DEPARTMENT OF INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Master’s Thesis in Information Systems

Technical Analysis of Established Blockchain Systems

Florian Haffke

DEPARTMENT OF INFORMATICS
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Technical Analysis of Established Blockchain Systems

Technische Analyse etablierter Blockchain-Systeme

Author:	Florian Haffke
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Patrick Holl
Submission Date:	15.11.2017

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.11.2017

Signature

Abstract

Since the invention of Bitcoin as a digital currency in 2008, the underlying blockchain technology has become a much-debated subject. The blockchain design promises to bear tamper-resistant data due to its continuously growing list of records, that is cryptographically connected. Blockchain database systems assert to be secure as a distributed ledger for financial transactions with many other suitable applications expected to arise.

This thesis covers an analysis of three blockchain protocols, Bitcoin, Ethereum and Ripple. We decompose their structure and investigate their elements individually and comparatively to give a better understanding about their functionality and issues. This includes predominantly the block setup, the consensus algorithms, the transaction systems and the networks. Furthermore, we compile our gathered intelligence into abstract schemes of the technical ecosystem.

Keywords

Asymmetrical Cryptography, Bitcoin, Blockchain System, Blockchain Design Space, Block Header, Digital Signature Scheme, Distributed Consensus, Distributed Ledger Technology, Ethereum, Hash Pointer, Hash Tree, Mining, Peer-to-Peer Network, Proof of Work, Ripple

Contents

List of Figures

List of Abbreviations

ASIC	Application-specific Integrated Circuit
BIP	Bitcoin Improvement Proposal
DAC	Distributed Autonomous Company
DAO	Distributed Autonomous Organization
ECDSA	Elliptic Curve Digital Signature Algorithm
EIP	Ethereum Improvement Proposal
EVM	Ethereum Virtual Machine
ICO	Initial Coin Offering
ILP	Interledger Protocol
IOU	I Owe You
ISP	Internet Service Provider
PoA	Proof of Activity
PoC	Proof of Correctness
PoS	Proof of Stake
PoSC	Proof of Storage Capacity
PoW	Proof of Work
RPCA	Ripple Protocol Consensus Algorithm
RLP	Recursive Length Prefix
RSA	Rivest–Shamir–Adleman
RQ	Research Question
SHA	Secure Hash Algorithm
SPV	Simplified Payment Verification
Tx	Transaction
UNL	Unique Node List
UTXO	Unspent Transaction Output
XRP	Ripple Protocol Token

1 Introduction

1.1 Motivation

On one side, we currently have a market hype in cryptocurrencies, the first application of blockchain systems. (coinmarketcap.com, 2017) show an increase in amount to more than 1,200 currencies to date and an investors' valuation explosion from practically nothing at the first release of Bitcoin in 2009 to roughly more than one hundred billion USD as of summer 2017. Enlargements of the networks with rising computing power support the evaluation. On the other hand, comments from adversaries like Jamie Dimon, the CEO of JPMorgan Chase & Co, diminish faith and trust in a long-term success against fiat money calling Bitcoin "a fraud" and comparing it to the speculative fallacy of tulip mania. Dimon further alleged "You can't have a business where people can invent a currency out of thin air [..]". (Reuters, 2017)

What both sides agree on is a potential of the blockchain technology to cut middlemen costs, support transparency and generally digitalize a variety of processes currently handled in the real world. (Google Trends, 2017) indicates the topic "blockchain" has become exponentially more popular over the last three years. Upon this common ground, we want to examine the technology and author profound material.

1.2 Research Questions and Purpose

The novelty of blockchain as a database technology rises the demand for easy understandable and explanatory information to bridge the purely technical source code and developers' view with the outside ecosystem and users. We aim to answer five research questions (*RQs*) to give insights in the status quo of current blockchain systems by evaluating their technical substance.

RQ1 Which are established Blockchain Systems?

Before we can investigate blockchains, we need to define basic related terms and identify the established systems after appropriate criteria to get encompassing yet profound insights into the technology.

RQ2 What is the respective Setup of established Blockchain Systems?

To build up an enlightening overview of the setup of blockchain systems, we scrutinize these systems guided by four sub-questions: What is the content and setup of a block, its header and its data? How do the consensus processes work? What are the structure and communication forms of the underlying network? What issues evolve for the system to pursue its goal of existence?

RQ3 How do established Blockchain Systems differ?

Additionally, we explain how the systems differ from each other regarding their setups and why. This includes, but is not limited to be subject to distinctions in blocks, variations in the network design and consensus algorithms. RQ2 and RQ3 are the most comprehensive parts of this thesis.

RQ4 What are crucial Components and Characteristics of all established Blockchain Systems?

Blockchain systems may have a very distinguishable format, others are very alike. The question arises what the common ground of all these systems is. We aim to identify crucial elements and characteristics used in established blockchain systems. Further we construct high-level depictions of their resembling components in their architectural relationships as well as roles involved in the ecosystem. In addition, we research actual achievements of the blockchain technology and compare their implications to traditional databases and distributed systems.

RQ5 How can a Design Space of Blockchain Systems be defined?

The design space, especially the choice of variables and parameters dictates the quality and success of a system. We propose how to define a potential design space for blockchain systems in a morphological analysis. This constitutes attribute definitions, possible parameter values and their multidimensional combination and interaction. We then categorize and classify the blockchain systems. Furthermore, we investigate common technical issues in the blockchain sphere such as scalability, system access, the role of a native token and security risks.

1.3 Research Approach

Technical analysis is a broad term. We demonstrate our specified strategy in the figure below to gain a better understanding of the formal concept. The principle procedure follows four major parts. First, we identify the established systems according quantitative criteria and consolidate first-hand material, such as the source codes, the developer's documentations and the whitepapers. Second, we decompose the complex matter by identifying contained elements, their relationships plus behavior and existing processes within the systems. Third, we scrutinize the found entities individually and jointly to discern potentials and issue patterns. And last, we synthesize the critical components to derive high-level models and knowledge of the design space. Complementing the system analyses we regard ancillary literature reviews, although the focus remains on the technical specifications.

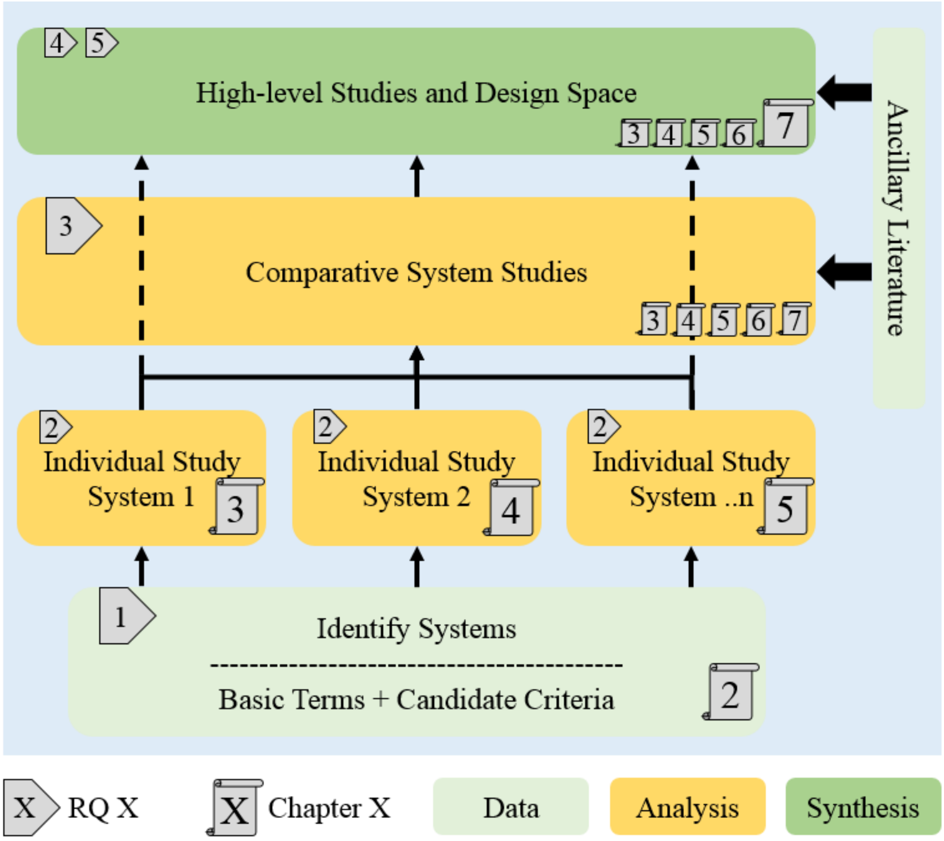

4
5
High-level Studies and Design Space
3 4 5 6 7
3
Comparative System Studies
3 4 5 6 7
Ancillary Literature
2
Individual Study System 1
3
2
Individual Study System 2
4
2
Individual Study System ..n
5
1
Identify Systems
Basic Terms + Candidate Criteria
2
X RQ X
X Chapter X
Data
Analysis
Synthesis

Figure 1 Research Strategy

1.4 Outline

Chapter two comprises essential concepts and the selection process for established blockchain systems, thus RQ1.

We compound RQ2 and RQ3, the comprehensive studies about the specific blockchain systems, together in the proximate three chapters.

The depictions and high-level views from RQ4 will be partly in the chapters for each respective blockchain. Moreover, we aggregate their intersections in chapter seven. The same pattern applies for RQ5 about the design space.

2 Established Blockchain Systems

2.1 Introduction

To examine established blockchain systems, we first define the technological fundamentals for what a blockchain system is and then arrange criteria to discover the most established ones.

2.2 Determine: *Blockchain System*

2.2.1 Basic Terms

In literature, different authors describe the meaning of the term *blockchain* in various ways. Common ground is, that the term derives from its very first application, Bitcoin, and its used data structure. Satoshi Nakamoto is the pseudonym the founder and developer of Bitcoin used to publish their original paper and codebase. The real-world identity of Nakamoto and whether the name refers to a single person, a group of people or some other institutional entity remains unknown to date. To be able to academically analyze established blockchain systems, we first need to define the term blockchain and its related terminology. Per original paper (Nakamoto, 2008) and codebase of Bitcoin, we propose to generically define as follows:

Blockchain

An ongoing chain of blocks, i.e. records, forming a sequential linked list with hash pointers and separately containing data. A blockchain is typically redundantly distributed across a peer-2-peer network, that verifies the integrity of existing blocks and adds new blocks to serve as a distributed database. Verification obeys a set of protocol rules, the codebase. The goal of a blockchain is to be secure by design with a tamperproof validation of data at time.

Blockchain System

The entire system backing a blockchain. This comprises the data and its structure, the network infrastructure and the codebase. Specifically excluded is the revolving ecosystem around a blockchain such as applications or external participants.

Blockchain Token

An optional virtual-only token used in the data of a blockchain as a means of ownership, identifier or any other form of right or obligation.

