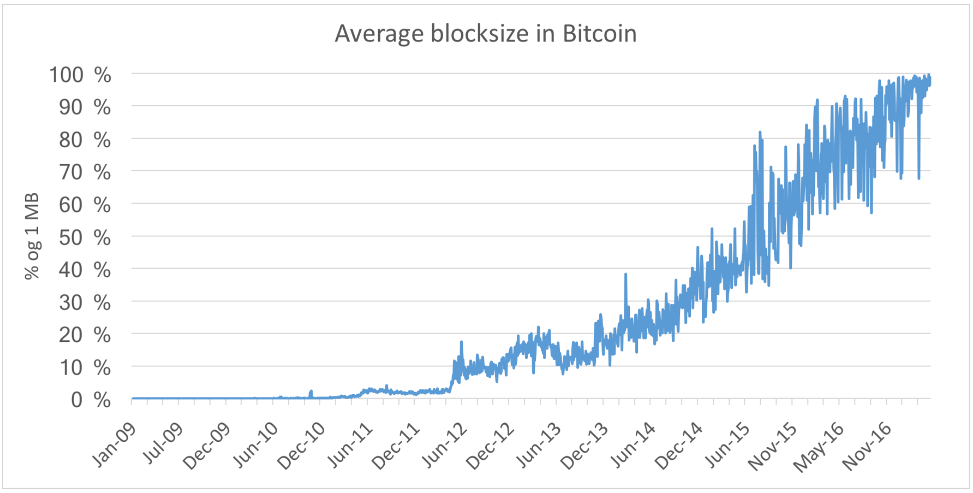


Figure 3.9: Graph of 7 day average block sizes since the beginning of 2009.

used to verify that the transaction is a part of the block. Consequently, light nodes have to trust that the majority of the full nodes honestly validate the transactions in the blocks – as they are not able to prove this for themselves without downloading the entire blockchain. As long as this assumption holds, the longest chain will include only valid transactions. They can be proved to be a part of the blockchain by showing the Merkle tree branch including the given transaction.

3.6.3 ASICs

Recall from Section 3.5.2, to mine a valid block the miner needs to find a partial pre-image to a hash, where the time it takes to produce the proof is directly proportional to the amount of hashing power you control. As Bitcoin has become increasingly popular and valuable, creating new blocks and earning coins has become a business for serious miners. There is a continuous race to be in possession of the fastest hardware that can run the Hashcash algorithm, so the likelihood of producing blocks is higher. The design of specialized hardware units, so-called Application-specific Integrated Circuits (ASICs), has sped up the hashing process by several magnitudes – making it infeasible with the difficulty level today for regular CPUs to produce new blocks [NBF+16].

ASIC hashing rigs are usually set up in data farms, running full time to validate transactions and produce new blocks. The level of energy these data farms consume for the sake of reaching consensus in the blockchain has caused concern in the community. Micali [Mic16] raises the *computational wastefulness* of the mining process as one of the main technical problems with Bitcoin. Narayanan et al. [NBF+16] roughly estimates the energy consumption of mining in the orders of a few hundred Mega Watts, which is comparable to the energy consumption of a medium-sized country.

3.6.4 Mining Pools

Mining pools are organized Bitcoin mining nodes that mine towards a common block and then split the reward after mining a successful node. The organizations are completely outside of the Bitcoin protocol. Mining pools typically have a lot of miners trying to solve the proof of work for a block together, and an operator that is in charge of paying out rewards when successful blocks have been found. A mining pool known as *SlushPool* claims to be the first mining pool [Homb] and has been around since December 2010 – less than two years after the first Bitcoin block was mined.

Why do miners wish to join mining pools? There are three main reasons for this. First of all, even though the reward for mining a successful block is enough to incentivize nodes to mine, the probability of being the one who finds a block is like winning the lottery – with diminishing odds as the network difficulty goes up.

Statistically, the *variance* of getting a reward is high. A miner could start mining, and it might take years before actually receiving a block reward unless she gets very lucky. By joining a mining pool, the probability of finding a valid block in the pool goes up, and the reward goes down – because you have to split the reward with the other pool members – but the expected return over time remains the same. E.g. joining a mining pool lowers the variance of the return of reward, giving the miner a more steady income.

Secondly, with the event of ASICs taking over the mining market, miners have to initially invest in extra hardware to mine. This development has made mining pools even more lucrative because the risk of being unlucky and waiting years to get a reward is reduced, helping to pay off on the initial investment.

Lastly, miners that are a part of a mining pool only handle the mining process – not the validation – meaning that they do not have to store the entire blockchain more than once in the pool. With the size of the blockchain expanding every year. This means that for nodes that are a part of a mining pool, investing in extra storage is not necessary.

Centralization

The effect mining pools have on the network topology is interesting, and some would say contradictory to the original intents of having a decentralized system. A mining pool is a centralized unit of miners that put their trust in the pool operator to behave honestly and return rewards to the miners when a block has been mined. Should the network operator behave dishonestly, this might not be apparent to the miners in the pool.

Figure 3.10 shows the distribution of hash rate among the different pools in the network. It is evident that the control of the network is distributed among a few factions of mining pools. This natural tendency towards centralization is the second of Micali's technical problems for Bitcoin [Mic16].

3.7 Security of Bitcoin

This chapter concerns the security assumptions of Bitcoin and potential attacks. The security of Bitcoin is based on the cryptography that enforces transactions and that the nodes correctly validate the given cryptographic proofs. Additionally, the assumption of the proof of work consensus mechanism is that the majority of the hashing power in the network is controlled by honest nodes – so that the blockchain cannot be rewritten by a malicious actor. If a malicious actor controls more than half of the hashing power in the network, this person can perform what is known as a 51% attack.

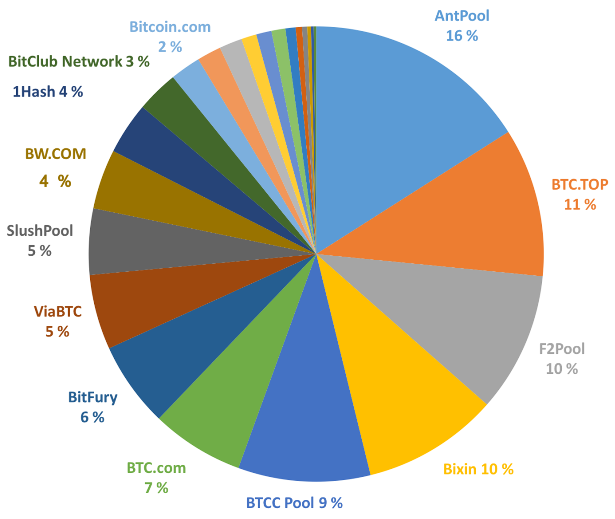


Figure 3.10: Pie chart showing an estimate of the current hash rate of the Bitcoin mining pools over the last seven days (28. May 2017).

3.7.1 51% Attack

A 51% attack is the most serious attack at Bitcoin because it makes it possible for the attacker to double spend coins by reverting transactions. Should such an attack be made, it would undermine the value of Bitcoin, causing people to lose trust in the system and possibly refusing to trade for the currency. Nakamoto described this attack in the original white paper [Nak08].

Consider the following scenario: A transaction is added to the blockchain in return of some service. The appropriate six-block-confirmation time has passed, and the service is exchanged. Afterwards, a malicious node that controls 51% or more of the total networks hashing power would be able to rewrite the block proofs of the last six blocks – excluding the transaction paying for the service – making it so the value was never exchanged, and the coins can be used again by the attacker, e.g. double spent.

The 51% attack is possible because, in time, the majority of the hashing power will always tend towards the longest chain. The rewritten blocks do not include any illegal transactions, so this is hard to detect by other nodes. The attack exploits the *ambiguity* of the eventual blockchain consensus. Micali [Mic16] lists this as the third technical problem with Bitcoin: No transactions can be guaranteed at any point in the blockchain, should the network be overpowered by an actor controlling significant hashing power the blockchain can be rewritten.

The situation where one node controls such an amount of hashing power is considered unlikely. That more than half of the nodes are honest is the security assumption Bitcoin is based on. However, with the event of mining pools that operate as centralized units and control large portions of the hashing power, this might not be as unlikely as originally thought.

3.7.2 Selfish Mining Attack

The selfish mining attack was first described by Eyal et al. [ES14]. Eyal et al. state that for miners of substantial mining power the general Nakamoto Consensus is not the most profitable strategy. This attack involves miners using a different strategy to increase their total block rewards. The idea behind the attack is to strategically withhold blocks in a separate chain as long as the separate chain stays ahead of the main chain. Say, a pool of miners finds a valid block, B_0. Instead of announcing it to the network immediately the miners withhold the block. In secret, they will keep mining for the next block, B_1 – getting a head start on the block from the rest of the network. If an honest miner finds a different B_0 – before the selfish pool finds the next block – then the selfish pool announces their block, creating a fork, but with still a greater chance of winning the fork against the other block since half the network will be mining on either block in the fork. As long as the selfish pool monitors the main chain so that it does not get ahead the attack is more profitable to the selfish miners. The attack is more effective the bigger the mining pool is, and makes the efforts of the honest miners wasteful.

Chapter 4
Ethereum

Ethereum received startup funding in 2014 as a crowd-funded development project, marketed as an improved protocol to Bitcoin. The value proposal in contrast with Bitcoin is that Ethereum aims to be something more than just a cryptocurrency: By allowing functionality for defining and executing so-called *smart contracts* on its blockchain, the idea is that Ethereum should function as a *platform*. Users of the system should be able to design decentralized applications as a second layer on top of Ethereum – Ethereum should be a *Global Virtual Machine* accessible to anyone. To facilitate this functionality, Ethereum provides a stateful blockchain with storage capabilities and a Turing-complete scripting language. This chapter will go through the technical details of Ethereum as a parallel description to the previous chapter of Bitcoin.

4.1 What is Ethereum?

The Ethereum System is founded and developed by The Ethereum Foundation. The Ethereum Foundation is in charge of developing the specifications for the Ethereum protocol, as well as several types of open-source clients that can operate the protocol – the most popular of these; the Geth node.

Similarly to Bitcoin, system specifications exist in various forms. For Ethereum the main sources of information are the Ethereum White Paper [But13], and the Ethereum Yellow Paper [Woo14] – a formal technical specification of the protocol. Additionally, continuous updates from the development team are posted on the Ethereum blog [Pro] and details on technical aspects and discussions are available on GitHub [Homa].

The Ethereum Foundation launched the system in May 2015, but the project was far from finished at that point. The developers are in close contact with the community to continue improving the protocol and provide tools for accessing the system. With the intention of avoiding splits in the community about the direction of Ethereum some planned development milestones have been published [Gup15]. These are essentially hard forks that the whole system has to agree to perform:

Olypmic: May 2015 Ethereum proof of concept. This was the first testing phase, no value transferred from this blockchain to the next.

Frontier: July 2015 First real version. A simple implementation of the protocol, with command line client interface.

Homestead: March 2016 Current version of Ethereum. There may still be more versions following up before the last two milestones.

Metropolis: TBA This version is identified by a full-featured user interface for non-technical users.

Serenity: TBA The final version of the system. This includes a major change in the consensus algorithm. A lot of research is currently being done on the topic of consensus algorithms.

4.2 World State

Ethereum has a stateful blockchain, in contrast to Bitcoins stateless blockchain. This implies that in Bitcoin keeping track of the state of who owns coins in the system is encouraged; it is not strictly necessary for block generation and validation. In Ethereum however, keeping track of the state of each account is strictly necessary to be able to mine new blocks. In Ethereum this concept is called the *World State*. The world state of Ethereum is a Merkle Patricia Tree (see Section 2.2.3) of data objects, called *Accounts*. Each new transaction that is generated in Ethereum triggers a *State Transition Function*, that is applied to the current state and deterministically updates the state of the system [Woo14]. When nodes in Ethereum come to consensus on the blockchain, the updated World State is what they agree upon. Additionally, Ethereum mints its own currency – Ether – which has intrinsic value in the system.

4.2.1 Accounts

Accounts are *stateful* objects. Transactions to and from accounts cause the states of the accounts to change. For instance, a regular transfer of funds would cause the balance of an account to update, changing the state of the account.

There are two types of accounts that are defined by how they are created and how they are controlled:

Externally Owned Accounts Externally owned accounts are owned by external actors and controlled by private keys (much in the same way as Bitcoin addresses). The external actors are typically human users of the system.

Contract Accounts Contract accounts are autonomous agents that are controlled by the program code that they are initialized with. They can be interacted with by creating transactions through Externally Owned Accounts.

A contract account can be created for holding some value in deposit while a physical exchange is happening in the real word. As an example: Alice wants to ship some goods to Bob in return of payment. Bob does not trust Alice, and does not want to send the money directly to her without seeing the goods first – Alice does not trust Bob to pay once he has received the goods, and will not send the goods before payment has been proven. There is no mutual trust. In this situation, Bob can prove that he has the money by transferring it to a specific contract account that has been programmed for this purpose, where the money will be locked until the goods are received. When the goods have been received, both Alice and Bob will send new transactions to the contract indicating that the money should be forwarded. Unless they both send this transaction the money will stay locked down. To mitigate the problem of Bob maliciously refusing to forward the money, the contract could be expanded to include a trusted third party to mediate the conflict.