The close dependence on the termini of Bitcoin helps to stick to the very ground principles, but on a more abstract perspective. However, we acknowledge the existence of more than one blockchain, blockchain system and blockchain token. We will therefore always use the undefined article or specify it using distinguishable names, such as the Bitcoin blockchain, the Bitcoin system and the Bitcoin token. We broadly define the just used termini in more detail in the following sub-chapters.

2.2.2 A Block

A data element acting as a record comprising of a header for meta data and a body for separated arbitrary data. The header contains at least some form of hashed reference to (1) the arbitrary data and (2) a hash pointer to one different existing block header. A special case of a block not referencing one existing block is named a genesis block, which therefore marks the beginning of the data structure.

Hash Pointer

A hash pointer is a pointer to where some data is stored via a cryptographic hash of that data. It enables to look up the data and through the hash to verify that it is untampered. (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016, p. 32)

2.2.3 *The* Chain

A blockchain as we defined is a chain. With *the* chain or main chain, we define the longest chain of blocks at a specific snapshot in time, usually the current time. *Longest* refers to the number of chained blocks compared to other chains having an identical starting block zero. The chain builds the data structure of a logical clock. It orders the creation event of its blocks in distinct and network-wide logical time.

Logical Clock

A logical clock is a mechanism for assigning chronological relationships to events in a distributed system.

Mining

The process of continuously creating new blocks and adding them to the chain, meanwhile emerging new tokens following verifiable rules. Participants verify the legitimacy of the blocks according a specific set of rules. We call blocks competing if they are not linearly chained.

2.2.4 The Set of Rules

The set of protocol rules a blockchain system obeys is the identical codebase running on a computer or a network of computers. Usually participants use a peer-2-peer network structure. We therefore determine the identity of a blockchain from the codebase alone. If two or more blockchains with the identical code exist at the same time, they are either competing chains or run on separate computers or networks. If one modifies any part of the source code and participants use altered and unaltered codebase both within the same system, we say their rules are compliant or compatible. If they are not, we name that as *protocol-based* forking. It leads to two unidentical blockchains. An exception is a modification of the genesis block in the source code, what we call *cloning*.

2.2.5 The Network

A blockchain can be executed on a single computer or on a network of computers. To achieve its goal of data immutability, the protocol typically runs in a decentralized or distributed manner. An unstructured peer-to-peer network has become best practice since Bitcoin.

Peer-to-Peer Network

A peer-to-peer network is a distributed application architecture interconnecting nodes, the peers, in an equipotent way to achieve a common goal. The peers share resources amongst each other with or without using a central authority for administration. In a blockchain system the network prominently shares the resource data, namely the blockchain.

2.2.6 Functionality and Application

By achieving a tamperproof validation of data at time, a blockchain can serve as a data ledger. The distributed nature of the peer-to-peer network sharing and replicating the data misses a single point of failure, which makes it more robust against concentrated attacks. Therefore, the first ever application of a blockchain, viz. Bitcoin, functions as a shared public ledger of transactions. To introduce the concept of a transaction a blockchain system can use a virtual currency unit, which parties create via mining and transfer between each other.

Data Ledger and Digital Currency

A data ledger is a file used to total and record economic transactions conducted in a monetary unit of account. A blockchain is a purely digital and distributed data ledger. Its unit of account is the self-emitted virtual and digital-only currency.

2.2.7 Access and Participation

The last concept, considers how to access and manipulate the data from outside the system - both by humans and machines. Bitcoin utilizes a node-independent form of identity to participate in the data ledger of transactions. It takes advantage of a digital signature scheme, where every human or machine party can deploy his activities on the data with multiple anonymous identities.

Digital Signature Scheme

A digital signature is a mathematical scheme to authenticate digital data, e.g. a message. A valid signature on data adequately insures the recipient that the data was undeniably created by a known entity (*non-repudiation*) and was not altered on transit (*integrity*). A digital signature scheme builds on an algorithm to draw a private key with discrete uniform distribution from a vast enough set of possible private keys to not get duplicates. This allows for multiple identities per participant. Then it determines the corresponding public key using an asymmetrical cryptographic function. One part is an algorithm to create a digital signature from a private key and a specific message. The other part is an algorithm to verify the private key - given the signature, the same message and the respective public key. (Driscoll, 2013) depicts the process:

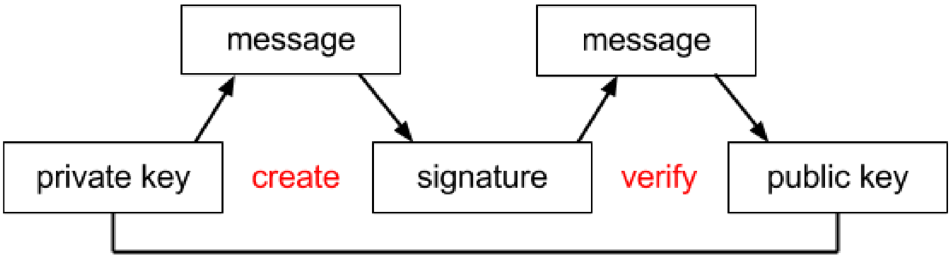


Figure 2 *Digital Signature Scheme*

2.3 Determine: *Established*

2.3.1 Criteria

To analyze the setup of a blockchain system and compare it to other approaches, we first need to know which are established working blockchain systems. We determine relevant blockchain systems on a qualitative evaluation supported by eight quantitative criteria. We choose these criteria with the intent to have independent, not correlated metrics. Due to the novelty, not all existing blockchain systems have available data or it may fluctuate drastically in short time. That is why we cannot meaningfully test for statistical metrics such as deviation or a correlation value, but rather reason the importance and independency as stated below:

Longevity

This criterion shows the age of a blockchain system in years elapsed since the initial release date of its source code. It is very essential since a longer active existence can have a positive impact on undergone research, defeated attacks and general trust built in the system.

Supporting Community and Development Support

Supporting community takes the absolute number of subscribers to a blockchain system's related content at the widely used social news, content and discussion web portal Reddit. *Development support* meters the activity in public source code repositories, such as subscriptions, commits and push and pull requests in total number. Both figures seem to be similar, but consider two distinct communities, the IT-engineering site and the user site, that have little intersection. For that reason, we consider them two independent measurements and of high importance on our list.

Public Awareness and Interest

The overall public awareness and interest in a blockchain system plays a major role in the adoption of a system. Our instrument of measurement is the Alexa rank of the main page of the respective blockchain project. Alexa ranking is an algorithm to calculate the internet traffic and page views recorded in a single ordinal number. However, just gauging the number as a full criterion might overlap with the information from the figure in *supporting community*. Therefore, we weigh the ranking with lower effect.

Investor Evaluation

We derive the investors’ evaluation from the market capitalization of a blockchain’s native token, that investors trade on public exchanges. This does not only reflect the current establishment of a blockchain system, but also implies speculative value about the future of a system. Whereas capital inputs in infrastructure and the ecosystem made by companies or private parties are explicitly not measured directly. We consider the token value a better indicator since it is a publicly available market and the latter capital inputs would therefore be redundant.

Application Ecosystem

The integration and interoperability of a blockchain grows with evolving an application ecosystem around it. Since that is hard to gauge, we set up an ordered possible spectrum from one (“Very Few”) to five (“Very Many”) and argue the classification.

Network Activity

Transferring a token on a blockchain bears cost, such as degrading the arriving token amount by fees or by computational effort. This reasons to take the number of transactions per day as a suitable measurement for the overall network activity of a blockchain system. Checking the number of active nodes in the network would not fit our needs since it highly depends on the nature of the network architecture, some are open and facile to join others are not.

Technical Uniqueness of Protocol

The final attribute shows the technical uniqueness of the underlying protocol code. We regard it decisive impact on our selection process. Originality yields in a first mover advantage and gains reputation. The scale ranges from one (“Very Low”) to five (“Very High”).

2.3.2 Candidates

We apply these metrics on a preselected set of six blockchain systems. The preselection is argued as follows:

Bitcoin

Bitcoin is the very first and still active blockchain system designed as a payment protocol. It has the biggest supporting community, the highest token market capitalization and the most matured ecosystem.

Litecoin

Litecoin is an early stage clone of Bitcoin that gathered up a large following community because of its technical improvements mainly relating to scaling and efficiency. The ecosystem landscape also exhibits many integrations and wide acceptance.

Ripple

The Ripple blockchain system arose from an earlier existing payment processing company and is unique. Furthermore, network activity in form of daily transactions amount is one of the highest in the blockchain sphere.

Ethereum

Ethereum is the first general purpose blockchain to allow Turing-complete smart contract implementations and machine-to-machine communication. Yet it is young and in early protocol development with much on the roadmap of its foundation. It relies on vast development support, high network activity and public awareness.

Hyperledger Project

The Hyperledger Project is a consortium, which consists of several independent blockchain systems and concepts led by the Linux Foundation. Hyperledger Fabric is the first in-use permissioned and therefore token-less blockchain of the technology company IBM. The protocol is highly unique.

Zcash

Zcash is a recent, distinctive clone based on Bitcoin focusing on privacy improvements. It is the first blockchain to successfully put zero-knowledge proofs into practical use for transaction anonymity. It rapidly gained public awareness and a comprehensive development and research community.

Our table shows an aggregated overview of the average of two snapshots taken in a quarterly interval at 2017-07-08 and 2017-10-08.[1]

[1] Cf. Appendix B for both individual snapshot tables.

Data Snapshot – Averaged

Criterium	Metric *[Unit]*	Bitcoin	Litecoin	Ethereum	Hyperledger Project	Ripple	Zcash
Supporting Community	***Reddit Subscribers [#]***	300,900	50,740	107,310	--- Linux Foundation, IBM	21,901	4,747
Development Support	***Activity in Public Source Code Repos [#]***	15,673	1,658	6,745	--- Linux Foundation, IBM	1,514	2,701
Longevity	***Age since Initial Release Date [Years]***	8.5 (01-2009)	6 (10-2011)	2 (07-2015)	1.5 (12-2015)	4.5 (10-2012)	1 (10-2016)
Network Activity	***Transactions [# per Day]***	215,524	17,864	265,131	---	811,778	4,576
Investor Evaluation	***Market cap of native currency [Bn$]***	58.5	2.65	27.25	---	9.95	0.5
Public Awareness and Interest	***Alexa Rank [-]***	6,025	60,876	8,023	111,634	13,967	21,351
Technical Uniqueness of Protocol	***Ordered Attribute Scale [1..5]***	5 Very High (First Mover)	2 Low (Bitcoin Clone)	4 High (Parts of Bitcoin)	5 Very High (First Mover)	5 Very High (First Mover)	2 Low (Bitcoin Clone)
Application Ecosystem	***Ordered Attribute Scale [1..5]***	5 Very Many	3 Middle	4 Many	2 Few	3 Middle	2 Few

Taking all data into account, we can see a dominance of Bitcoin in most criteria with a lead. However, looking at the derivative with respect to lifetime, Ethereum holds the steepest slope. The gap between all other blockchain systems is closer with altering lead. Litecoin and Zcash are prosperous systems and worth analyzing phenomena. But since they are both protocol-wise descendants of Bitcoin and therefore technically related, not suitable for our IT-analysis. Hyperledger Fabric looks innovative and promising on protocol level, but because of its young age and lack of awareness it is too immature to be labelled as established. An extensive public awareness and the highest network activity alone, would not give us reason to contemplate Ripple. However, Ripple offers a completely distinct codebase compared to the other systems. To get the most reasonable and insightful analysis, we therefore decide to study Bitcoin, Ethereum and Ripple in themselves and highlight how and why Ethereum and Ripple differ from Bitcoin.

2.3.3 Further Blockchain Concepts

Using only Bitcoin, Ethereum and Ripple, we would not paint a complete picture of the existing blockchain landscape. Many derivations from the Bitcoin protocol have emerged with slight or mere fundamental changes. Therefore, in addition to these three, we also need to scrutinize the design decisions of the described essential concepts. We compare them to Bitcoin and relate singularities of their typical blockchains. We reason to include the concepts and group them as follows:

Cryptography

Working and interdigitated cryptographic choices are the fundament of communication without trusting another human participant. Therefore, we examine both major cryptographic parts, namely different hashing algorithms and different digital signature schemes.

Distributed Consensus

Data integrity, that is especially accuracy and consistency of data, is a critical aspect for design choices of a database. The blockchain as a distributed database crucially needs a functioning consensus algorithm. That is why we investigate the implications of different variations.

Interfaces and Access

To fit in the existing world of a more centralized internet led by many companies and established communication protocols for their users, we must find a way to assess different interface choices for a blockchain. Hence, we consider oracles a way to communicate data and algorithms for privacy and intransparency a way to communicate identity and access possibilities.

Network Structure

Defining the rules how nodes in a network communicate with each other and if they are equipotent peers has considerable impact on the structure of the network – and consequently on the allocation of roles and power. On that account, we want to compare two approaches, namely a multi-tiered peer-to-peer network and private centralized network.

3 Bitcoin Protocol

3.1 Introduction

This chapter covers the setup of the current Bitcoin blockchain system. If not specified otherwise the technical data is derived only from the Bitcoin Core reference client[2] and the official documentations led by the Bitcoin Project[3]. If there is conflicting information, the source code is seen to be the Bitcoin identity and acts as single source of truth. Structure, presentation and evaluation of the information is the work of the author.

3.2 Background and Application Purpose

There have been several attempts to discover a method for creating electronic money, that is, purely digital monetary units which can be stored as data and transferred via the internet without being double spent. Two mentionable - but not implemented - precursors to Bitcoin are *b-money* by (Dai, 1998) and *bit gold* by (Szabo, 2008). The original white paper of Bitcoin was published by an unknown entity under the pseudonym Satoshi Nakamoto in October 2008 and the first working source code released in January 2009 by Nakamoto.

As stated in the white paper (Nakamoto, 2008), Bitcoin is a purely peer-to-peer electronic cash system, that allows to send online payments directly from one party to another without going through a financial institution and without the possibility of double spending. It aims to provide non-reversible transactions, eliminate trust in any third party and cut the cost of mediation to support commerce on the internet.

3.3 A Block

As for every part of a blockchain system, the rules written in the Bitcoin source code define the structure of a Bitcoin block. It consists of five fields as shown in the byte depiction below. The *file signature*, also called the magic number, is the constant hex value of D9 B4 BE F9 and used as an identifier for a Bitcoin block in the network. Otherwise the receiver of the data would not know to assign it as Bitcoin-related. The second field indicates the *size* of the coming block in bytes and the fourth the *number of included transactions*. These three fields make up for the meta data with network communication purpose. The two major parts however, are the 80-byte block *header* and the actual *transaction data* using up the rest of the blocksize, which is the artificially set upper limit for a single block: one megabyte.

[2] Cf. (Bitcoin Core, 2017)

[3] Cf. (Bitcoin - Developer Documentation, 2017) and (Bitcoin Technical Wiki, 2017)

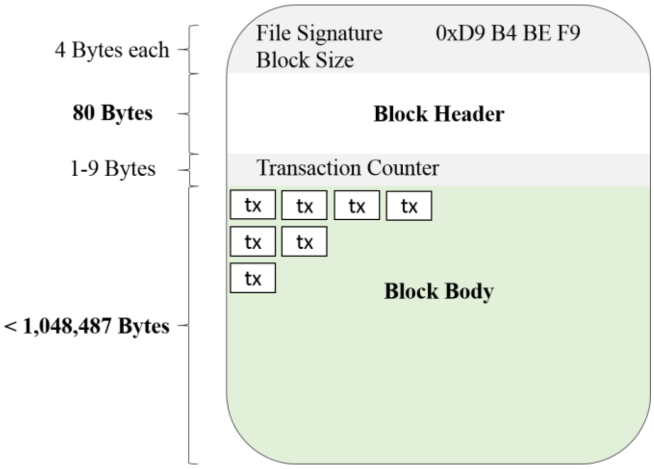


Figure 3 *Bitcoin Block simplified with Byte-map*

3.3.1 Body - Transaction Data

We cover the detailed setup of a single transaction data object in the chapter of the transaction system. For now, we only want to show how the data structure for all transactions in one block is stored. According to (blockchain.info, 2017), a block roughly contained 1400-2000 transactions on average in 2017. The transactions of one block can be written in arbitrary sequential order except that the first one must always be a special *coinbase* transaction. The final piece included in the transaction data part is the logical data structure applied for these transactions, a *Merkle tree*. This binary tree with hash pointers is constructed by (1) hashing every transaction with the SHA-256 hash function twice, (2) then concatenating every two consecutive hashes and hashing them again. This second step repeats until there is only one hash left, which is called the root hash. All hashes of the tree are stored as part of the transaction data in the block. We depict an exemplary tree below. The tree structure is very useful for large data amounts as it can prove if it contains some data or not in logarithmic time O (log n) just by checking the path from a data leaf to the root hash. Besides proving existence, it is also cheap to add or delete a transaction since the vertical hash path to the root is the only work to be redone apart from hashing that single transaction itself. This is beneficial for Bitcoin since transaction allocation in a block and reading old transactions are frequent operations and thus can be validated much faster. Using the entire block as hash input every time would perform extremely poor.

3.3.2 Header

The header of a Bitcoin block is of crucial meaning for the whole model of the blockchain. It consists of six elements. Besides the *block version number*, a standard *timestamp* and the *Bits field*, which indicates the target value for the mining process, there are three very important elements. First, the calculated *Merkle root hash* of all included transactions. Second, the double SHA-256 *hash of the header of the previous older block*. And third, a *nonce*, a random onetime number[4]. The inclusion of these data items leads to them being input variables when hashing

[4] We explain the importance for the nonce in the chapter about mining and Proof of Work.

the entire header. The header hash is then the input string for the subsequent block header. This mechanism is the method for connecting a block with one precursor and one successor. One property of the hash function is, that the output value changes unpredictably if the input string differs only a single bit. This makes it practically impossible to alter the referenced raw data without losing the reference connection. The raw data mainly are all included transactions and the header of the previous block. An overview of this linking scheme is illustrated below where directed arrows indicate a hash function.

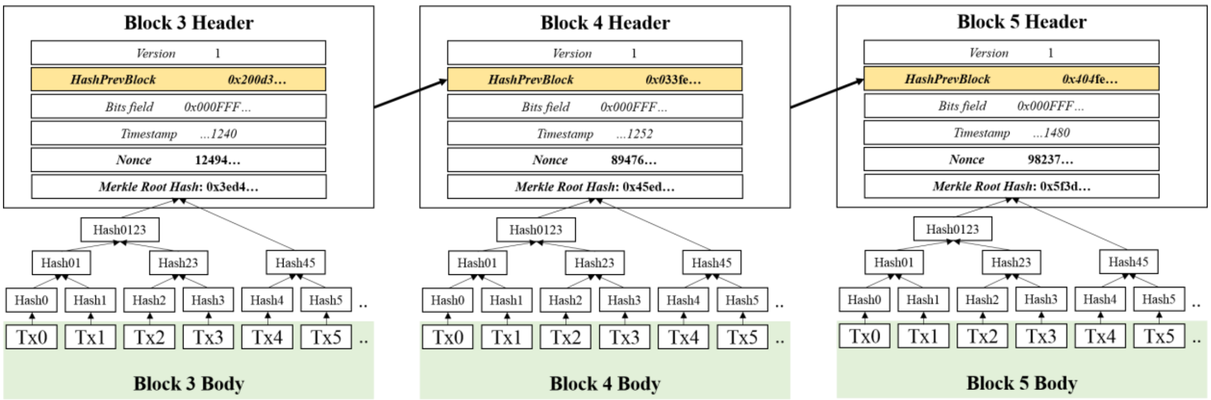


Figure 4 *Block Connection Mechanism*

3.4 The Chain

We have shown the mechanism for connecting one block with two others, a previous and a subsequent one. This accomplishes the chain structure of a blockchain. But where does this chain begin and where does it end? The very first block, the *genesis block*, is hardcoded into the source code. It has a hash pointer with only zeros and therefore is considered to have no previous block. Additionally, it contains some arbitrary data within its coinbase transaction to be identifiable. In the case of Bitcoin, it is data from a newspaper article to prove the genesis block is not older than the release date of that newspaper. Using another genesis block means exclusion and results in having another chain. The end of the chain is the most recent block, whose hash is not yet used as an input for another block. We can easily imagine a scenario where the same block is used as an input for two different blocks, thus being connected to more than one subsequent block. This is totally possible. However, the source code defines the rule that there is only one valid chain, which is the longest path from the genesis block to an ending block. It contains the highest number of blocks or more precisely the biggest hashing effort.[5] As of October 2017, running the Bitcoin Core client shows this main chain has a height of about 487,000 blocks. All other blocks not in the main chain are part of mining forks and labelled stale or *orphaned*. The judging is gradual if nodes happen to concurrently create competing blocks.

[5] Cf. the next chapter about mining.

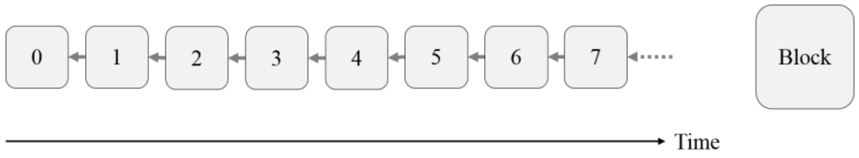


Figure 5 *Blocks in a straight Chain*

3.5 Mining

The Bitcoin mining algorithm reaches consensus to add new valid blocks to the chain. We divide it into two sub-sections:

(1) Creating new blocks

- Assembling transactions
- Hashing with Proof of Work (*PoW*)

(2) Selecting valid blocks

- Checking PoW
- Deciding for longest chain

3.5.1 Distributed Consensus

So far, we have seen how blocks connects with each other to a chain and that there is only one valid main chain. But since the blockchain is operated on a peer-to-peer network, how do we achieve a distributed consensus for the overall system to obtain reliable and consistent data? To be a working payment system, Bitcoin needs to have high byzantine fault tolerance, that is if more than half of the network nodes behave honestly the system's consensus is resilient to any type of error that may occur. It therefore needs to have four properties we derived from (Kshemkalyani & Singhal, 2008) and translate to Bitcoin.