The generic account state object is composed of four fields and is identified by a 160-bit number called an *Address*. Depending on what type of account we are dealing with the data fields in the account will have different values and properties. In general, they can be described like this [Woo14]:

nonce A counter that is incremented based on the number of transactions sent and contracts created by the account.

balance The value owned by this account.

storageRoot The root hash of the Merkle Patricia Tree that constitutes the storage of the account.

codeHash A hash of the code that specifies how the account should operate. This is empty if the account is an externally owned account type.

Cryptographic Primitives of Ethereum

There are two significant cryptographic primitives for Ethereum: The hash function and the digital signature (see Section 2.1). Ethereum utilizes the non-standard SHA3 version, *Keccak*, for its hashing operations. The same security properties for SHA3

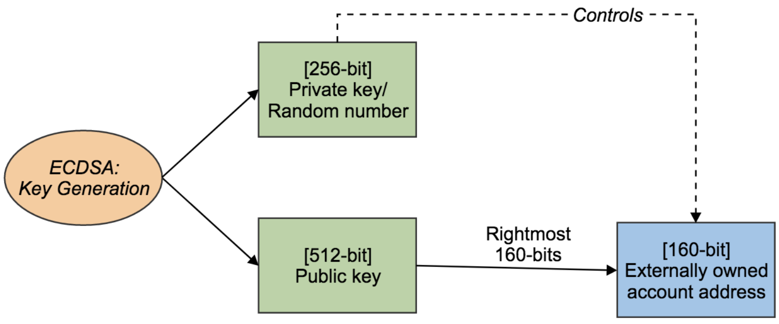

Controls
[256-bit] Private key/ Random number
ECDSA: Key Generation
[512-bit] Public key
Rightmost 160-bits
[160-bit] Externally owned account address

Figure 4.1: Address and key generation for externally owned contracts.

applies for Keccack, but there are some implementational differences. The digital signature of choice is the same ECDSA curve that Bitcoin utilizes, Secp256k1 [Qu99], probably with the same reasoning (see Section 3.3).

4.2.2 Externally Owned Accounts

An externally owned account can be generated by anyone. A private, public key pair for an externally owned account is generated using ECDSA key generation, and the account address is then derived from the output public key. The private key needs to be protected because whoever holds the private key controls the funds in the account. Externally owned accounts are identified by their address – where the address is the last 160 bits of a hash of the public key [Woo14]:

$$\text{Address} = (h(\text{publicKey}))_{\text{Bits}[96:255]}$$

Figure 4.1 depicts the procedure of generating a public key – a new account – as a function of a random 256-bit number. The random number constitutes the private key. ECDSA key generation is the basis for creating an address with a corresponding private signature key. The address equals the rightmost 160-bits of the public key. The private key controls the account. All transactions from this account will need to be signed by the corresponding private key related to that address.

4.2.3 Contract Accounts

A contract account is created by executing a *contract creation transaction* from an externally owned account [But13]. The contract creation transaction specifies program code for the new account. This code controls the behavior of the contract in future transactions. Contract accounts can be thought of as autonomous agents that

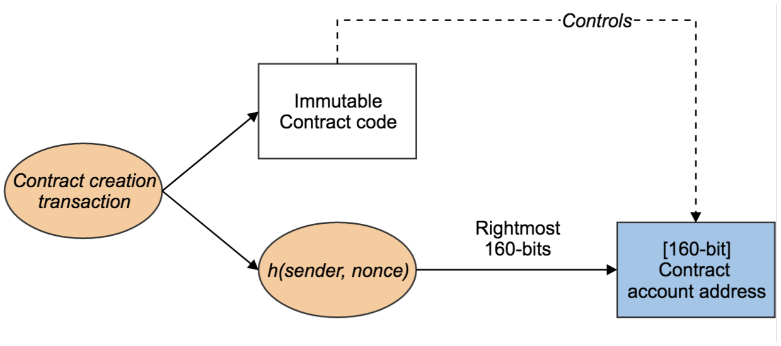


Figure 4.2: Contract account creation.

come alive when they are interacted with through other transactions. The contract has read/write access to the storage of its own account and can utilize this access if the initial code has been constructed appropriately. However, the code the contract been initialized with is immutable after the contract has been created. Contracts can also produce new contracts if this is defined in the initial code.

A since a contract account is controlled by the implicit code in the contract account, it does not have any associated private or public keys. With no public key to derive an address from, the contract account address is generated as the rightmost 160 bits of the hash value derived from the sending accounts address and nonce [Woo14]:

$$\text{Address}_{\text{contract}} = (h(\text{Address}_{\text{sender}}, \text{Nonce}_{\text{sender}}))_{\text{Bits}[96:255]}$$

Figure 4.2 shows how a contract account is created. Unlike an externally owned account, there are no keys that control the value in a contract account, the contract code does this autonomously. The contract creation transaction is in turn generated by an externally owned account.

4.3 Transactions and Messages

Ethereum differentiates between two types of communication: *Transactions* and *Messages*. These two classes exist because of the nature of the different types of accounts available in the system.

4.3.1 Transactions

Generating transactions is how *external actors* – nodes in the network – interact with the blockchain. Transactions are assembled by external actors and then propagated to the other nodes in the network in a peer-to-peer fashion. Transactions may be subject to malicious behavior from other nodes. For this reason, transactions need to be signed by the external actor wishing to execute the transaction with the corresponding private key to the account the transaction originates from. This is much the same procedure as we know from Bitcoin, except the signature provided is not for spending a UTXO, but rather to authorize a state transition on an externally owned account. Note that in Bitcoin the entire value in a Bitcoin address needs to be spent in a transaction, this is not necessary for Ethereum. It is perfectly fine to spend only a fraction of the value in an Ethereum account.

Transaction Structure

A well formed transaction contains the following fields [Woo14]:

nonce The nonce is a value that is equal to the total number of transactions sent from the sender's account.

gasPrice The price per computational step (see Section 4.4.3 for details).

gasLimit The maximum accumulated price for all the computation this transaction will invoke (see Section 4.4.3 for details).

to Recipient account address.

value The amount of Ether to be transferred to the recipient's address.

v, r, s The values r and s represent the ECDSA signature for this specific transaction, signed by the sender's account. the value v is the 'recovery id' of the curve – the value is either 27 or 28.

data/init For transactions that execute code this field is used. If the transaction creates a new contract account, the *init* field is utilized, and it will contain the initial code for the contract. If the transaction interacts with an already established contract account, the *data* field is used and contains input data to the contract.

4.3.2 Messages

Messages are different from transactions. They are communication spawned by contract code execution e.g. they originate from a contract account. A message is always created as an effect of a transaction calling the contract. Messages are therefore not signed, as contracts have no keys to sign with. More importantly, messages do not *need* to be signed because they are never relayed on the network and only exist as a part of the Ethereum execution environment, as a consequence program code executing. Figure 4.3 is an example of how messages and transactions are passed between accounts in the Ethereum execution environment. As is evident from the figure, all transactions need to originate from an externally owned account. Messages are passed from contract accounts, and in-between contract accounts, only as a result of an initial transaction triggering code in a contract account.

Message Structure

A message is constructed from the following information [Woo14].

sender The contract address that spawned the message.

transaction originator The address of the externally owned account that called the sender contract code execution.

recipient The address of the recipient of the message.

code account to be executed The account whose code should be executed, usually the same as the recipient code.

startGas Available gas for this message call (see Section 4.4.3 for details).

value Value transferred in this message.

gasPrice The price per computational step (see Section 4.4.3 for details).

data Optional input data relevant to the message call.

depth of execution stack Present depth of execution stack.

4.4 Ethereum Virtual Machine (EVM)

The EVM is the execution environment of Ethereum. All full nodes in Ethereum run the same instance of the EVM, and the transactions that are input to the virtual machine changes the state of the system deterministically, so everyone has the same updated state of the system. Every full node has to execute the program code invoked from each transaction. This implies a high level of redundant computation, but this is also what ensures trust in the system.

The EVM is sometimes referred to as a *Global Singleton Machine*. Where a singleton is the only instance of its kind. Ethereum is of course in the physical sense distributed in a network, but the virtual machine it embodies is, in fact, the same instance. Figure 4.4 is a visualization of this. The externally owned accounts are at the edge of the network and are where inputs – transactions – are generated by external actors. Contract accounts can be interacted with through transactions, and

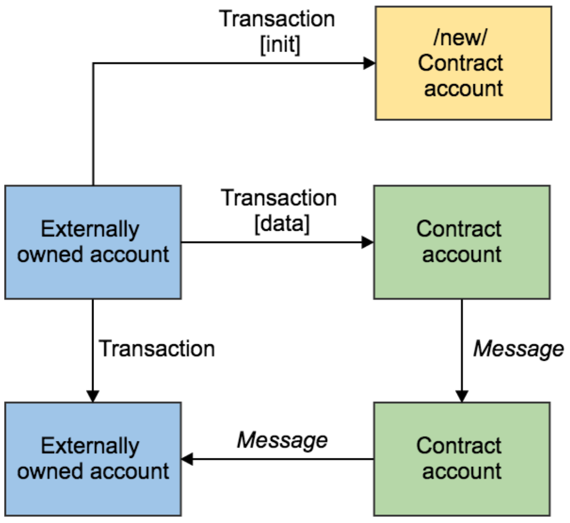


Figure 4.3: Types of communication passed between different types of accounts.

they can, in turn, interact with each other through messages. The EVM is run by all the nodes on the network and is a global virtual machine in that sense. Figure 4.4 visualizes how the virtual machine is spread out over nodes in the network in a peer-to-peer fashion, that are operating on the same set of account objects. These account objects can interact with each other through messages and transactions, and the nodes can control the accounts using corresponding private keys.

4.4.1 Ethereum Virtual Machine Programming

A low-level, stack-based byte code language – EVM code – mandates the behaviour of the EVM. The EVM code is a Turing complete object-oriented programming language [But13]. The account objects have their own code and storage that can be accessed through message calls, e.g. transactions from externally owned accounts and messages from contract accounts.

The specifics for programming in Ethereum are out of scope for this thesis. It is enough to know that there are capabilities for creating a variety of applications as a second layer to Ethereum. Contract accounts have implicit storage capabilities that can be accessed through code, and *Application Binary Interfaces* can be defined for interaction between with contracts already on the blockchain, creating a powerful environment for programmers.

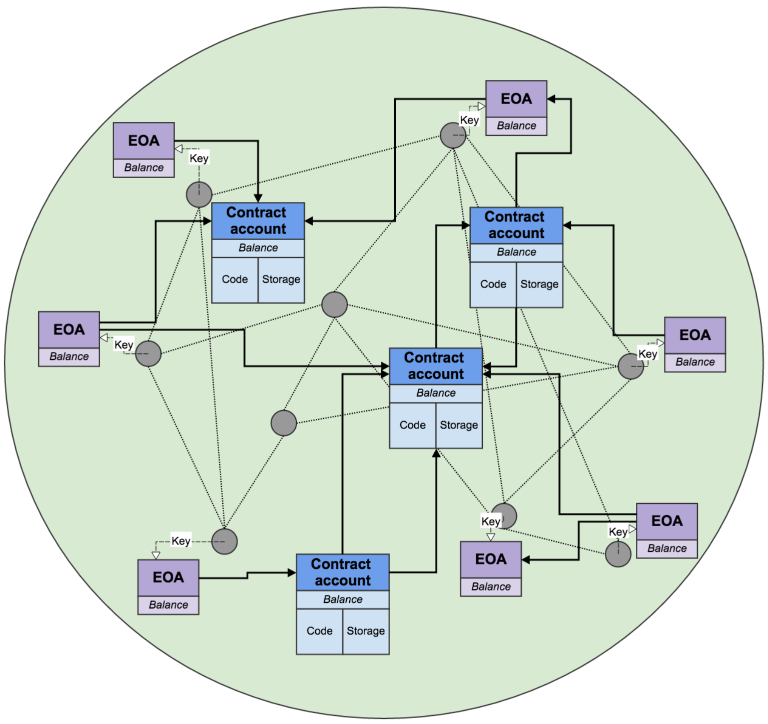


Figure 4.4: A visualization of the EVM, adapted from Woods figure **??**.

There exist several high-level languages that compile down to EVM code. They have been created for ease of programming. *Serpent* [Wik15] – a python like language – and Solidity [Sol] – which is inspired by JavaScript, are instances of programming languages that compile down to EVM code.

4.4.2 The Halting Problem

The halting problem is a classic problem from computer science, first described by Alan Turing [Tur37]. It states that it is impossible to make an algorithm that can determine for some input whether an arbitrary Turing-complete program will terminate at some point, or run forever.

Since every full node in Ethereum has to run the program code invoked by a transaction, this exposes a serious problem in the architecture; a program with an infinite loop would effectively shut down all the nodes in the system if it were executed.