(1) ***Termination***: *Every correct process needs to decide some value*.

The Bitcoin processes for persistent data storage are writing a block and reading a block.

(2) ***Validity***: *If all correct processes propose value v, then all correct processes decide value v.*

For Bitcoin, this means if a block is valid, it needs to be accepted as valid and added as part of the chain.

(3) ***Integrity****: If a correct process decides value v, then v must have been proposed by a correct process.*

Relating to Bitcoin, all invalid blocks must be denied being accepted in the chain.

(4) ***Agreement***: *Every correct process must agree on the same value.*

> Translated to Bitcoin, this says there can never be more than one valid block referencing to the same previous block or in other words the chain must be linear without branches.

The main part of the mining process, called Proof of Work, takes over these tasks as a distributed consensus algorithm. We label every network node participating in it, a miner or mining node.

3.5.2 Creating new Blocks – *Proof of Work*

Our linking scheme for blocks as illustrated in the header section, would be efficient to produce as it only uses the input values hashed with the hashing function. This is one-time computation and mining nodes would quickly be able to calculate a vast number of new blocks in decreasingly shorter time, which clogs the network and branches the chain. That is why the Bitcoin mining algorithm utilizes the hashcash Proof of Work (*PoW*) function specified by (Back, 2002). It states the fixed-length output hash of a block header needs to be under a specific adjustable threshold value, the target value, to be accepted as a new valid block by others. Most hashing functions including Bitcoin's SHA-256 are puzzle-friendly. Puzzle-friendliness means, that the goal to find a hash from a target set of outputs is infeasible to be conducted with a solving strategy significantly better than simply trying random inputs. Verifying the correctness of a calculated target hash is very easy. This is crucial because it prevents to relay the costly effort to other nodes for checking. Updating any data in the header, such as the Merkle root hash for the transactions, the timestamp or the nonce, all alter the output hash of the header in an unpredictable manner. A miner therefore repeatedly updates and assembles all input values and hashes the block header to find a correct target hash. This brute-forcing necessity implies statistically ensured computational work for every valid hash smaller than the target value. The target is directly stored in the *Bits field* of the block header and adjusts bitwise every 2016 blocks - roughly two weeks. The goal is to average the mean time for anybody in the network to calculate a new valid hash - and therefore a block - at about ten minutes. Every node can do the calculation on how to adjust the target independently if they measure the time elapsed for 2016 blocks from their point of view. The ten-minute rhythm is somewhat arbitrary as a balance for synchronizing data within the network and having steady and fast confirmations of blocks and transactions.[6] The final question is why would any miner care to assemble a valid block if it costs much effort to do so? The answer is simple and uses game theoretic incentives. The miner gets to have a financial reward in the form of Bitcoin tokens. A fixed portion is a set block reward creating new Bitcoin tokens, that halves every 210,000 blocks – roughly 4 years. The variable part are fees from the transactions he integrates in the block. He receives the reward with the special coinbase transaction, that must be the very first transaction he includes. The financial income on one side and the computational cost on the other, construct a market for mining following the economic terms of supply and demand. If there are more transactions than fitting into the one megabyte space of a single block, there is competition between transaction and the miners can select the ones with the highest fees. A miner could deny adding any transaction other than the coinbase transaction, but this would give him income disadvantage and only reduces the cost of hashing marginally since he just saves one-time work

[6] 455 weeks elapsed since the release date for 48,700 blocks. This results in ~9.4 min/block. The discrepancy is because the hash power increases within the atomic adjustment interval of two weeks.

for computing the Merkle root again, but does not save any of the significant brute-forcing time. The miners are therefore also in economic competition.

3.5.3 Selecting valid Blocks

All nodes check a created block they receive independently for correctness. The fulfilling criteria especially contain that all transactions must be valid and that the hashes need to be right. This is one-time work and not expensive. The source code rules further state that only the longest chain of blocks should be considered valid. This is equivalent to following the most computational hashing power put into the blocks' creation. Cheating this rule means that a node tries to broadcast invalid blocks for whatever reason. First problem case would be, if the block's data or hashes in themselves are not correct. Other nodes will simply reject that. The second situation is where a node creates a correct block with a hash pointer to a previous block that already has one or more valid successors. This leads to a competition between the two branches. Any miner creating a new block for one of the branches further validates this branch and claims the other one as invalid. We should keep in mind, that every node has a different view of the chain, and therefore does not necessarily know if there is a conflicting block or branch at the time he creates a new block. But as soon as he finds out, he would switch to the longer branch even if it may invalidate his own block. A tie would mean miners on both branches are in a race with each other to create the longest chain. Since this race is determined by hash power, one side should need to have more than 50% of computational power to be likelier to win the other nodes back on their branch. Otherwise they are doomed to fail and their blocks and coinbase transactions become invalid to others. A tie happens occasionally and leads to an overall situation of the chained blocks as depicted in the figure. The view we depict is an external perspective for the system. A single node would only have one of the blockchains from time four to seven making them concurrent regarding global logical time. The stale outpaced blocks not in the main chain will get lost over time since there is no incentive to keep them. This economic peer pressure finally guarantees to generate a linear chain of sequential blocks with consensus about a consistent state of the data.

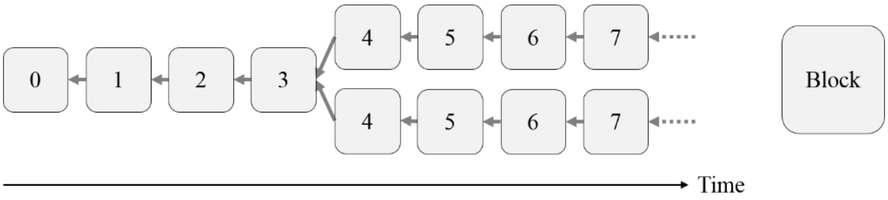


Figure 6 Mining Race after Block #3 with a Tie Situation

3.5.4 The Collaboration Dilemma

If two parties permanently stuck to their branch of the chain even if they knew there is a longer one, we call that *mining-based forking.* This would gradually split one chain into two independent ones. The scenario appears unlikely if there are many nodes with approximately equal computational power. However, if there are only a few nodes with significant computational power, these leading miners may have enough supporting power by the community to continue an individual branch as a new chain. *Mining pools,* a collaboration of hashing power to increase chances of finding a valid block, essentially act as a single node with high significance. They are a thread to the distributed nature of the peer-to-peer network. The incentive to collaborate is simply because they share the financial reward between participants. The share regards different measurements of participation. Usually mining pools compensate participants for a calculated block within a slightly bigger target range than the actual block target, even though that block is not added to the main chain. Participants in a mining pool therefore have a more steady and plannable income stream with less risk. A tendency for centralization of hashing power and interest are the consequences.

3.5.5 Implications

The mining algorithm implements a timestamping server on a peer-to-peer basis. Once a valid block is in the main chain, it cannot be changed without redoing the proof of work for this block and all succeeding blocks. The decision making is based on the majority of computational power. We sum up if the Bitcoin mining algorithm satisfies all mentioned properties for reaching distributed consensus:

(1) ***Termination**: The Bitcoin processes for persistent data storage are writing a block and reading a block.*

The block creation process decides for a hash value within finite time without looping or aborting. Since there is a target range and only brute forcing strategy, we get a probabilistic approach for the entire network that adjusts to a target value of ten minutes. Reading is trivial and done locally on each node on demand.

(2) ***Validity**: If a block is valid, it needs to be accepted as valid and added as part of the chain*.

This is stated in the rules. Transaction validations and hash verifications are one-time work done at each node. Applying the rule for the longest chain is ensured via economic incentive.

*(3) **Integrity**: All invalid blocks must be denied being accepted in the chain.*

Nodes deny malformed blocks or blocks not in the longest chain because of economic peer pressure that their own following block would not be accepted either. Otherwise they are considered a fork or split and not part of the same blockchain system anymore.

*(4) **Agreement**: Translated to Bitcoin, this says there can never be more than one valid block referencing to the same previous block or in other words the chain must be linear without branches.*

Again, this is done via the economic rewarding. Nodes try mining the chain with the most hashing power since it is the costliest and therefore the most trusted.

Concluding we can say, mining new blocks does secure a distributed consensus and data consistency, but with only probabilistic certainty since every node has a different view of the network. However, the data uncertainty over one block in the main chain drops exponentially with the number of succeeding blocks. Assuming a majority of honest nodes, we reach practical certainty very fast.[7]

3.6 Transaction System

We have shown how transaction objects get accumulated to a block and ordered on a logical time basis. This sub-chapter explains how Bitcoin transactions work and why they are the fundamental construct for the monetary ledger system with tokens.

3.6.1 The Token – *Currency*

Bitcoin, the unit of account on the blockchain, is significantly different from state controlled fiat currencies such as the US Dollar or the Euro. Bitcoins exist purely digitally without any physical or tangible entity, nor are they issued by a humanized system, but rather in compliance to the source code rules.

We answer the questions of what Bitcoins are and how they get transferred. First, we need to clarify that a transaction does never transfer any token on a geographical level - like a data package moves between two locations.[8] Hence, we cannot determine possession of a token by the physical location of a token object since there is no token data object. The Bitcoin blockchain system exploits the concept of Unspent Transaction Outputs (*UTXOs*), that is ownership via cryptographic access. It means, the token only “lives” as a value-parameter in the output of a transaction script. That value-parameter is a simple integer representing the smallest unit, a Satoshi. One Bitcoin equals 100,000,000 Satoshis. The value can be claimed via script execution to use as input for another transaction. If a transaction output has not yet been consumed as an input, it is considered unspent. Anyone who is able to deliver – but has not publicly shown - a successful script execution for that UTXO owns the tokens. These scripts however, rely on asymmetrical cryptography so that usually only one person knows the corresponding private key for access. One single UTXO can be viewed as one atomic impartible unit with any quantity of Bitcoins. Using them as multiple inputs lets them aggregate in size and having multiple outputs in one transaction splits their size.

3.6.2 Addresses - *Access*

As explained in the token section, Bitcoin does not establish any form of account-balance system. Rather the source code rules say, ‘anyone who gets a true Boolean on running the output script of a transaction has the right to use the amount of the value-parameter as input for another transaction’. A good analogy for this concept is a public single-use vault where anyone can throw one item in (=transaction input), but only a person knowing the key code to the vault can get that particular item (=transaction output). To implement this concept digitally, Bitcoin uses a digital signature scheme. The scheme is the Elliptic Curve Digital Signature Algorithm (*ECDSA*), a proven and widely used standard. Bitcoin favors it over RSA, the competing

[7] For detailed calculations with poison distribution see (Nakamoto, 2008, pp. 6-8).

[8] Of course, the transaction itself gets relayed and broadcasted on the network. Cf. the chapter about the network.

standard, mostly because ECDSA provides the same security with shorter key values, which is a wanted property to save storage and bandwidth in a peer-to-peer network. We adopt and extend the schematic diagram from the fundamentals chapter with Bitcoin terms:

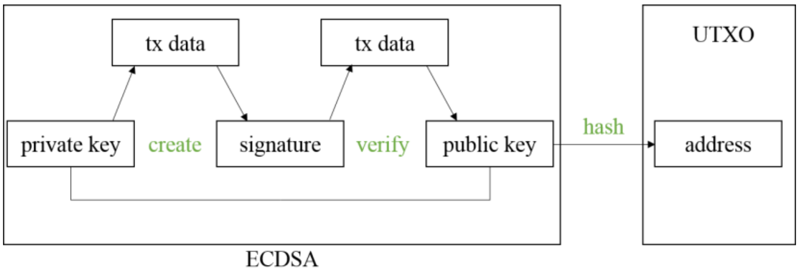


Figure 7 *Bitcoin Address Derivation*

In words, a transaction output script contains a hash of a public key. To execute the script to true and spend the output token value, a spender must provide (1) the *public key*, that when hashed, yields that destination address and (2) a *signature* to show evidence of the corresponding private key. As the hash conversion algorithm, Bitcoin serially combines the SHA-256 and RIPEMD-160 hash functions as well as adding version byte and Base58Check binary-to-text encoding. This procedure is done to receive a very high level of randomness in the hash output and protect against accidental collisions. This makes the hash output a suitable way to express identity. The output script uses this identity as the destination address. An exemplary Bitcoin address may look like this:

1DiTACHiQM2xp8BFyAd1VEHHwcXg1dfHK4

Accessing and owning Bitcoins via that address system offers pseudo-anonymity for humans., since addresses can be traced and linked backwards to the transactions before, but a participant can create new identities very easily at will and offline. Using an address only once and having transactions with as few inputs and outputs as possible brings higher anonymity.

3.6.3 Transaction and Address Graph – *Ownership Structure*

Transferring access of Bitcoin tokens via the model of addresses in output scripts and one-time redemption by using them in an input scripts of another transaction, chains transactions together via the script execution. Since a transaction can have multiple inputs and multiple outputs we obtain a directed acyclic transaction graph, a form of traceable ownership structure for the tokens. In the figure, we draw an extract of a possible graph and bring out the relationship with the blockchain data structure. In this case we mark transactions happening at block two, three and four. The starting tips in the graph with only incoming edges are coinbase transactions. The ending leaves with only outgoing edges are the transactions having UTXOs, hence the current addresses which control tokens. From the transaction graph, we could derive a corresponding address graph indicating the token flow between them.

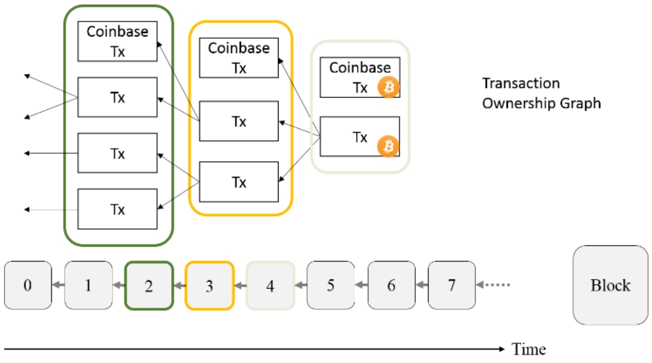


Figure 8 Bitcoin Transaction Graph

We note that in practice users do not usually spend UTXOs every next block as in our linear example. These gaps twist the real transaction graph much more.

3.6.4 A Transaction

Structure

The concepts of tokens, addresses and a deducted ownership structure are crucial features for a payment system. They all happen indirectly inside a transaction. Therefore, we need to dig deeper into the setup of a single transaction to understand how they are achieved. Transactions are broadcasted over the network, where they need to have further meta data as a network package. Mining nodes collects them into blocks. They are unencrypted which makes the blockchain data available for every participant. The main task of a standard transaction in Bitcoin is to transfer the tokens, or more precisely the accessibility of the tokens. Each transaction can embed multiple *input scripts* from other transactions and multiple *output scripts* to other transactions. However, once a transaction exists, all inputs of that transaction can never be used again. Thus, the outputs should consume all inputs minus optional fees for the miner. The transaction system manages change by having an output back to the sender's address or one related to him. The general format of a transaction inside a block is shown in the byte-map below. The main parts are the lists of input scripts and output scripts used for that transaction after their respective counters. A filled *lock time field* at the end hinders that nodes consider the transaction valid before a specific time or block height. This is to support smart contracts with varying use cases for that mechanism such as transferring ownership access between different blockchain systems in atomic swaps.

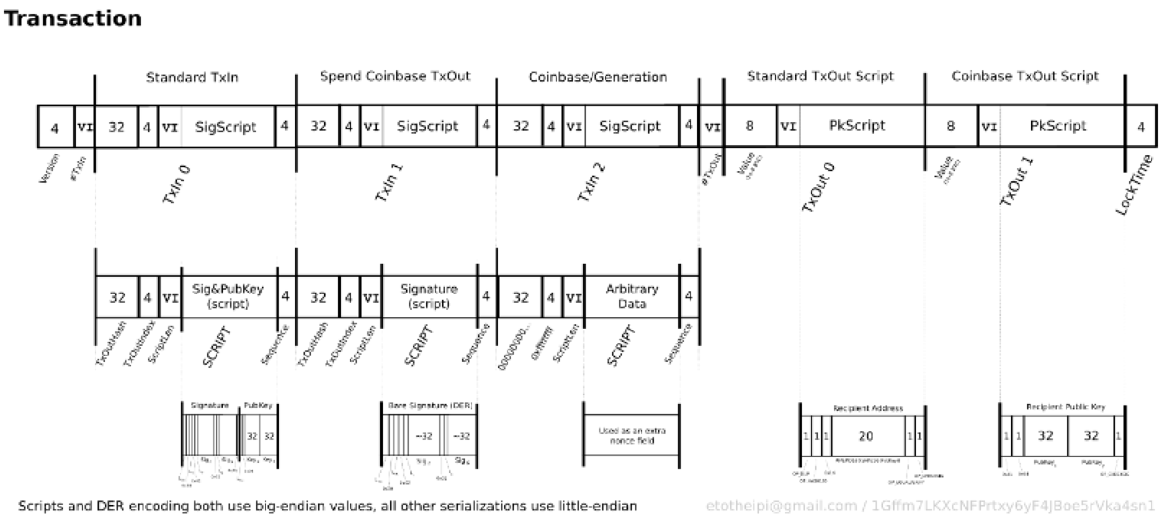
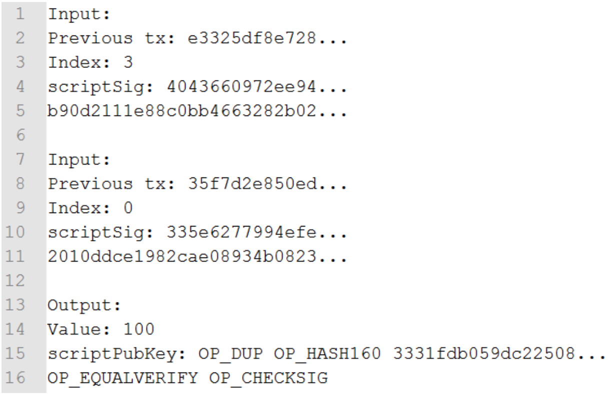


Figure 9 Bitcoin Transaction Byte-map (Bitcoin Technical Wiki, 2011)

Herefrom, we construct an example for the raw data of a standard transaction with two inputs and one output:



```
Input:
Previous tx: e3325df8e728...
Index: 3
scriptSig: 4043660972ee94...
b90d2111e88c0bb4663282b02...

Input:
Previous tx: 35f7d2e850ed...
Index: 0
scriptSig: 335e6277994efe...
2010ddce1982cae08934b0823...

Output:
Value: 100
scriptPubKey: OP_DUP OP_HASH160 3331fdb059dc22508...
OP_EQUALVERIFY OP_CHECKSIG
```

Figure 10 Bitcoin Transaction Data Sample

Input- and Output-Scripts – *Language*

The two input fields in the figure above reference to two different previous transactions. Index values three and zero mean, that these inputs stem from the fourth and first output of their transaction, respectively. The token value in our case is 100 Satoshis. The actual chaining of transactions is a product of concatenating two scripting parts, *scriptSig* (=acting as input) and *scriptPubKey* (=acting as output), and interpreting them. Bitcoin uses its own imperative interpreter with similarities to the scripting language Forth. A stack-based approach with no loops has its purpose in stability and determination of the results, which is advantageous in a distributed peer-to-peer network. To verify authorization and claim the tokens from our single output as input again, a script must return true with no errors when running all instructions from our output script. The broader steps are:

(1) Concatenate scriptSig[9] and scriptPubKey into one script

(2) Stack scriptSig's containing public key and signature

[9] This is a different scriptSig from a new transaction referencing to our output script.

(3) Execute our scriptPubKey containing destination address in hexadecimal

- *OP_DUP* duplicates the public key on top of the stack
- *OP_HASH160* hashes the public key to receive the same value as the destination address "3331fdb…" in our scriptPubKey
- *OP_CHECKSIG* verifies if the signature and the initial public key left on stack correspond

The instructions to redeem Bitcoins in scriptPubKey usually reference an address, but may also use operational codes to use multiple addresses and therefore require multiple signatures. This is a good feature for escrow payments and contracts between multiple parties. Furthermore, it may decompose to a password or never run errorless and burn the tokens as unusable. Bitcoin extends its application ability with these scripts if participants use more complex code, functioning as smart contracts. In our transaction though, we have the standard script.

We demonstrate the process in a picture. The arrows are the same connections as in the ownership structure we introduced previously. The circled box points out one script execution cycle described in the process above.

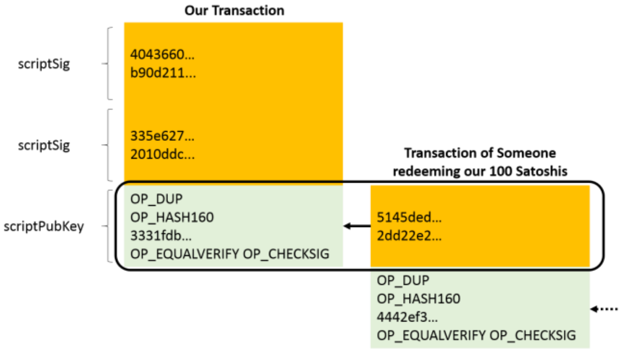


Figure 11 *Bitcoin Script Execution*

A Coinbase Transaction – *Special Case*

We already know that every miner of a new block emerges new Bitcoins via a special form of transaction, the coinbase transaction, he includes in his block. This process is extremely important to control inflation of total coins over time and preserve the value of existing ones. The transaction differs mainly in one part, the scriptSig. It can contain arbitrary data since there is no old transaction output to reference to. This can serve as an extra nonce field to have more flexibility for Proof of Work or to signal a statement or opinion within that data.