Because there is a real possibility of a node generating code that runs forever – either by mistake or malicious intent – we know from the halting problem that there is no way to verify if this will happen *before* execution. To prevent this situation Ethereum utilizes *gas* as a measure, which is the topic of the next section. On a side note, Bitcoin prevents this problem by having a non-Turing complete programming language, where there are no loops. This strictly limits the utility of the scripts, but also effectively mitigates the problem.

4.4.3 Gas

Gas is to Ethereum, what fuel is to a car. Ironically, it is commonly referred to as cryptofuel in the community [Acc], in line with this analogy. The general idea of gas is that every computational step that a miner has to perform to run the code in a transaction has a cost, a *gasPrice*, that the transaction originator has to pay to the miner.

Upon initiation of the transaction a *gasLimit*, is set. The gasLimit is the total amount of value that this transaction is allowed to consume. Over the course of the execution, this buffer is tapped into for every computation that is performed, until the transaction is complete.

Should the transaction complete without draining the buffer, the remaining gas is returned to the transaction originator. Should the transaction run out of gas, an error is returned, and whatever changes the transaction code invoked are rolled back. The gas consumed, however, still goes to the miner.

This approach effectively halts any infinite loops in the code because the code will, at some point, always run out of gas during execution. It also mitigates attempts at attacking the system by relaying transactions with big loops, because they will be expensive for the attacker. However, there is a potential weakness in the system related to using gas as a solution for the halting problem, Section 4.8.1 discusses this further.

The gas that is consumed becomes a transaction fee to the miner. Because the mining fee is directly related to the complexity of the computation, miners are not de-incentivized from mining large programs, even though they take longer to verify, because the fee reward will be higher.

The gasPrice is not a fixed Ether value, even though the price is paid in Ether. This is because the price of Ether is subject to fluctuation when it is sold on the free market; one Ether today may be worth twice as much in a year. What is expected to be a fair gasPrice is therefore up to the users and miners of transactions, like in Bitcoin with transaction fees, a miner is free to ignore a transaction with too low gasPrice. However, the total execution cost of the transaction will always be proportional to the level of computation required [Acc].

4.5 Transaction Execution

Formally, a valid transaction is what causes the state of the EVM to change – *only* valid transactions can can inflict such a change [Woo14]. What needs to be agreed upon, to find a common world state for all the nodes in the network, is which transactions and in what order, they are to be executed.

In Bitcoin transactions that were considered invalid were simply not included in the blockchain, as a result of the validation process. Ethereum, however, includes all well-formed transactions in the blockchain, but changes to the system state may be reverted during transaction execution if they result in errors in the EVM. A well-formed transaction has the structure described in section 4.3.1 and need to pass an initial validation phase to be considered well-formed [Woo14]:

- The nonce is correct, e.g. valid to the senders current nonce.
- The signature corresponds to the public key of the sender's account.
- The gasLimit is larger than the gasPrice
- The sender's account contains at least the gasLimit.

If these checks hold the transaction is considered valid and included in the blockchain.

Once the transaction is added to the blockchain there are two possible outcomes from execution in the EVM:

1. **The transaction runs to completion:** Fees are transferred to the miner, remaining gas refunded to the sender and changes are saved.
2. **The transaction completes with errors:** This happens if the sender does not have enough balance, or the code execution has errors or runs out of gas. In this case, all changes are reverted, but the miner still gets the fees accumulated from execution and the transaction is included in the block.

4.6 The Ethereum Blockchain and Mining

Ethereum runs its own blockchain with no reference to Bitcoin apart from similarities in the design. This has offered the potential for configuring the blockchain differently than Bitcoin. The main differences are:

Block time Ethereum's block time targets 13 seconds in between each block [Woo14].

Block size Ethereum has no upper cap for the size of blocks. A fluctuating value – gasLimit – is used to limit high increase levels in gas expenditure in between blocks, but does not limit the size [Woo14].

Optimizations Several header fields for optimizing execution and verification [Woo14].

Similarly to Bitcoin, Ethereum's mining is based on a proof of work mining scheme. This implies that nodes in the network perform the task of validating blocks generated by other nodes, as well as constructing new blocks. However, a different consensus protocol and different mining algorithm are used compared to Bitcoin. The rationale and process of this will be discussed in the following sections.

4.6.1 Block Structure

As a reference for the following sections, a fully constructed block is shown in figure 4.5. The block has a header/body structure, similarly to Bitcoin, but the header is more extensive. The pink box wraps around the fields in the block header. The green boxes are regular hashes; the yellow boxes are hashes of root nodes of Merkle Patricia Trees. The fields of the block are defined in the yellow paper [Woo14] and described in the following.

parentHash The hash of the previous, e.g. parent, blocks header.

ommersHash The hash of the ommers portion of the block (the last section of the block).

beneficiary The account address of the miner that receives rewards.

stateRoot A hash of the root node of the Merkle Patricia tree that populates all the accounts in Ethereum. This hash is cryptographic proof of the entirety of the state of the system.

transactionsRoot A hash of the root of the Merkle Patricia Tree populated with the transaction data in the transaction portion of this block. This is similar to the Merkle tree root of Bitcoin, but the structure of the tree is different (see section 2.2.3).

receiptsRoot A hash of the root of a Merkle Patricia Tree populated with receipts of the transactions in the block. The receipts are generated automatically during transaction execution in the form of execution logs if this has been configured in the contracts.

logsBloom A Bloom filter [Blo70] – a cryptographic tool for proving non-existence of data in a data set – for logging data in the receipts.

difficulty An integer representing the current difficulty of this block.

number The block height, e.g. the number of blocks since the first block; block 0.

gasLimit The maximum allowed gas expenditure for this block. This value fluctuates deterministically based on the expenditure in previous blocks: If blocks with high gas consumption are created, the value for the subsequent blocks increase. Equally, if the gas expenditure is low, the value will decrease.

gasUsed The total amount of gas spent in the transactions in this block. This value is transferred to the miner of the block.

timestamp A reasonable timestamp for the creation time of this block.

extraData Arbitrary data of maximum 32 bytes.

mixHash Hash value that is necessary for the Ethereum Proof of Work algorithm.

nonce Nonce value that is necessary for the Ethereum Proof of Work function.

Transaction list A list of all transactions included in this block.

Ommer list A list of all new ommers found by the miner of this block (see Section 4.6.4 for details on ommers).

4.6.2 Ethash: The Ethereum Mining algorithm

The Proof of Work mining algorithm of choice for Ethereum is called Ethash. Ethash is a derivative of the Hashimoto algorithm [Dry14] and the Dagger algorithm [But]. The algorithm is designed to be ASIC resistant e.g. to make it difficult – or at least un-profitable – to produce ASICs to increase the level of mining performance [Etha].

Similarly to Bitcoin's Hashcash algorithm, the end goal of the algorithm is to brute-force a partial pre-image of a hash of a block header, so that it meets the target

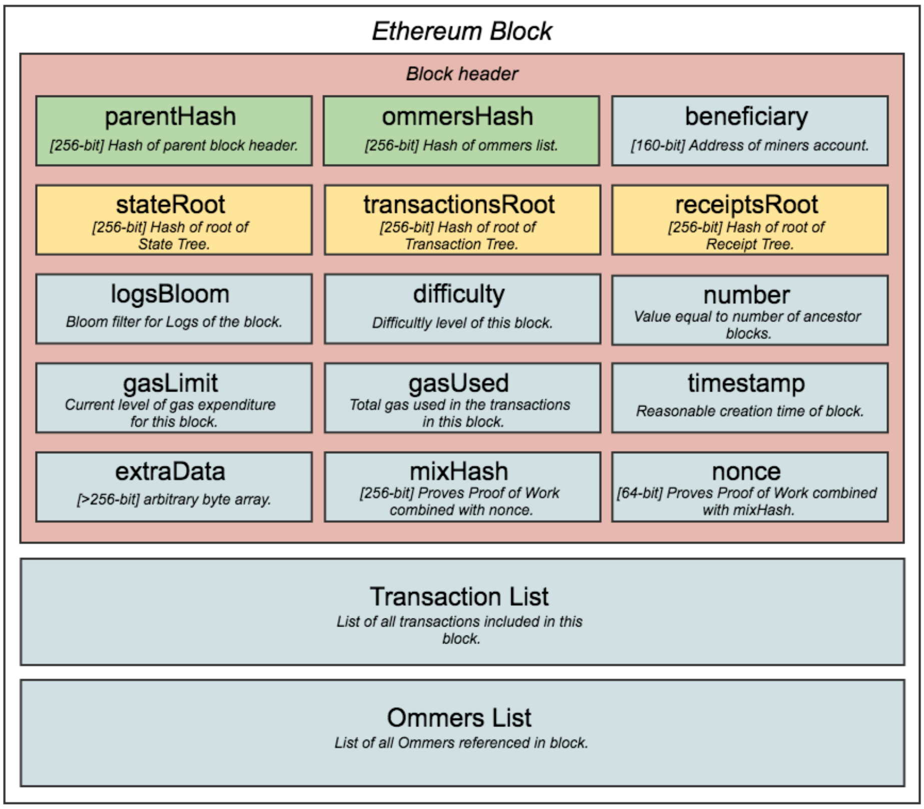

Ethereum Block
Block header
parentHash
[256-bit] Hash of parent block header.
ommersHash
[256-bit] Hash of ommers list.
beneficiary
[160-bit] Address of miners account.
stateRoot
[256-bit] Hash of root of State Tree.
transactionsRoot
[256-bit] Hash of root of Transaction Tree.
receiptsRoot
[256-bit] Hash of root of Receipt Tree.
logsBloom
Bloom filter for Logs of the block.
difficulty
Difficultly level of this block.
number
Value equal to number of ancestor blocks.
gasLimit
Current level of gas expenditure for this block.
gasUsed
Total gas used in the transactions in this block.
timestamp
Reasonable creation time of block.
extraData
[>256-bit] arbitrary byte array.
mixHash
[256-bit] Proves Proof of Work combined with nonce.
nonce
[64-bit] Proves Proof of Work combined with mixHash.
Transaction List
List of all transactions included in this block.
Ommers List
List of all Ommers referenced in block.

Figure 4.5: Diagram of an Ethereum Block.

below a certain difficulty. However, the algorithm is more complex than the simple "hash twice" of Bitcoin: The Ethash algorithm

· e70:online[] uses data collected from the blockchain as a seed value in a pseudo-random generator to assemble a data set. The data set is a Directed Acyclic Graph (DAG) that takes 1GB of storage once assembled. The algorithm for finding a valid proof of work includes, not only changing a nonce value but also fetching pieces of data from the DAG for each try in a pseudo-random way. Every 30000 blocks this data set is recalculated.

The rationale behind this algorithm is that the most costly operation in the algorithm is not the hashing function, but rather the I/O operation – input/output operation – of reading from memory [Etha]. Because most regular personal computers are already optimized for I/O operations, it should be increasingly hard or expensive to create hardware that would speed up the mining process considerably [Dry14].

Additionally, the construction of the DAG is made in such a way that light clients – e.g. verifying, but not mining, clients – only need to store a 1MB cache of the DAG when they are verifying block headers. They can use this cache to compute the necessary nodes in the graph when verifying the block header, without having to store the whole data set.

4.6.3 Adapting the Consensus Protocol

As we know from our Bitcoin reference in section 3.5.2, the consensus protocol is the prescribed method for how the nodes in the network come to agreement on which blocks are valid/accepted and which blocks are not. The Bitcoin consensus protocol – Nakamoto consensus – is the simple strategy saying that *The longest chain wins* (see section 3.5.2). This protocol assumes that propagation delay in the network is negligible, compared to the time it takes to produce a valid proof of work [Nak08]. For the configuration in Bitcoin with a block time 10 minutes between each block, this may hold. However, according to research by Sompolinsky and Zohar [SZ13] changing factors relating to propagation time in the network may impact the security of the protocol. Ethereum has a different configuration than Bitcoin, with 13 seconds block times and unlimited block sizes, to facilitate this Ethereum uses a different consensus protocol. This section aims to explain the rationale behind the design of this protocol.

Sompolinsky and Zohar explain that for each new block that is created there is a window of time from the time that block was successfully created, to the time it takes for that block to propagate to the other nodes in the system. In this window of time, all the other nodes are essentially working on forking the blockchain or wasting their mining efforts, because a new block is already imminent. The bigger portion of the time the system stays in this vulnerable state, the weaker the security of the protocol. Two factors directly influence this state:

Block creation rate Assuming the propagation time to other nodes is more or less fixed, shortening the period of time between each block will increase time in the vulnerable state. For each new block, a fixed portion of time will be spent propagating the blocks, if blocks are created more rapidly this portion of time will be relatively bigger for each new block. Figure 4.6 shows this relation. The block time is divided into two periods; the red portion is the period before the block reaches the other nodes. The upper sequence shows a longer period between each block, by increasing the block creation rate in the lower sequence

– while the propagation time remains the same – more time overall will be spent in the vulnerable state.

Block size Assuming a fixed block creation rate, the time to propagate a new block to other nodes is defined by the size of the block. Increasing the size of the block will lengthen the propagation time, causing the block propagation window to become bigger relatively to the block time. Figure 4.7 shows this relation. The block time is divided into two periods; the red portion is the period before the block reaches the other nodes. By increasing the block size in the lower sequence, the propagation time increases causing the relative time spent propagating blocks in each block period to increase.

While Somopolinsky and Zohar's model is based on a network topology where two sets of nodes of equal hashing power need to propagate the blocks in between, Buterin [But13] makes an important note regarding the centralization of mining nodes of unequal hashing power, that might further strengthen this argument: Nodes that are more centralized in the network will have shorter propagation times for blocks they create, and will, therefore, be more likely to create blocks that are added to the main chain in the long run, because they spend less time waiting for blocks mined by other nodes. If these centralized nodes also have a larger hash rate than other nodes, this bias increases even more, making it very difficult for other nodes to mine on the main chain.

4.6.4 GHOST

To facilitate the differences from Bitcoin without impeding the security of the protocol, Ethereum uses a different approach for reaching consensus than Bitcoin: The Greedy Heaviest Observed Sub-Tree (GHOST) protocol. The protocol was first described by Sompolinsky and Zohar [SZ13], and has been modified slightly in the Ethereum implementation for practical reasons.

The Greedy Heaviest Observed Sub-Tree (GHOST) protocol is a variation to the Nakamoto consensus where the general block selection strategy is no longer *the longest chain wins*, but rather *the heaviest sub-tree wins* [SZ13]. The heaviest sub-tree is the tree that has the most combined computational work performed – not only on the main chain, but including valid computational work performed on closely related forks as well.

The chain selection algorithm works as follows this: From a given block height, if there occurs a split in the blockchain the *sub-tree* – e.g. the descendant blocks from the parent the split occurred at – with the most blocks generated is the one that should be selected – not just the longest chain. In the Nakamoto consensus these blocks that were deemed valid, but a part of the selected longest chain are commonly

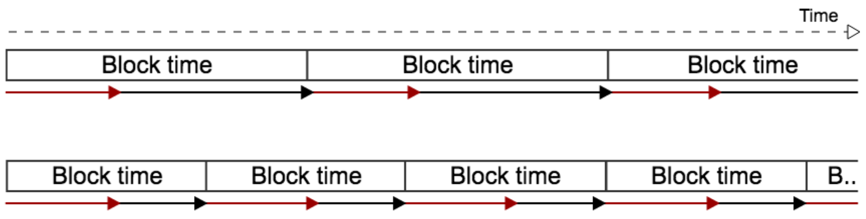


Figure 4.6: Significance of shorter block times.

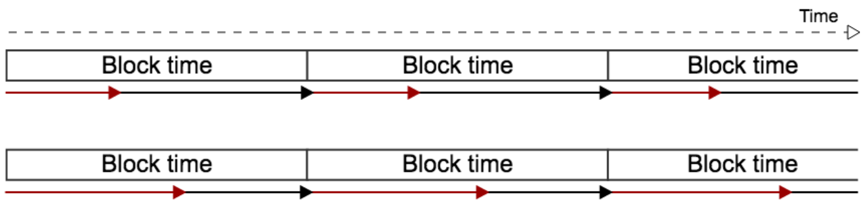


Figure 4.7: Significance of larger block size.

called *stale blocks*. In Ethereum the stale blocks are referred to as *Ommers* – the gender neutral term for an aunt or uncle – and are rewarded fractions of the block reward that the miners of blocks on the main chain receive [Woo14].

Figure 4.8 shows the distinction between Nakamoto consensus and GHOST consensus: The green boxes denote two separate subtrees A and B, of depth 4 and 3. The purple box is an *Ommer block* e.g. a stale block. In this scenario, according to the Nakamoto consensus protocol, *Subtree A* would be the valid chain e.g. the longest chain. However, in the GHOST protocol *Subtree B* is considered the *heaviest sub-tree* and therefore the correct block to be mining from is either of the lowest blocks in *Subtree B*. Depending on which of B3 or B3' gets a child first, one of them will become an ommer as new blocks are mined. Ethereum limits the depth of the heaviest-sub tree to a maximum of six ancestors, and a maximum of two ommers included in each block. The ommer has to be a direct child of an ancestor to be considered valid [Woo14] – e.g. in the analogy of family relations, "cousins" are not allowed.

The rationale behind this branch selection strategy is that in networks with high latency an honest miner – e.g. a miner compliant to the common strategy – would typically continue to mine on a block even after another valid block has appeared

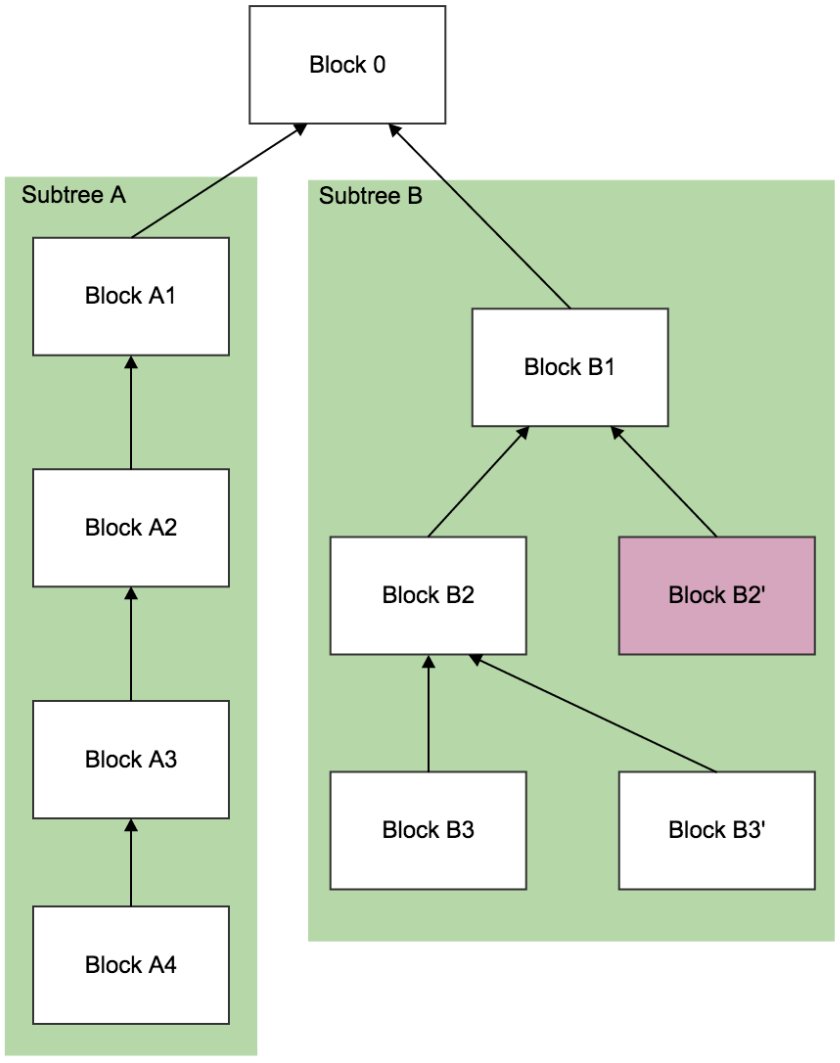


Figure 4.8: Example of GHOST consensus protocol.

because the block takes time to propagate. If the miner produces a valid block before the block reaches her side of the network – instead of wasting the computational effort of that miner – the work that was done on both blocks is added up in the branch selection strategy. This means that even poorly connected miners can contribute to the security of the blockchain by mining blocks that are off the main chain in the long run.

Sompolinsky and Zohar, the creators of the protocol, state that by utilizing this protocol the threshold of hashing power needed to perform the 51% attack in Nakamoto consensus protocol, remains the same even with high network propagation delays.

4.6.5 Minting Procedure

As is the case for all cryptocurrencies there needs to be consensus in the community that the coins minted in the system have value. For Bitcoin, this consensus evolved as more people gradually saw the potential in the currency. For Ethereum the process has been a bit different.

In the months before the Genesis block – e.g. the very first block – of Ethereum was mined, there was a pre-sale of Ether. Approximately 72 million Ether was sold in exchange for Bitcoins, launching the first state of the system so that initial stakeholders already had value in the system. By linking the value of Ether up to Bitcoins value system, the developers created an expectation of what Ether was worth.

After the initial pre-sale, a general minting rule was set in place. For each new block mined, the miner of said block gets 5 Ether. However, unlike Bitcoin, Ethereum has no upper cap on its Ether supply – the reward is not decreasing over time. Additionally, miners of ommer blocks get fractional rewards of this value. This value is created in addition to the mint of 5 Ether. Each new block therefore adds 5 Ether, plus whatever ommer block rewards are included.

For each ommer that is included in a block, the miner of that ommer gets 7/8 of the block reward. The miner of the block that is included in the main chain gets the full block reward plus, an additional 1/32 of the block reward per ommer included. The miner of the included block also gets whatever fees have been accumulated during execution.

Figure 4.9 shows the growth of the general Ether supply since the pre-sale in July 2015. The data is collected from *etherscan.io* an online blockchain explorer for Ethereum. The growth of Ether follows an approximately linear model that grows

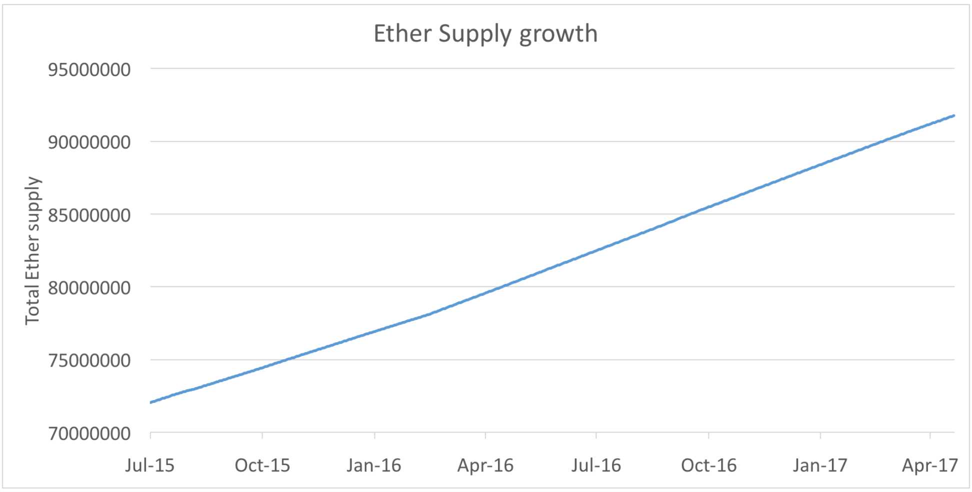


Figure 4.9: The growth of Ether since the project was started in July 2015.

monotonically. The supply growth is subject to small variations as the ommer rate changes, which depend on the state of the network in the future.

It was expected that this model for minting Ethereum would change with the future version updates, according to the Ethereum release plan [Gup15]. In fact, a final model for the minting procedure was expected to be included in the Homestead hard fork, but this model is yet to be published. The consequences this will have for the currency is unknown.

4.6.6 Block Validation

When a block has been constructed and mined correctly it is relayed to the other nodes in the network. Upon receiving the block, it needs to be validated. The block validation procedure evaluates the following points [Woo14].

Validate Ommers ommer validation includes checking the ancestry of the ommer – that it is indeed an ommer and not, for instance, a cousin – and that the header is correct e.g. the proof of work it correct.

Validate transactions All transactions structure and signatures must be valid.

Apply block reward and fees The miner of the block gets the whole block reward, ommer reward and transaction fees. Ommers receive ommer rewards.

Verify the state The Merkle Patricia Tree roots for the system state should be correct; this includes executing all transactions in the block and updating the trees.

Verify block nonce The block Proof of Work needs to be correct.

4.7 Scaling Ethereum

Since Ethereum is newer than Bitcoin it has yet to reach the same level of adoption. However, with no fixed limitations for block sizes, and 40x faster block rate from Bitcoin it should seem that Ethereum would quickly face a scalability problem regarding the overall size of the blockchain. Indeed, if nodes were required to store the entire blockchain to be able to mine efficiently, this would pose a problem. According to this recent forum discussion [Whab], the size of the full blockchain has already surpassed 140GB since the launch in 2015. However, not every node needs to store the whole blockchain in Ethereum. Only so-called *archive nodes* store the complete blockchain. Because of the stateful protocol Ethereum utilizes, keeping track of the entire blockchain is not necessary for validation – like it is in Bitcoin. By keeping track of a reasonable portion of the last blocks, the world state tree of the longest chain can be verified without needing to store the rest of the blockchain. Partial storage occurs after what is commonly called *State Tree Pruning*, and severely cuts down the storage necessary to be a full mining node. Depending on which client implemented the current size can be limited to less than 30GB [blob].