Invalid and Manipulated Transactions

A transaction gets rejected if the script executes to false or with errors. This can have several reasons:

Error	Reason
Incorrect hash of public key	To prevent unauthorized access to Bitcoins
Incorrect signature to public key	To prevent unauthorized access to Bitcoins
Output token amount > input token amount	To prevent creating Bitcoins out of thin air
Referenced output already used as input	To prevent double spending of Bitcoins
Transaction has running lock time	To prevent precipitated access of Bitcoins in Smart Contracts
Other unrelated error	To hinder spamming and attacks

3.7 Network

The Bitcoin blockchain is designed for an unstructured peer-to-peer network overlay based on permanent TCP connections. Studies over a period of approximately one month by (Donet, Pérez-Solà, & Herrera-Joancomartí, 2014) on the size of the overall Bitcoin network - via repeatedly asking known peers for their known peers with *getaddr* messages - discovered roughly 880,000 temporarily active IP addresses but only about 6,000 permanent active full nodes. The main tasks of the nodes are to replicate and administer the blockchain data. Administration involves creating, validating and relaying transactions and blocks plus communicating peer connections. So long as a node complies to the communication structure and rules of network messages documented by the code, it participates in the Bitcoin network. That is why there are several clients in different programming languages in addition to our reference client primarily written in C++. A node might also alter its own code such that there is different default behavior or mining strategies. For example, he could deny relaying all new blocks but the ones he mined himself, which is a totally valid behavior. Furthermore, it is possible to store only the block headers without validating all transaction scripts and ask other nodes to verify the existence of a transaction in a specific block on demand. This is called Simplified Payment Verification (*SPV*).

3.7.1 Peer-to-Peer vs. Client-Server Approach

Running a blockchain on a single server by a single authority makes it prone to manipulation of data by that authority and exposes the risk for a biased selection of allowed participants. Also, the payment system would be fully dependent and the authority could easily shut it down or change it at any time at will. What could remain a trust-indicator for consumed hash power by the central party is the Proof of Work. Nonetheless, it is a piece of data, which is costly and time-consuming to produce but easy to verify, preserving a uni-directional and random nature. However, a fully replicated blockchain on a peer-to-peer network reduces the need for trust with a power shift to many distributed nodes instead of a central entity. These are the principal reasons why there is no Bitcoin authority server, but many peer-to-peer connected nodes. The huge trade-off though lies in the effort and time for synchronizing the peers, which leads to a much slower payment process. In fact, the question on how to scale on demand - that is to reach validation times for transactions and therefore an average throughput of transactions per second at a similar level as a centralized client-server model - remains unsolved to date.

3.7.2 Peer Communication and Discovery

To bootstrap a new node, a node first needs to download the protocol code over http from known Bitcoin-related websites or get it from someone else trusted. He can discover neighbor peers using (1) some voluntarily *DNS services*, (2) *hardcoded IP seeds* in the source code, (3) IP connections in *local storage* he knew from his last time active or (4) via *manual import*. The variety aims to assure joining the network is simple and setting up a node is cheap. To maintain an active and healthy network, all peers regularly ping if their connections are still online and kick them if not. They further dump spamming peers, that relay invalid transactions or blocks. Additionally, a node can ask his peers via *getaddr* to learn some of their connections chosen at random. This endeavor leads to a random topology, which reduces the risk for nodes getting isolated by an attacker on purpose. (Tschorsch & Scheuermann, 2016, pp. 13-15) None of the behavior and parameters are set in stone, but the more percentage of peers are honest, the quicker the network overlay stabilized as supposed. Leaving is trivial and can be done at any time.

3.7.3 Propagating Transactions and Blocks

A logical database model normally uses *data partitioning* for manageability and performance. In Bitcoin, transactions and blocks can be seen as these distinct independent parts, easily identifiable via their unique hash. They are the only actual blockchain data and therefore the only to send over the network. Besides all blocks, a full node locally tracks a list of UTXOs per default to validate all newly incoming transactions and blocks more quickly by executing the transaction scripts. If an incoming transaction tries to use the same UTXOs as another one or references to an output already spent, a node will ignore it and not propagate it any further. This is to avoid double spend attempts.

Synchronizing the Bitcoin network relies on a combination of pull- and push-based *dataflows*. Pulling data from neighbors is especially useful to bootstrap a new node or let a node catch up after longer inactivity. Besides that, SPV-nodes can request verification data for a single transaction from neighboring full nodes. On the other hand, the pushing mechanism resembles controlled flooding. Nodes do not propagate all new incoming transactions to all neighbors, but rather a randomly chosen subset of transactions to most nodes and all transactions to only a randomly chosen subset of nodes. This is called *tickling*. (Tschorsch & Scheuermann, 2016) In both cases, nodes send a list of the transaction hashes first and the neighboring node pulls the ones he needs with a *getdata* message. This combination saves bandwidth significantly, so that it is not an issue to date. Peers propagate new blocks in a nearly identical way. An additional default rule for conflicting validations of transactions and blocks is the *race condition*. It means to accept what you her first. In combination, (1) communicating the presence of new data, (2) validating all transaction scripts - equally in blocks - and (3) propagating via tickling induce significant latency in the dataflow. A study by (Decker & Wattenhofer, 2013) shows that the mean time for a node to see a block is about 12.6 seconds. Each additional kilobyte of data for a standard one-megabyte block costs about 80 milliseconds of additional latency until a majority of nodes know about it. The tool based on the paper reveals that to reach a 50th percentile of nodes a transaction and a block each need about one to five seconds. A block is of bigger size, but also of higher priority to broadcast. In contrast, to reach a 90th percentile of nodes it already could be up to 120 seconds.[10] That is why the target time to create new blocks with a maximum of one megabyte adjusts approximately ten minutes via the proof of work. With this network design, it is just not feasible to reduce the consensus time or increase the block size dramatically. This scalability bottleneck limits the current throughput to less than ten transactions per second.

3.7.4 Attack Vectors and Implications

As for every computer system, the Bitcoin blockchain system is vulnerable to attacks from dishonest or spamming nodes. To shed light on what types of attacks are crucial to the payment system, we juxtapose covered features of Bitcoin with their addressing issues.

Bitcoin Feature	Addressed Issues
All data is purely public	-Eavesdropping (4)
All blockchain data gets fully validated and replicated on every persisting node	-Persistent data modification (1) -Sybil attacks (3)
Unstructured peer-to-peer network with semi-random and semi-permanent connections and relays	-Encapsulation of a subset of nodes (2) -Denial-of-Service attacks -Man-in-the-Middle attacks -Packet sniffing attacks (4) -Sybil attacks (3)
Pseudo-anonymous addresses based on a digital signatures scheme	-Identity spoofing (4) -Unauthorized-access attacks, like password-based (4) -Manipulation of personal data
Local IP reputation and timeout system	-General network spamming
Transaction costs	-Clogging with valid transactions

Apart from Bitcoin facing most of the mentioned issues, we need to look more closely into four problems annotated in the table, that are non-trivial and possible threads:

(1) Persistent data are valid blocks - with valid transactions - inserted in the main chain. The block itself cannot be altered validly, however the main chain can get outpaced by any other chain striving to become the main one. This race attack is inevitable assuming one attacker controls more than half of the network hashing power, but it becomes exponentially unlikelier for older blocks. To summarize, there is no absolute guarantee on data consistency, but the probability increases with time elapsed approaching the 100% consistency limit quickly.

(2) (Apostolaki, Zohar, & Vanbever, 2017) have pointed out that large ISPs could effectively split up disjoint parts of the bitcoin network with routing attacks since they control traffic interfaces. This however would only affect nodes temporarily since there are many other

[10] Cf. (bitcoinstats.com, 2017)

ways for a node to realize it is not persisting the main chain any more. For instance, one could access trusted public entities over http, servicing as block explorers. Further, ISPs could also undetectably delay data propagation by a maximum of the connection-timeout limit set in Bitcoin.

(3) Proof of work effectively prevents Sybil attacks on mining and therefore token influence on system level. However, slowing down a part of the network and distracting connections is theoretically possible if an attacked node or set of nodes is not connected to an honest node of the network's majority.

(4) A system cannot prevent all privacy-related attacks typical for any computer network. Attacks can reveal the identity of single nodes. The entire transaction system though, is not affected so long as the underlying digital signatures scheme is mathematically operational.

3.8 Outlook

3.8.1 Updating - BIPs and Forks

We previously defined the Bitcoin source code as the identity of the blockchain.[11] As for every software, it needs to be maintained, adapt to environmental changes and dynamically evolve over time. In a distributed peer-to-peer database this is a tedious process, since there is no central authority to roll out a new version. The network requires consensus over an update and an exact date to not split the blockchain. There are many influencing stakeholders to satisfy, foremost developers, miners, investors and users.

The *update process* wears out in three major steps: proposing, signaling and forking. First, some developing community publicly proposes a new design concept and implementation to improve Bitcoin. These are called Bitcoin Improvement Proposals (*BIPs*). Second, all stakeholders can signal their attitude towards a discussed BIP. Investors typically upvalue or devalue the tokens. More importantly, miners can include a *vote* by filling the input script field in their coinbase transaction of their mined block with informational data. This tends to make the hashing power of mining nodes the governmental system of the blockchain. The third step is the synchronous transfer from the old source code to the updated codebase. This causes a fork of the blockchain. There are two forms. The new code either tightens the existing rule set which makes it backwards compatible (called: *soft fork*) or it loosens the rule set which makes it backwards incompatible (called: *hard fork*). Both cases lead to a split in two distinct blockchains if some nodes stick to their original client software and at least one event with conflicting rules occurs. In general, a soft fork sets gradually pressure to deploy the update and a hard fork abrupt.

Example An example of a hard fork would be to allow blocks of bigger size than one megabyte. As soon as one updated node finds a block of bigger size, a split happens if some other nodes wouldn't want to update. A complementary soft fork would be to set an upper blocksize limit of smaller than one megabyte. If a not-updated node finds a block between the new and the old limit, a split occurs if nodes continue both source codes.

[11] We remind, a node can run whatever source code it prefers or modify the reference client. Identity exclusion first comes from violating the rules and getting ignored by peers.

3.8.2 BIP-141-144: Segregated Witness

As of August 2017, Bitcoin is undergoing a fundamental update which alters transaction format and block structure to slightly improve scalability and transaction malleability. The concept is called segregated witnessing. As explained in the block chapter, the raw transaction data occupies more than 99% of all data space in a block with the transaction signatures taking up about two thirds of it. The new source code extracts the *signature* fields from the inputs and appends them at the end of a transaction – serialized for storage and network relay. All witnessed signatures in a block are then Merkle-hashed and the root included in the coinbase transaction. This inclusion guarantees the link of the signatures to the block header. Nodes using former client software see an empty signature field and consider witnessed transaction inputs as validly spendable by anyone which makes it a soft fork. We illustrate the new design.