4.7.1 Light Clients

Light client support for Ethereum is currently underway, but not yet released [Lig]. The current proposal is to have a sub-protocol in Ethereum for light clients so they can verify transactions without keeping track of the blockchain. This is possible because Merkle Patricia Trees allow for simple verification by requesting branches of the tree from other full nodes (see Section 2.2.3). As discussed in section 4.6.2, the mining algorithm also supports verification of light nodes.

Regarding security of the light clients, it seems that in Ethereum one would be less dependent on trusting other verifying nodes, compared to Bitcoin. Because of the verification of the state tree in the block header any transaction could be verified directly by the light node by requesting the correct branch in the state tree. Double spending can, therefore, be detected by light clients without having to store the entire blockchain as they would need in Bitcoin.

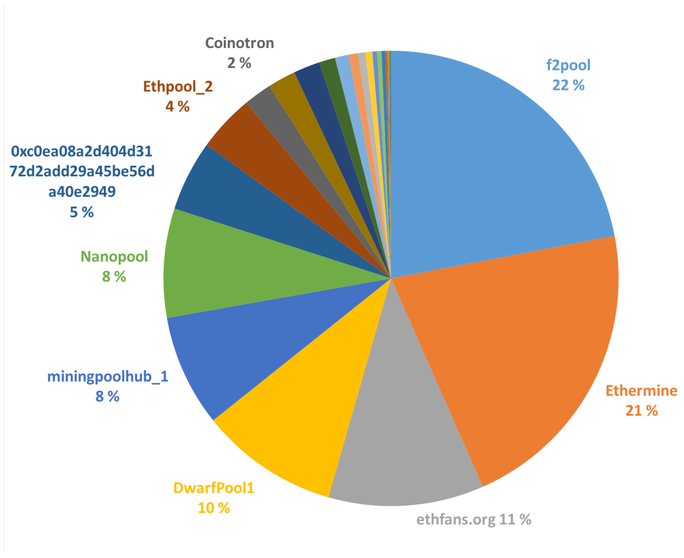


Figure 4.10: The current pool distribution in Ethereum the last seven days (28. May 2017).

4.7.2 Status on Mining

The current status on mining Ethereum is that GPU mining is profitable, and ASICs are not prominent in the community [doc]. However, despite the measures taken in Ethereum's protocol to dissuade miners from forming mining pools the community has formed into a structure much like Bitcoins. Figure 4.10 shows this development, the data is collected from etherscan.io.

4.8 Security

Being blockchain based – and thus relying on an eventual consensus mechanism for agreement on transactions – like Bitcoin also Ethereum is susceptible to the 51% attack and is, therefore, relying on a majority of honest miners. Additionally, two other known attacks are discussed in this section.

4.8.1 The Long Range Attack

The Long Range Attack [But14] is an adapted version of the 51% attack that can be quite serious for Ethereum. The Long Range attack increases the efficiency of the 51% attack by utilizing the Turing-complete programming capabilities of the system to slow down other validating nodes.

The idea of the attack is that a malicious actor can force other miners to execute computationally exhaustive contract programs on the main chain. The programs can include computational trap-doors, making them easy to execute for the attacker, but hard for the other nodes. By pretending to execute these contracts in real blocks, but instead running the trap-door function, the attacker can get a head start on mining. This attack is costly on the main chain in terms of gas, but the main point is that the attacker can mine new blocks on a fork further down the blockchain and essentially rewrite the blockchain while the other nodes are stuck executing these transactions – hence the name of the attack.

4.8.2 Uncle Mining

Another weakness that has been discussed that is Ethereum-specific is called Uncle Mining – or Ommer Mining in line with our terminology in this thesis. The attack is described by Sergio Lerner on his blog [Ler]. This attack resembles the Selfish Mining Attack because it is a change of strategy that increases the profit of the miner – as such it does not break the security of the system, but might cause problems in the network in terms of stability and efficiency.

The uncle mining attack involves withholding valid blocks until the blocks have become ommers. Interestingly, Lerner demonstrates that under certain assumptions this strategy is more profitable both for the deviant miner and the honest miners – this is because both ommers and valid blocks are rewarded for ommer inclusion. However, the general quality of the network goes down, causing network latency and eventually dropping the hash rate.

4.8.3 Security of Programming in Ethereum

Another security aspect of Ethereum should be noted. By having an environment for contract creation, users can program their own smart contracts and decentralized applications. Because of this Ethereum opens up a much larger attack surface. Essentially, Ethereum contracts are subject to all the same problems that regular computer programs suffer from when it comes to program coding. This includes code bugs, logic faults, implementation errors, and so forth that could be exploited for malicious intent by other nodes in the network.

Security issues related to this is off-protocol for Ethereum, but for non-technical users, this may appear a subtle difference. In fact, the world has already seen an instance of an Ethereum contract code being maliciously exploited. It is known as the Decentralized Autonomous Organization (DAO), and described by Buterin in this blog [But16]. The DAO was a smart contract implemented in Ethereum with an unintentional loop hole in the contract code. The loop hole was exploited by a malicious actor, causing the people that had invested in the DAO to lose a significant amount of their investment. Interestingly, the DAO contract did exactly what it was programmed to do – but the general understanding of the code among the users was different.

The DAO caused controversy about the legitimacy of the exploited loop hole; whether the "code was law" or the intent of the contract was the most important. The result of this dispute was a vote where all nodes could participate, resulting in a hard fork initiated by the Ethereum foundation that reverted the changes caused by the attacker. However, a large portion of the party voting against this hard fork refused to conform to the change – resulting in a permanent network split in the blockchain. As a result, there are currently two forks of Ethereum that are being mined and used today, by two different communities: Ethereum and *Ethereum Classic* [Ethb]. The future development of the Ethereum Classic project is unpredictable. The current situation is that both Ethereum and Ethereum Classic is considered valuable on the exchange market, admittedly at different prices. Incidentally, anyone who originally had value in the system before the split now has value in both systems, because the two systems are effectively copies of each other up until the point of the fork.

Chapter 5

Comparison and Discussion

Chapter 3 and Chapter 4 have provided a thorough technical description of Bitcoin and Ethereum. As was described in the introduction of this thesis, the primary goal of this thesis is to perform a comparative analysis of Bitcoin and Ethereum in light of how they relate to the three technical problems of Bitcoin: Computational Waste, Concentration of Power and Ambiguity, and determine whether Ethereum suffers from the same problems as Bitcoin. The comparison will be the topic of this chapter.

Additionally, we highlight two points of comparison that are added to this chapter at the end: Scalability and Security. They do not relate to the original problems defined by Micali [Mic16], but have nonetheless been raised as potential issues for cryptocurrencies in general, and have therefore been included in this thesis.

The nature of decentralized cryptocurrencies is that every transaction and block that is published is available to every node in the system. Both Bitcoin and Ethereum have live blockchains that have been running for several years, and the data accumulated is accessible to the public. Open web applications provide statistical information derived from the public blockchains. In addition to the technical discussion statistical data gathered from the blockchains is synthesized to enlighten the discussion and help determine if the assumptions made in the technical comparison align with what is happening in the networks.

An important note is that the information gathered from the network is constantly evolving. Many of the points made in this chapter only constitute a snapshot image of the situation today, which are subject to change daily. The web applications used for data collection are *etherscan.io* for Ethereum and *blockchain.info* for Bitcoin.

5.1 Comparative Summary

To give a basis for the following discussion, Table 5.1 gives an overview of the most relevant technical similarities and differences between in two systems. The first three

Table 5.1: Summary of relevant technical differences

	Bitcoin	**Ethereum**
Maximum block size	1MB	Flexible limit
Target block time	10 min	13 sec
State	Stateless	Stateful
Consensus mechanism	Proof of work	Proof of work
Consensus protocol	Nakamoto consensus	GHOST
Mining algorithm	Hashcash	Ethash

rows are related to the configurations of the blockchain. Importantly, Ethereum utilizes a smaller block time than Bitcoin and has no fixed block size. Additionally, Ethereum's blockchain is stateful, while Bitcoin is not.

The last three rows concern the methods for reaching distributed consensus. Both systems utilize a proof of work consensus mechanism – which creates a common ground for discussion. However, the consensus protocol and algorithm used in Bitcoin and Ethereum is different. Recall the details of this, from Sections 4.6.4 and 3.5.2.

5.2 Computational Waste

The first technical problem listed for Bitcoin is *the computational waste* of the proof of work protocol. Waste is unprofitable spending – spending for no purpose. The general amount of computation power necessary from Bitcoin miners today is several times that of the top 500 supercomputers in the world [Mic16].

5.2.1 What Level of Energy Consumption Can We Expect Anyway?

Obviously, running an equivalent of a financial system is bound to have some costs. Questions that arise when considering how wasteful Bitcoin or Ethereum is are:

Is the existing system in use today less expensive? What does the current financial industry consume in terms of energy? When considering how much energy is put into running banks worldwide today the cost of Bitcoin might not be as expensive comparably.

Can the same task be done in another way with less overhead? Certainly, trusted third party systems like PayPal and Visa provide similar services as a cryptocurrency. At what level of energy consumption they operate is difficult to measure.

Drilling down, the consensus mechanisms in both systems serve two purposes: First, to solve the *decentralized consensus problem* (see section 3.1.3). Second, to mint new coins in the system. Strictly speaking, this process involves performing the following tasks in a distributed manner:

1. Verifying the blockchain.

2. Creating new blocks.

3. Minting new coins.

Point one and two are tasks where the computation, in general, does not have to be costly. Point three, however, may be another matter: Because cryptocurrencies mint their own coins, there are some arguments *for* that process being, if not wasteful, at least expensive. For instance, historically, *gold* was considered valuable because it is hard to find and heavy – and therefore rare and difficult to steal. If a cryptocurrency was inexpensive to mint, would it still be considered valuable? Is the value of the cryptocurrency somehow linked to the expenditure of energy per unit? Section 5.2.4 looks more closely into this matter.

5.2.2 The Technical Differences and Similarities

The proof of work protocol constitutes the consensus mechanism of both Bitcoin and Ethereum. Arguably, the proof of work protocol is inherently wasteful because it involves an expensive cost function which is at the root of the mechanism. Therefore the purpose of this chapter is to establish *how wasteful the implementation of the protocols are when compared.*

There are several factors that can influence the energy consumption; typically what kind of hardware is being used, how much cooling is needed, the hashing algorithm and so forth. Quantifying the exact – or even estimating – the energy needed to mine is not an easy task. As a general mode of comparison, we use the hash rate of the network, e.g. how many hashes per second is performed overall in the network. This gives an estimate of how much mining is being done in either system, which should be roughly proportional to the consumed energy in either of the systems.

Looking back at what we have learned from the previous chapters, there are two parts of the consensus mechanism: the consensus protocol and the mining algorithm. We will start by comparing the mining algorithms.

Hashcash versus Ethash

Hashcash and Ethash are the mining algorithms for, respectively, Bitcoin and Ethereum. For both systems, the end goal of both algorithms is to find a partial pre-image of a hash function. There is an upper bound to the expected time it should take to solve this challenge, so depending on the hash rate in the system, the difficulty of the pre-image is adjusted. Which factors influence the hash rate, and how Ethereum and Bitcoin compare are discussed in the following.

The Algorithm Construction The Hashcash algorithm was not constructed for any other purpose than as a cost function [B⁺02]. Ethash, however, was designed years after Hashcash and – having observed the weaknesses of the Hashcash – was therefore designed to be resistant to improvements in hardware. The bottle neck for the Hashcash algorithm is the hashing function. In Ethash the bottleneck is I/O operations, which most personal computers are already specialized for (see Section 4.6.2 for more detail).

Hardware Improvements Utilizing special hardware – e.g. ASICs or Graphical Processing Unit (GPU) mining – will increase the network hash rate. Because of the ASIC resistant property of Ethash, the network hash rate in Ethereum should, therefore, be significantly lower when compared to Hashcash.

Mining Participation The more nodes participating in mining, the higher the hash rate overall in the network will be. This holds for both algorithms.

Nakamoto Consensus vs GHOST

The Nakamoto Consensus is the consensus algorithm of Bitcoin. Recall from Section 3.5.2, the Nakamoto consensus rewards only blocks that are a part of the longest chain, thereby disregarding any computation performed that results in stale blocks. Comparably, Ethereum's GHOST protocol rewards both blocks that are a part of the main chain and ommer blocks and weighs their combined computational effort when picking a leg in the fork.

Consequently, Ethereum's consensus algorithm better utilizes the computation of all the nodes in the network, compared to Bitcoin's consensus algorithm, by wasting less computation when the blockchain forks.

5.2.3 Quantifying the Difference

By comparing Figure 5.1 and Figure 5.2 there is an obvious gap in hash rate between the two systems. They both have an increase in hash rate over the last few years, but while Bitcoin is operating on approximately 4,5 million TH/s, Ethereum only

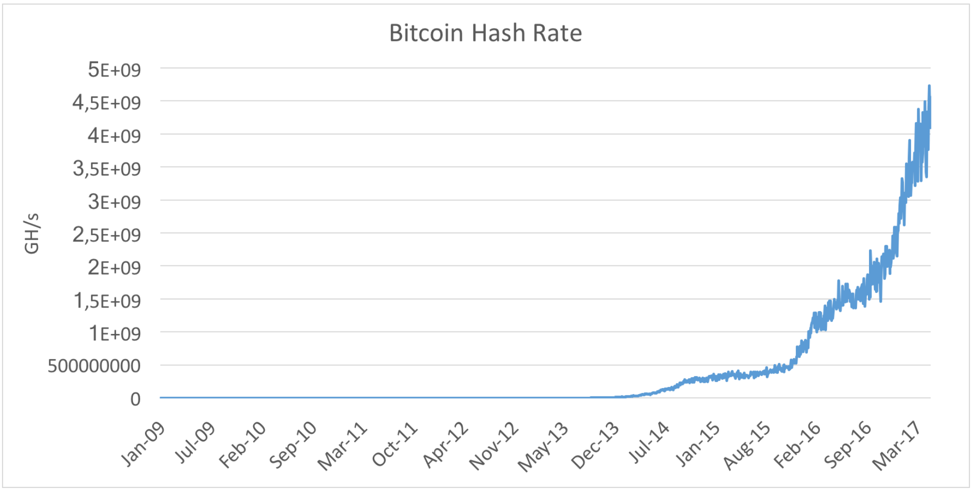


Figure 5.1: The historical hash rate of Bitcoin.

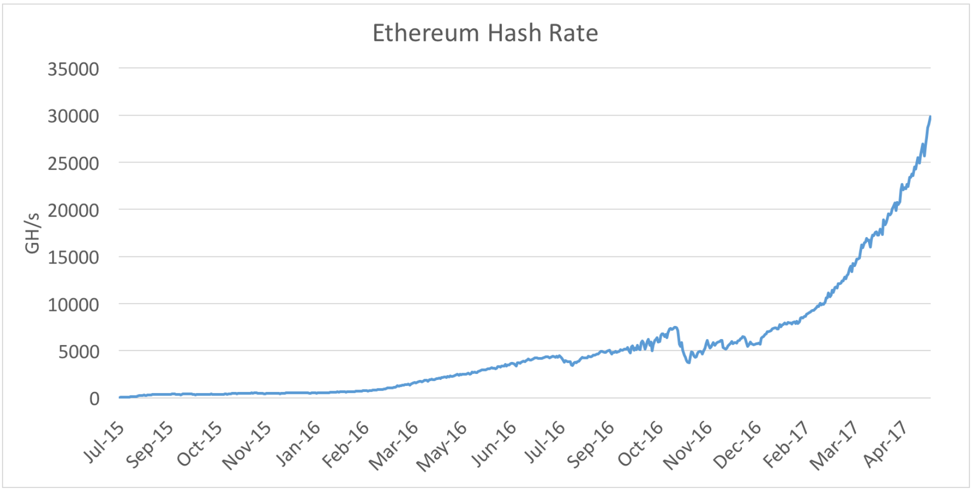


Figure 5.2: The historical hash rate of Ethereum.

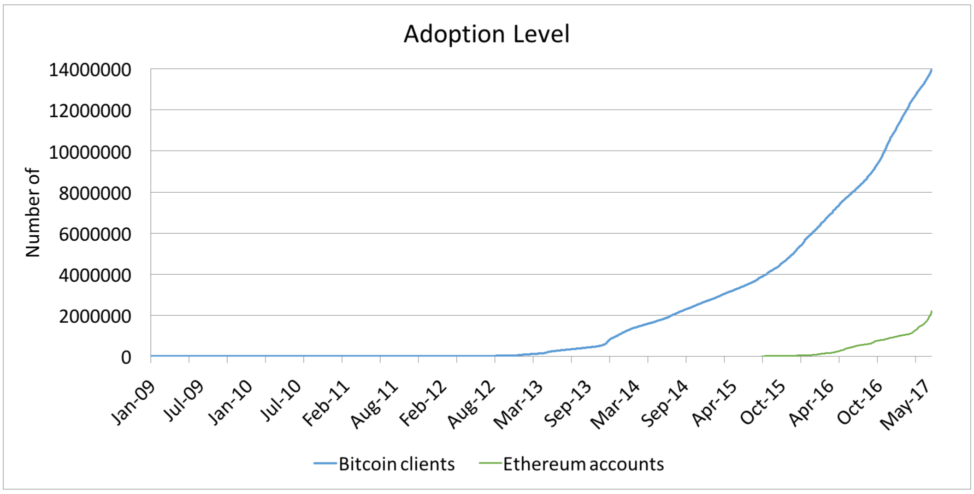


Figure 5.3: An estimate of the level of adoption for Bitcoin and Ethereum.

amounts to about 30 TH/s. Simply comparing the situation as of now, Ethereum has a considerably lower hash rate than Bitcoin. However, influencing factors in this comparison need to be mentioned:

Period of operation While Bitcoin has been operational since 2009 i.e. eight years, Ethereum has only been around since the summer of 2015, i.e. two years. While it is not entirely fair to compare the two in terms of years of operation – because there has been a great increase in interest for blockchain technology since 2009 – it should be mentioned that due to developments in the following two points, Ethereum might not come out of this comparison as well after a few more years of operation.

Hardware development Although Ethash has been designed to be ASIC resistant, other attempts at creating resistant algorithms have proved unsuccessful over time, and specialized equipment has been developed (Scrypt – the mining algorithm of LiteCoin – is an example of this). To put the effects of this equipment in perspective the first Bitcoin ASIC miners were manufactured in 2013 – from Figure 5.1 it is clear that this was when the hash rate started to take off.

Level of adoption Bitcoin is better known than Ethereum, primarily because it has been around longer. We know that the hash rate increases with the number of active miners. The hash rate might be higher because of this factor. Figure 5.3 gives an estimate of the level of adoption for Bitcoin and Ethereum. The data is based on the number of accounts for Ethereum[1], and the number of downloaded clients for Bitcoin[2]. Users can, of course, have several accounts in Ethereum, and several Bitcoin clients so this is, in general, a rough estimate.

5.2.4 Value for Effort

By observing the four graphs in Figures 5.1, 5.2, 5.4 and 5.5, it may appear that there is a correlation between the market value of the currency, and the network hash rate. There are two scenarios for this relation that seem equally likely: Either the community perceives a higher value of the currency because of the effort to create the currency increases, or the hash rate goes up because more people invest in mining when the value goes up.

If the first case holds true, i.e. the value of the currency increases because the network hashing power goes up, this might indicate that the value of the currency is linked to the computational effort put into the system – that this is how the value of the currency is created. If this is the case, the computational waste is, in fact, a critical factor in the cryptocurrency, because without the waste people might not perceive the currency as being valuable – and therefore not willing to trade for it.

However, if it is the other way around, the increased difficulty of the system simply serves to slow down the blocks and the added computational effort performed by all the nodes not winning the race for creating new blocks is wasted. The assumption is then that the price of the currency is based solely on other social and economical factors (for instance market supply and demand).

What Is the Relation?

To try and find a more substantial answer to the question of whether the price of cryptocurrencies is linked to the network hash rate we have combined the statistical data from the market value and the hash rate, $y = \frac{hashrate}{USD}$, Figures 5.6 and 5.7 illustrate this. Importantly, note that the y-axis of the Bitcoin graph has been adjusted after December 2013 to provide some readability for the early values. For reference it was around December 2013 ASICs became prominent in the mining community, causing a spike in the general network hash rate.

What we might expect to see from these graphs, should the hypothesis be that: "the price of the currency is linked to the computational waste" hold, more or less a straight line, where k is a constant:

$$\frac{\text{hash rate}}{\text{USD}} = k$$

[1] https://etherscan.io/chart/address
[2] https://blockchain.info/charts/my-wallet-n-users

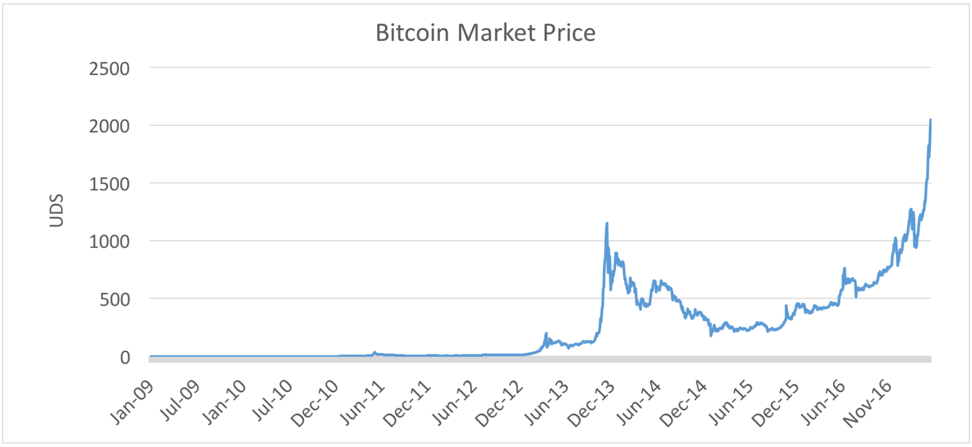


Figure 5.4: The historical market price for Bitcoin.

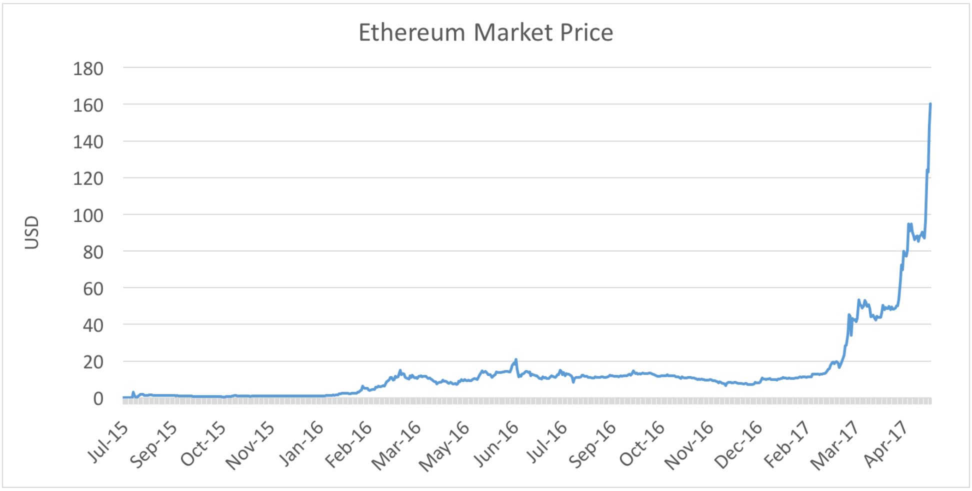


Figure 5.5: The historical market price of Ethereum.

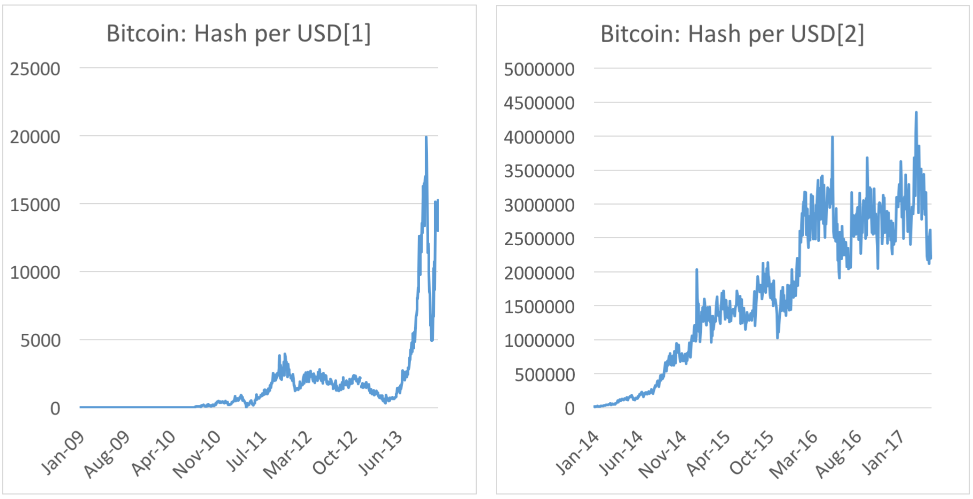


Figure 5.6: The Bitcoin relation for number of hashes per USD.

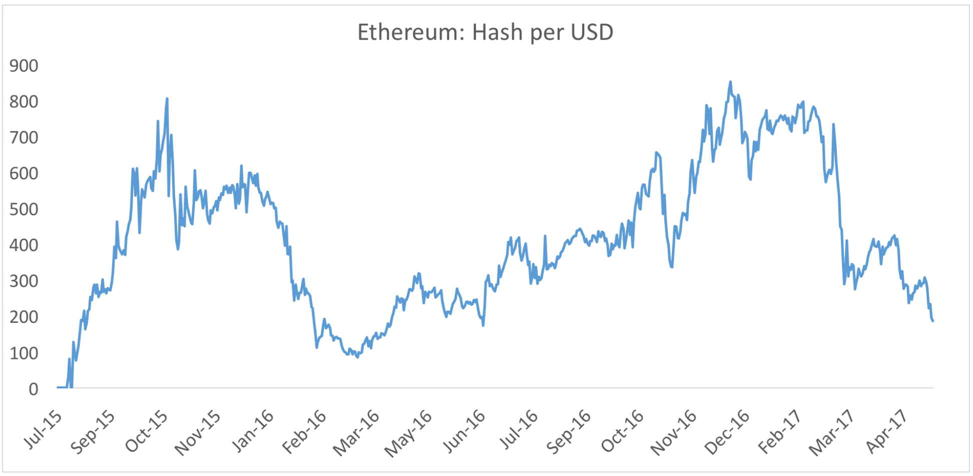


Figure 5.7: The Ethereum relation for number of hashes per USD.