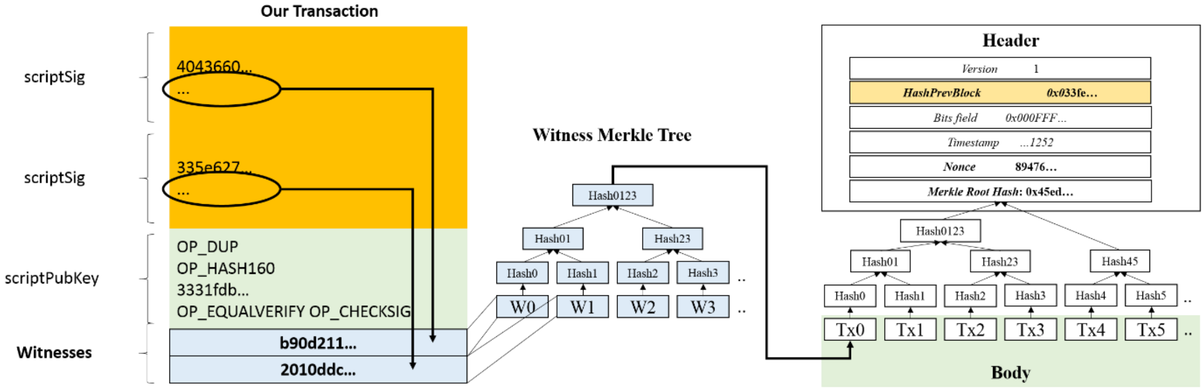


Figure 12 *Segregated Witness Diagram**

*Our diagram omits parts of a transaction to reduce complexity and focus on the structural changes to our example. A real segregated witness transaction embeds a new type of transaction script, which would still need to run without errors to be valid. Therefore, output operations to redeem tokens adapt, too.

3.9 Wrap-up

The overall tasks of each individual components are distinct. In a nutshell, the Bitcoin token is a chain of digital signatures enabling authorized-only and tamperproof access via transaction scripts. The unique block setup comprises transactions effectively and orders them by chaining subsequent block headers together with hash pointers. Double spending is prevented with a brute-force race where nodes of an equipotent distributed peer-to-peer network aim to construct the longest chain with the most Proof of Work. Full database replication circumvents a single point of failure and mitigates network attacks. The major achievement is cryptographically ensured trust in the notion of a global logical time. This results in the ability to transfer value-binding data instead of information-binding data. With the right semantic, private key data consumes the token value and binds redemption to other key data.

We compare the Bitcoin blockchain system with a distributed and deterministic stateful machine. The set of binary UTXOs form the state. Finalizing transactions in blocks are state transitions. Validity checks at every node guarantee determinism under uniform rules.

4 Ethereum Protocol

4.1 Introduction

In this chapter, we interpret the Ethereum blockchain system in detail. We focus on demonstrating how and why it differs from Bitcoin. We notice the young history of Ethereum with only about two years after the first release. Therefore, we recognize the source code is since and still under heavy development by the Ethereum Foundation team. Several updates occurred even during this thesis on three clients in use in different programming languages – all with some disparities. We therefore decide to mainly stay with the technical specification from the official yellow paper by (Wood, 2014) and for the ideological concept with the initial whitepaper by (Buterin, 2013) as the system's source of truth. We remark, that we do not cover basic concepts if they are akin to Bitcoin, such as

- the consensus algorithm of mining with Proof of Work and a hash-based block connection mechanism,
- the use of a virtual token,
- the use of a peer-to-peer network structure and all its communication affairs,
- the updating process via BIPs (in Ethereum: EIPs) and forks.

4.2 Background and Application Purpose

Since the rise of Bitcoin as a payment system, developers engineered many alternative blockchain protocols. Some are clones from the Bitcoin source code, thus very alike, others have severe distinctions to pursue a variety of applicable use cases. In late 2013 and 2014 Vitalik Buterin initially proposed the concept for Ethereum finalized in his white paper. The yellow paper published by Gavin Wood specifies it further technically. The Ethereum Foundation led by Buterin and Wood implemented the initial software with a first release in mid-2015.

Ethereum is an alternative blockchain protocol with a general-purpose approach to facilitate building all transaction-based state machine concepts on top of it. It does so by being the abstract foundation layer, "a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions." (Buterin, 2013, p. 13) Besides the model of a currency, transaction-based state machines may handle other assets, such as stocks and real estates, or trace items in their supply chain.

4.3 Account Model – *World State*

4.3.1 From UTXOs to Accounts

We concluded the chapter about Bitcoin with comparing it to a deterministic state machine, where the set of UTXOs defines the current state and finalizing transactions in one block marks a transition from one state to the next. This model - using simple stack-based script execution and multiple transaction in- and outputs - allows to easily determine the ownership structure and is very resilient to script failures of any kind.

Ethereum however, needs a more powerful, a Turing-complete scripting language to be able to reproduce any possible application of a transaction-based state machine. Therefore, it uses the model of attributed account objects and maps them to identifiable addresses as the current world state. This concept represents an abstract form of a UTXO.

4.3.2 An Account Object

An account object is the state σ of an address. It comprises four fields to represent its respective state, a *nonce*, a *balance*, the *storageRoot* and the *codeHash*.

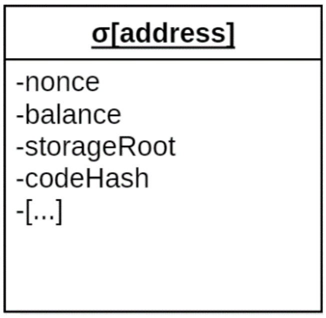


Figure 13 *An Ethereum Account Object*

The nonce is a scalar value representing the number of transactions sent from this address. The balance is the amount of Ether[12] announced in the smallest token unit Wei (1 Ether = 1,000,000,000,000,000,000 Wei) owned by the address. These two fields only bear information the UTXO-model holds too. The number of transactions equals the outgoing edges in the address(!) ownership graph of Bitcoin for that address. The balance equals the sum of all value-parameters of all UTXOs redeemable by that address. A distinction is that the scalar balance parameter of an account eliminates the need for transactions with multiple inputs and outputs to have dividable measurable units.

The knack to build the abstract version of a state machine lies in the other two fields. The *storageRoot* holds the 256-bit hash root of a Merkle Patricia tree. The tree encodes the storage contents of that account, which is basically a key-value mapping of integers. The *codeHash* is the hash of Ethereum Virtual Machine (*EVM*)[13] code optionally belonging to this address. It is the only immutable field after its construction via a transaction. It therefore lets us categorize accounts into two separate types, the ones containing programmed code and the ones not having

[12] An Ether is the native Ethereum token, the conceptual counterpart to a Bitcoin token.

[13] Cf. the chapter about Solidity and EVM.

any, thus an empty codeHash field. The latter ones are externally controlled by humans - more precisely by their corresponding private keys much alike in Bitcoin. But the first ones, called Smart Contract accounts, are more sophisticated because they are controlled by nothing but their code alone and therefore act as an Autonomous Object. They allow for Turing-complete programming tasks and communication between accounts via internal messages. (Wood, 2014, pp. 3,4)

4.3.3 An Address

Getting uniquely identifiable addresses in Ethereum resembles the process in Bitcoin. Public keys are constructed from private keys via ECDSA and used as input for a hashing algorithm. The last 20 Bytes of the output are the associated address to the private key.

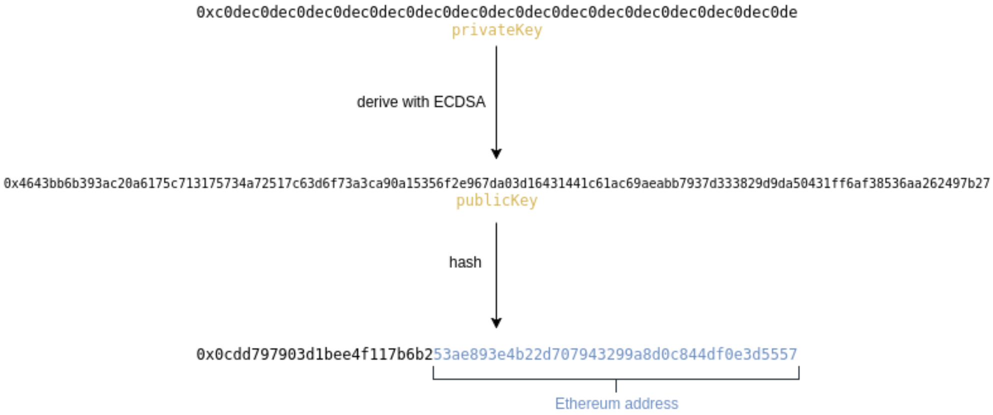

0xc0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0dec0de
privateKey
derive with ECDSA
0x4643bb6b393ac20a6175c713175734a72517c63d6f73a3ca90a15356f2e967da03d16431441c61ac69aeabb7937d333829d9da50431ff6af38536aa262497b27
publicKey
hash
0x0cdd797903d1bee4f117b6b253ae893e4b22d707943299a8d0c844df0e3d5557
Ethereum address

Figure 14 Ethereum Address Derivation (CodeTract, 2017)

However, the hashing algorithm is not SHA-256 based as in Bitcoin, but utilizes the *Keccak-256 (SHA-3)* algorithm. This algorithm is another standard set by the National Institute of Standards and Technology (NIST) after some attacks on the SHA-2 family were discovered, which eventually turned out to not be an issue.[14] Neither hashing algorithm class offers decisive advantage over the other for Ethereum nor for Bitcoin. The NIST considers both unbreakable to date. The Ethereum Foundation however plans to update the possibility for accounts to individually implement their own digital signature scheme with any hashing algorithm. (Buterin, 2015) This would allow for instance a combination of both SHA-2 and SHA-3 consecutively to make identity more secure or one could use the Lamport one-time signature scheme for quantum resistance. This shift away from a version hard-coded into transaction processing to an account-implementable version brings a major flexibility advantage for Ethereum to fit for a variety of application needs. This especially copes with developments in cryptographic security in the long-term.

[14] SHA-3 are based on sponge functions and have a completely different approach than SHA-2, which fall under a class to focus on collision-resistance via collision-resistant one-way compression functions. Cf. (Bertoni, Daemen, Peeters, van Assche, & van Keer, 2012) for further reading.

4.4 Transaction System – *State Transitions*

4.4.1 A Transaction

Structure

An Ethereum transaction consists of seven fields as shown in the overview figure below. The *nonce* stores the number of transactions sent by the sender. The *to* field specifies the receiver's address, which is always exactly one since the account model does not need multiple outputs. The scalar *value* represents the amount of Ether in Wei to be transferred. And the last field contains the relevant output from the *signature* algorithm to show evidence of knowing the private key. (Wood, 2014, p. 4) As explained in the account model, Bitcoin's UTXO-model can also reveal all information from these fields.