However, from the graphs it becomes apparent that the value is influenced by more than this. In the case of Bitcoin, there are shorter periods where the graph is close to straight: {Jan-12, Jan-13}, {Feb-15, Mar-16} and {Mar-16, Apr-17}, but even these periods are volatile. For Ethereum it is hard to find any pattern at all. This might be due to the short period of operation, or that there is not a strong link between the value of the currency and the computational effort.

5.3 Concentration of Power

The second technical problem relates to the tendency towards organizing into groups – mining pools – over time. Miners join mining pools for several reasons:

Lower variance on income The primary reason for miners to join mining pools is the high variance of expected returns from mining. Mining is essentially a lottery where the reward is high when you win, but you receive nothing before you win. In a mining pool, miners collaborate to find a block and then share the reward. The miners are rewarded according to their level of hashing power, so the overall reward becomes approximately the same for each miner over time, but the income is less volatile.

Size of blockchain Most mining pools are organized by a mining operator that performs the task of assembling blocks, meaning that the miners do not have to store the entire blockchain on their devices, but only focus on solving the block. There are instances of pools with different structures as well, such as p2p pools [P2P]. P2p pools have no operators, but share rewards.

Mining equipment Specialized mining equipment can be expensive to buy and run. In mining pools, miners can split the costs of mining, i.e. buying ASICs, renting space for the hardware, costs of electricity.

5.3.1 Why Are Mining Pools a Problem?

Why is the concentration of power considered a problem in a blockchain based cryptocurrency? Most of the factors relate to what the *original intentions* of Bitcoin and Ethereum were, and that the system no longer aligns with this vision.

No longer a distributed system One of the main value proposals for cryptocurrencies is the distributed property of the system. When mining pools control large portions of the network, this property is no longer as prevalent.

Trust The blockchain is supposed to be validated by all mining nodes so that no trust is necessary between the nodes. In centralized mining pools, only the pool operator performs this task, leaving us with what is, in essence, an expensive trusted third party model, where the miners trust their operators to act honestly.

Fairness Nodes that are clustered together in a mining pool have a greater chance of producing blocks that become a part of the main chain simply because the propagation time is smaller in the cluster. Section 4.6.3 discusses this effect.

Impact on security Concentration of power makes some attacks feasible for proof of work schemes. For instance, acquiring 50% hashing power in the network may be considered infeasible for one actor, but in a network where two or three mining pools control >50% of the computational power between the pools, this may be a matter of negotiation between the pools.

5.3.2 The Technical Differences and Similarities

Since both Ethereum and Bitcoin rely on block rewards from successful proof of work for minting their intrinsic currency, they both are both subject to this problem. Factors that influence the severity of concentration of power, where Ethereum and Bitcoin differ, are:

Stateful blockchain Ethereum full mining nodes do not need to store the entire blockchain to validate transactions – like they do in Bitcoin. This implies that the hardware storage cost of mining independently is the same as in a mining pool.

Consensus protocol Arguably, Ethereum's GHOST protocol handles the effects of centralization better than Bitcoin. By rewarding ommer blocks as well as blocks on the main chain, the positive networks effects from being a part of a mining pool are not as prevalent. This reduces the advantage in mining for clustered nodes on the main chain.

Overall hash rate As the difficulty increases, the more difficult the blocks are to find and the lower the expectancy is for finding a valid block. In other words, the expected time in between each block reward becomes longer when the difficulty goes up. Bitcoin has a high hash rate, meaning miners might tend more heavily to mining pools.

Quantifying the Difference

To quantify the levels of centralization in Bitcoin and Ethereum, we can measure the size of mining pools that are currently active. Unfortunately, we have little access to historical data in this development, and the pools are prone to daily change. The data available is based on the number of blocks accepted to the main chain, and the addresses they are relayed from.

Interestingly, even though in theory Ethereum has some theoretical advantages over Bitcoin when comparing the technical aspects of the protocols regarding centralization – the degree of centralization for Ethereum is higher than for Bitcoin. This is apparent from looking at the mining pool distribution charts in Figures 5.8 and 5.9. Recall from Section 3.7.1 that when more than half of the network hash rate is

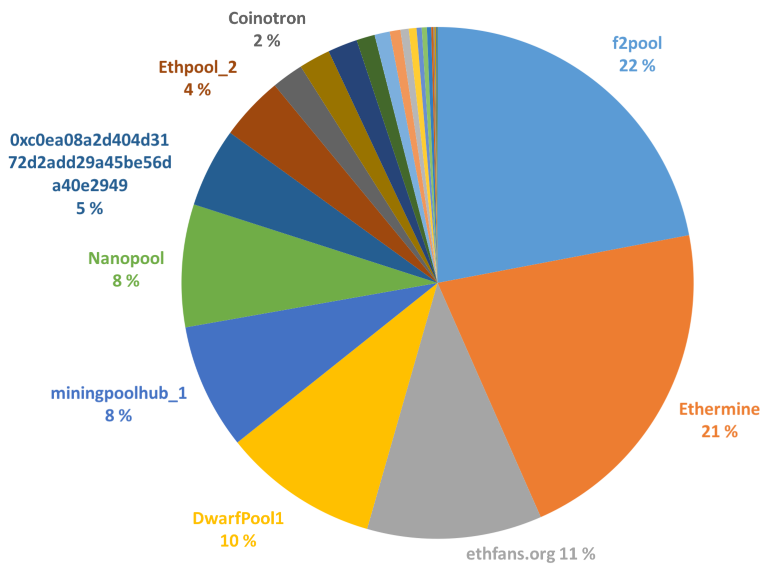


Figure 5.8: The current pool distribution in Ethereum the last seven days (28. May 2017, repeated figure from Chapter 4).

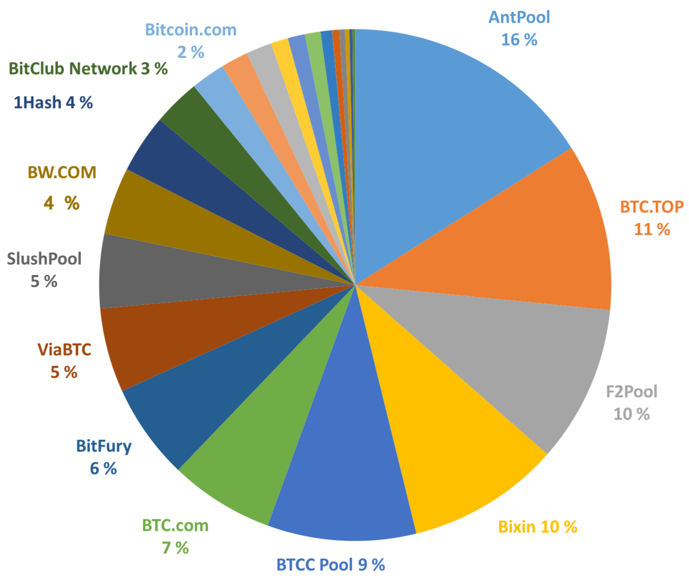


Figure 5.9: The current pool distribution in Bitcoin the last seven days (28. May 2017, repeated figure from Chapter 3)

controlled by a malicious actor, the blockchain is no longer secure. In Ethereum only three pools would have to collaborate to achieve this. Comparably in Bitcoin, where five pools would have to collaborate.

Effects of High Variance

The primary reason for miners to join mining pools instead of mining independently is the high variance related to mining rewards. When the hash rate reaches as high levels as seen in Bitcoin today, the expected waiting time for return on investment might be years if one is not part of a mining pool [NBF+16]. In fact, in Bitcoin mining pools are almost the only way for mining to be profitable today: Because the market value changes rapidly and hardware improvements make the competition harder – with a years waiting time the original investment might be obsolete before miners get any return.

5.3.3 Security Issues With Large Mining Pools

Over the course of time, at least for Bitcoin, there has been periods when one pool has been close and even reached >50% of the hashing power in the system [NBF+16]. An interesting counter-effect to this situation has been a culture among miners to migrate to smaller pools when one pool has become too big and to encourage others to do the same [Min]. This is, of course, voluntary for miners, but it seems many choose to follow this pattern.

The incentive for doing leaving large mining pools is that one pool having that high level of control over the blockchain may cause users to lose trust in the security of the system – we already know that double spending attacks are feasible for miners controlling more than half of the hashing power. However, even though the blocks are being relayed by different pools, we know very little about the operators of the pools. It is possible that one operator may, in fact, control several pools.

5.4 Ambiguity of Transactions

The last technical problem relates to the time users have to spend waiting for transactions to be confirmed. The general rule of thumb for a transaction to be considered finalized in Bitcoin is to wait until six consecutive blocks have been added after the block that included the transaction is added to the blockchain.

The exact number of block confirmations required for Ethereum for the same level of security as Bitcoin is not a trivial question, due to the difference in block time and consensus protocol. Buterin [OnS] discusses the matter with no final answer to the

Table 5.2: Average confirmation time of transactions over the last year.

Bitcoin average	56.2 minutes
Ethereum average	1.4 minutes

question, a discussion on Ethereum stack exchange [3] puts the exact number of blocks in the range of 5-500 blocks depending on the value exchanged. For consistency, we will assume that the general rule for Bitcoin also holds for Ethereum, but with a note that this might impact the level of security for the transaction.

As is evident from Figures 5.10 and 5.11, and Table 5.2. Ethereum's transactions are finalized after comparably short time to Bitcoin – assuming a six block confirmation time. This is because of the different block time configuration of Ethereum.

5.5 Scalability

An additional problem that has been discussed for Bitcoin in several instances is the scalability of the system. There are two key issues with the scalability of Bitcoin:

The size of the blockchain Recall that for Bitcoin miners to be able to validate that transactions are not double spending, they need to store the entire blockchain. This means that the necessary storage space for Bitcoin is continuously growing, forcing miners to invest in storage.

The maximum blocksize The 1MB maximum size of the blocks puts an upper cap on the number of transactions that can be processed by the miners. When compared to other payment systems, for instance, Visa, this cap is much too low. Putting this in perspective, Narayan et al. [NBF+16] estimates that Visa processes about 2000 transactions per second. Comparably, Bitcoin at maximum capacity can only process about seven transactions per second.

What makes scalability worse is that the second problem influences the first. Disregarding the security implications of increasing the maximum block size (see Section 4.6.3) – such an action would also severely increase the growth of the blockchain.

Ethereum handles both these problems efficiently in three ways: Firstly, (recall from Section 4.7) due to Ethereum's stateful blockchain, mining nodes can validate and mine successfully without downloading the entire blockchain. Only a few *archive*

[3] https://ethereum.stackexchange.com/questions/319/what-number-of-confirmations-is-considered-secure-in-ethereum

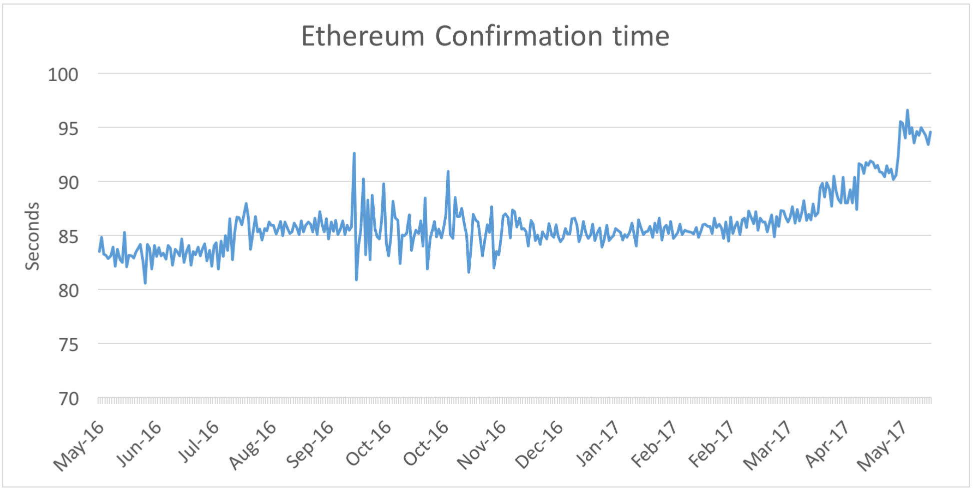


Figure 5.10: Ethereums daily average confirmation time of transactions over the last year.

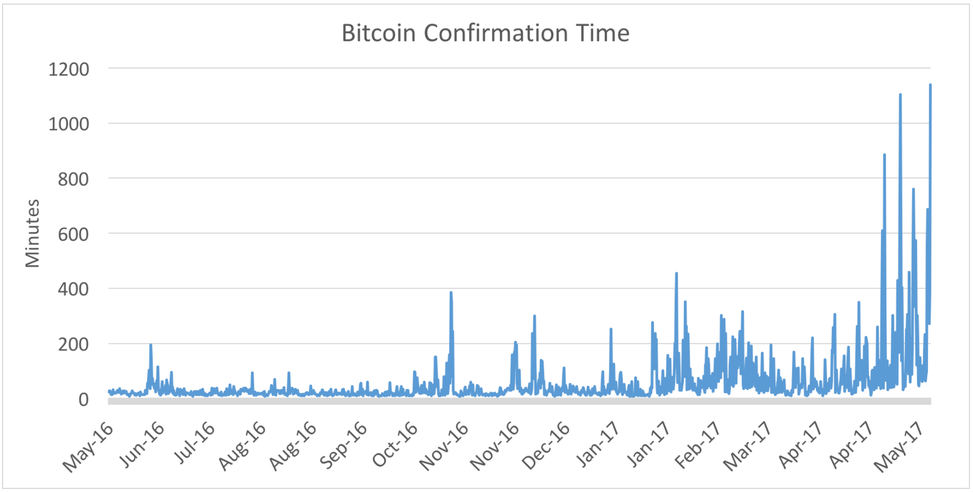


Figure 5.11: Bitcoins daily average confirmation time of transactions over the last year.

nodes store the entire blockchain for future reference, but storing some few hundreds of the last blocks in the chain is sufficient for validation and consensus. This is not possible for Bitcoin because all previous transactions need to be known to recognize attempts at double spends. Secondly, the size of Ethereum blocks is flexible, meaning that in periods of high usage the blocks will accommodate the increase. Finally, with the 13 second block time of Ethereum the size of the blocks need not be as large, compared to Bitcoin, to accommodate more transactions overall.

5.6 Security

From what we have seen in Chapter 3 and 4 both Bitcoin and Ethereum are susceptible to the 51% attack, although the attack would be incredibly expensive in terms of the hash rate required to execute the attack. Determining which is more secure regarding how expensive the 51% attack is to execute is hard because we are not aware of how much one hash operation costs in either system. However, in terms of the general order of hash rate in either system, achieving 51% as a single person can be considered infeasible.

Should mining pools be taken into consideration, the feasibility of the 51% attack might, in fact, be a question of negotiation between pool operators. From Section 5.3.2 we know that this would only require three pools to cooperate in Ethereum, and five in Bitcoin e.g. Ethereum is more vulnerable. However, as mentioned in Section 5.3.2 the mining pool distribution is subject to change daily, and the snapshot image presented does not capture this.

Additionally, Ethereum is vulnerable to the long range attack (see section 4.8.1. This attack requires a lot of value in the system to execute but exposes a threat of anyone rewriting the blockchain – which undermines the purpose of the system. Because of these factors, Bitcoin might be considered more secure than Ethereum.

Less serious attacks which do not break the system, but influence the profitability and fairness of the mining process, include the selfish mining attack (see Section 3.7.2) for Bitcoin and uncle mining (see Section 4.8.2) for Ethereum. Quantifying whether one of these provides a more significant advantage than the other is out of scope for the thesis, but it is sufficient to say that neither Bitcoin or Ethereum provides a perfect solution.

Chapter 6

Conclusion

To conclude this thesis we include a summary of the thesis and proposals to future work.

6.1 Summary

The main research objective of this thesis is to give a picture of how well newer cryptocurrencies handle known issues with Bitcoin. Instead of focusing on several different cryptocurrencies Ethereum has been chosen as the candidate for comparison and acts as a proxy the others.

To perform the comparison, a parallel basis for the technical functionality of both systems have been established. Chapter 3 provides an in-depth explanation of the different aspects of Bitcoin and lays the groundwork for the technical problems that are the basis for the comparative analysis. Chapter 4 provides a detailed description of Ethereum's functionality, with a focus on the factors that relate directly to the technical problems of Bitcoin. Chapter 5 examines the information laid out in the previous chapter and analyzes the significance of these likenesses and differences in light of the known problems of Bitcoin. The analysis is enlightened by statistical data collected from the blockchains of Bitcoin and Ethereum, to enlighten the discussion.

The thesis has found that because both systems are based on the proof of work consensus mechanism, they both suffer to some extent from the same problems. Ethereum has taken measures to minimize some of the negative sides of proof of work, which has so far proven effective in the case of computational waste (see Section 5.2) and ambiguity of transactions (see Section 5.4) – but less effective in the case of concentration of power (see Section 5.3). However, it is important to remember that Ethereum has only been around about a third of the time that Bitcoin has. While the adoption for Ethereum has happened faster than for Bitcoin, because blockchain technology has become widely popular, the problems that have arisen for Bitcoin took time to emerge – for Ethereum other significant problems could emerge over time as the system evolves.

6.2 Future research

Over the last few years, a score of new kinds of cryptocurrencies has been launched. As a proposal for future work within this space of research conducted in this thesis an important task will be to categorize these types of currencies. Although Ethereum is used as a proxy for them in this thesis, they can differ widely in scope, functionality, and design. Establishing some common ground for the different systems will be a good starting point for finding their weaknesses and strengths for future deployments.

Additionally, proposals to different types of consensus mechanisms have been introduced for other cryptocurrencies recently. The consensus mechanism is the key source of security for the blockchain and the weakness of Bitcoin and Ethereum. It is important that the security and functionality of these are established on a formal level.

References

[Acc] Account types, gas, and transactions — ethereum homestead 0.1 documentation. http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html?highlight=gas. (Accessed on 06/12/2017).

[B$^+$02] Adam Back et al. Hashcash-a denial of service counter-measure. ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf, 2002. (Accessed on 06/12/2017).

[Bita] Bitcoin core. https://bitcoin.org/en/bitcoin-core/. (Accessed on 04/30/2017).

[Bitb] Bitcoin wiki. https://en.bitcoin.it/wiki/Main_Page. (Accessed on 04/30/2017).

[BLN16] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. *Dual EC: A Standardized Back Door*, pages 256–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[Bloa] Block size limit controversy - bitcoin wiki. https://en.bitcoin.it/wiki/Block_size_limit_controversy. (Accessed on 06/12/2017).

[blob] blockchain - what are the ethereum disk space needs? - ethereum stack exchange. https://ethereum.stackexchange.com/questions/143/what-are-the-ethereum-disk-space-needs. (Accessed on 05/21/2017).

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[BR12] Elaine Barker and Allen Roginsky. Recommendation for cryptographic key generation. *NIST Special Publication*, 800:133, 2012.

[But] Vitalik Buterin. Dagger: A memory-hard to compute, memory-easy to verify scrypt alternative. http://www.hashcash.org/papers/dagger.html. (Accessed on 05/21/2017).

[But13] Vitalik Buterin. Ethereum white paper: a next generation smart contract & decentralized application platform. *www3. ethereum. org Nick Szabo, Formalizing and Securing Relationships on Public Networks, http://szabo. best. vwh. net/formalize. html*, 2013.

[But14] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work - ethereum blog. https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/, 2014. (Accessed on 06/02/2017).

[But16] Vitalik Buterin. Critical update re: Dao vulnerability - ethereum blog. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/, 2016. (Accessed on 06/02/2017).

[Cha92] David Chaum. Achieving electronic privacy. *Scientific american*, 267(2):96–101, 1992.

[Cry] Crypto-currency market capitalizations. http://coinmarketcap.com/. (Accessed on 05/30/2017).

[Dai98] Wei Dai. b-money. http://www.weidai.com/bmoney.txt, 1998.

[Dan13] Quynh Dang. Changes in federal information processing standard (fips) 180-4, secure hash standard. *Cryptologia*, 37(1):69–73, 2013.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. *RIPEMD-160: A strengthened version of RIPEMD*, pages 71–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Des] Design rationale · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki/Design-Rationale. (Accessed on 04/17/2017).

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[doc] Mining — Ethereum Homestead 0.1 documentation. The ethereum launch process - ethereum blog. http://ethdocs.org/en/latest/mining.html. (Accessed on 06/03/2017).

[Dry14] Thaddeus Dryja. Hashimoto: I/O bound proof of work. https://pdfs.semanticscholar.org/3b23/7cc60c1b9650e260318d33bec471b8202d5e.pdf, 2014. (Accessed on 06/12/2017).

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[Etha] Ethash design rationale · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki/Ethash-Design-Rationale. (Accessed on 06/12/2017).

[Ethb] Ethereum classic. https://ethereumclassic.github.io/. (Accessed on 06/02/2017).

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[Gal13] Patrick Gallagher. Digital signature standard (dss). *Federal Information Processing Standards Publications, volume FIPS*, pages 186–3, 2013.

[GCR16] Ilias Giechaskiel, Cas Cremers, and Kasper Bonne Rasmussen. On bitcoin security in the presence of broken crypto primitives. *IACR Cryptology ePrint Archive*, 2016:167, 2016.

[Gup15] Vinay Gupta. The ethereum launch process - ethereum blog. https://blog.ethereum.org/2015/03/03/ethereum-launch-process/, 2015. (Accessed on 06/02/2017).

[Homa] Home · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki. (Accessed on 05/22/2017).

[Homb] Homepage – slushpool.com. https://slushpool.com/home/. (Accessed on 05/11/2017).

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[HS05] Paul Hoffman and Bruce Schneier. Attacks on cryptographic hashes in internet protocols. Technical report, 2005.

[Lam81] Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, November 1981.

[Ler] Sergio Demian Lerner. Uncle mining, an ethereum consensus protocol flaw | bitslog. https://bitslog.wordpress.com/2016/04/28/uncle-mining-an-ethereum-consensus-protocol-flaw/. (Accessed on 04/11/2017).

[Lig] Light client protocol · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki/Light-client-protocol. (Accessed on 06/12/2017).

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[Mer82] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[Mic16] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[Min] Mining pool centralization at crisis levels — bitcoin magazine. https://bitcoinmagazine.com/articles/mining-pool-centralization-crisis-levels-1389302892/. (Accessed on 06/12/2017).

[Mor68] Donald R Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008. (Accessed on 06/12/2017).

[NBF$^+$16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press, 2016.

[NIS] Nist computer security publications - fips (federal infromation processing standards). http://csrc.nist.gov/publications/PubsFIPS.html. (Accessed on 02/24/2017).

[OnS] On slow and fast block times - ethereum blog. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/. (Accessed on 06/07/2017).

[OR94] Martin J Osborne and Ariel Rubinstein. *A course in game theory.* MIT press, 1994.

[P2P] P2pool - bitcoin wiki. https://en.bitcoin.it/wiki/P2Pool. (Accessed on 06/07/2017).

[Pat] Patricia tree · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki/Patricia-Tree. (Accessed on 03/23/2017).

[Pro] Promoting a free, decentralized and open future. https://blog.ethereum.org/. (Accessed on 05/22/2017).

[PUB95] FIPS PUB. Secure hash standard. *Public Law*, 100:235, 1995.

[PUB14] FIPS PUB. Draft fips pub 202: SHA-3 standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standards Publication*, 2014.

[Qu99] Ming Hua Qu. Sec 2: Recommended elliptic curve domain parameters. https://pdfs.semanticscholar.org/2840/bfaf7cf8f97deeaddcd4274e82863ec522ee.pdf, 1999. (Accessed on 06/12/2017).

[Riv92] Ronald Rivest. The MD5 message-digest algorithm. 1992.

[RS97] Ronald L. Rivest and Adi Shamir. *PayWord and MicroMint: Two simple micropayment schemes*, pages 69–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[Seq] Sequence number (transactions) - bitcoin glossary. https://bitcoin.org/en/glossary/sequence-number. (Accessed on 05/01/2017).

[SKR$^+$11] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto. *Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols*, pages 284–301. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Sol] Solidity — solidity 0.4.12 documentation. http://solidity.readthedocs.io/en/latest/. (Accessed on 05/15/2017).

[Sti05] Douglas R Stinson. *Cryptography: theory and practice.* CRC press, 2005.

[SZ13] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. http://eprint.iacr.org/2013/881.

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[Ult] Ultimate blockchain compression w/ trust-free lite nodes. https://bitcointalk.org/index.php?topic=88208.0. (Accessed on 04/17/2017).

[Whaa] What is ethereum? — ethereum homestead 0.1 documentation. http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html. (Accessed on 06/12/2017).

[Whab] What is the size of the ethereum block-chain?:ethereum. https://www.reddit.com/r/ethereum/comments/61zh94/what_is_the_size_of_the_ethereum_blockchain/. (Accessed on 05/21/2017).

[Wik15] Ethereum Wiki. Serpent 1.0 (old). http://mc2-umd.github.io/ethereumlab/docs/serpent_tutorial.pdf, 2015. (Accessed on 06/12/2017).

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.