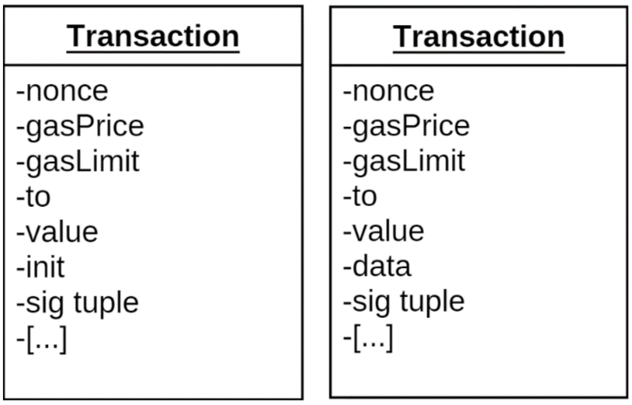


Figure 15 *Ethereum Transactions for Contract Creation (left) and Message Call (right)*

What goes beyond Bitcoin's capability are the *init* or *data* field, a byte array of unlimited size. These alternative fields lead to two groups of transactions. One, that initializes the creation of a new smart contract account, a new autonomous object, with its EVM code transcribed from the init field. And the other, more frequent one, specifies the input data for a message call via the data field. (Wood, 2014, pp. 4,9,10)

Messages

Messages are all data and values passed between two accounts. Thus, a transaction creating a contract is not a message. But the second transaction type of transactions are externally signed messages. These call an account's contract code if existent. The code operations may then recursively invoke new unsigned messages acting as input variables for further autonomous account objects. These unsigned messages are always internal between two smart contracts and therefore are not recorded in a block nor is their execution delayed by the mining process. (Buterin, 2013, p. 14)

Gas

The fact of supporting Turing-complete code with arbitrary length complicates the execution process tremendously and makes it error-prone. One problem to be solved particularly is to disallow infinite looping or any other non-deterministic message calls, i.e. dissolving the *Halting Problem*. This is tackled via the *gasPrice* and *gasLimit* field of every transaction. Every programmable computational step and storage operation has a universally agreed fixed cost in terms of Gas, an abstract and only transaction internal unit. Every transaction must in advance specify to associate a maximum amount of Gas - via the gasLimit field - at a self-specified price in Ether - via the gasPrice field - to be consumed by it and all its subsequent message calls. Exceeding the limit is not possible and leads to precocious abortion of the execution including reverting all changes made. Paid Ether are added as fees for mining and excessive Gas will be refunded back to Ether for the transaction. (Wood, 2014, pp. 7-9) Alongside achieving *quasi Turing-completeness*[15], another bearing effect of this cost mechanism is that it prevents spam and hinders Denial-of-Service attacks.

4.4.2 The Ethereum Virtual Machine - *State Transition Cycle*

As we know from Bitcoin, successfully executing a transaction script alters the binary state of transaction outputs - spent or unspent (=UTXO). Within Ethereum, transactions and messages can trigger more complex state changes of accounts. Provided they are intrinsically valid and do not run out of associated Gas while executing, which also invalidates them. Intrinsic validity includes a correct nonce, a valid signature, enough balance from the sender and that the data is generally well-formed.[16] Executing all transactions - and their subsequent messages - correctly is a part of finalizing the block with a valid world state of accounts.

The main distinctive feature lies in the *state transition cycle* defining the state changes by a single transaction. Every transaction or message can call contract code of an account by having that account's address as recipient. Besides, it can send Ether and read and write storage. Apart from the persistent key-value storage of all accounts, there are two forms of volatile storing space, a LIFO byte-array stack and an infinitely expandable word-array memory. The code execution itself then follows the model of the Ethereum Virtual Machine (*EVM*). The EVM specifies how to alter the system's state given the implemented bytecode instructions and a *tuple of environmental data*, such as the transaction value and the current block header. Hashing of current block header fields can act as a source of distributed randomness. The EVM further includes the *world state* with the persistent key-value storages of all accounts and the *machine state* mainly tracking available Gas and the volatile storage items. It interprets the low-level stack-based bytecode language, where each byte represents an operation. A single operation may for instance be to pop the top two stack items, multiply them and push the result back to the stack. After getting and applying an instruction and updating the stack, the available Gas for that transaction or message is reduced in accordance with the price for that operation. This *iteration function* ends with updating the world state and repeats until there are no more code instructions left or an error occurs. Errors include running out of Gas, which leads to a stop and reverts and invalidates all changes of that transaction or message. The EVM successively executes the entirety of transactions to include the valid ones in a block and finalize a valid state transition. (Wood, 2014, pp. 10-12)

We model the overall process of a state transition from one single block.

[15] Quasi refers to the fact of setting an intrinsic boundary through the Gas parameter.

[16] Cf. chapter 4.4.3.

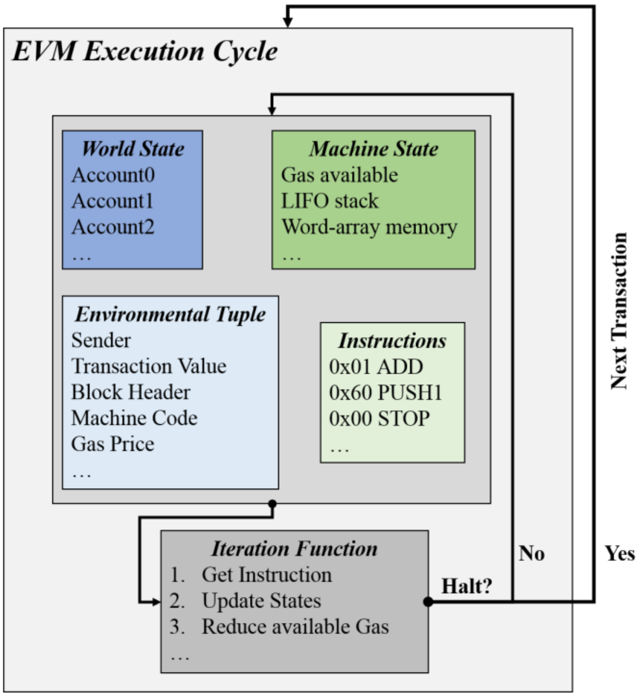


Figure 16 *Ethereum State Transition Cycle with EVM*

Since all nodes in the network must compute every single operation of each state transition and validate its correctness, Ethereum currently struggles to scale in terms of transactions per second much like Bitcoin does.

A Transaction Receipt - *History Data*

The outcome of a transaction with its log data[17], the Gas used and most importantly the new post-transaction state is stored in a new data object, a transaction receipt. (Wood, 2014, p. 5) It can be compared to an artefact of the results for later use, such as proving existence of a valid transaction or optimized searching for created contract addresses.

4.4.3 Scripting Language – From Solidity to EVM-Bytecode

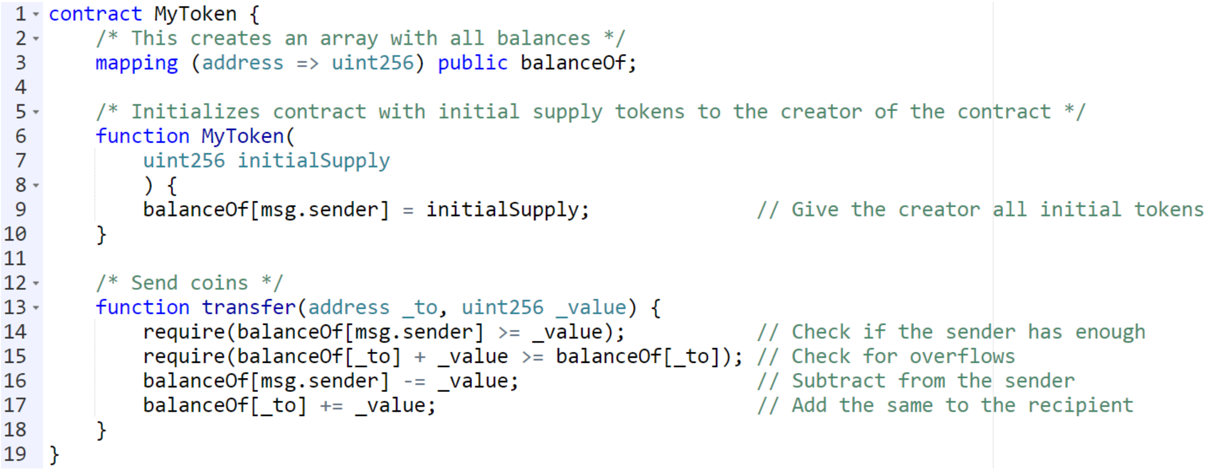
Bitcoin uses a scripting language to check validity of a transaction by taking the output scripts from the referred transactions as their input scripts and combining them with the new output scripts on how to claim the tokens in the future. A successful execution transfers access of the tokens. Ethereum diverts this process fundamentally. Since Ethereum employs account objects - simply having a balance attribute - associated with addresses, there is no need for the input-output claiming model. Instead, the EVM-model takes care of transfers by simply validating transactions - foremost signatures - and actively maintaining the world state of all accounts immediately. Furthermore, Ethereum lets a transaction transcribe bytecode via the *init* field for contract creation, which then exists permanently and immutably as the code associated with that new account via its *codeHash* field. The method of serialization is an Ethereum-specific Recursive Length Prefix (*RLP*) encoding, which is minimalistic and guarantees consistency. The purpose of RLP is to encode arbitrarily nested arrays of binary data, bearing structure, not

[17] Including a Bloom filter, a space efficient probabilistic data structure to test if an element is in a set, thus easing searches. The effect of false positives is negligible.

data types.[18] All existing contracts are thereby globally accessible by all other accounts via their contract ABIs making all functions reusable for unlimited execution[19]. This is a major complexity and flexibility advantage over Bitcoin's one-time scripts. But complexity bears security risks especially through human influence. Humans always act as the initiator of a contract in the first place, much before the EVM interprets them. To handle complexity and readability for developers, the low-level bytecode living on the blockchain gets compiled from high-level languages beforehand, most commonly Solidity. We want to give a quick overview of the programming language and known issues.

Solidity

Solidity is a high-level programming language for developing smart contract code with a similar syntax as JavaScript. It supports inheritance, has static data types and is contract-oriented, which essentially means contracts are the object with attributes and functions.[20] These powerful properties let us create a currency token on top of Ethereum in a few lines of code:



```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(
        uint256 initialSupply
        ) {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
    }
}
```

Figure 17 *Solidity: Minimum viable Token Contract (ethereum.org, 2016)*

This exemplary contract supports a minimal viable currency with a function to transfer tokens between two accounts. But we should consider all tokens are onetime issued to its creator without any mining or other specific features. A bundle of smart contracts interacting to form a single application are called distributed application (*DAPP*). If they provide self-organizing governance features, one might call them a distributed autonomous organization (*DAO*) or a distributed autonomous company (*DAC*).

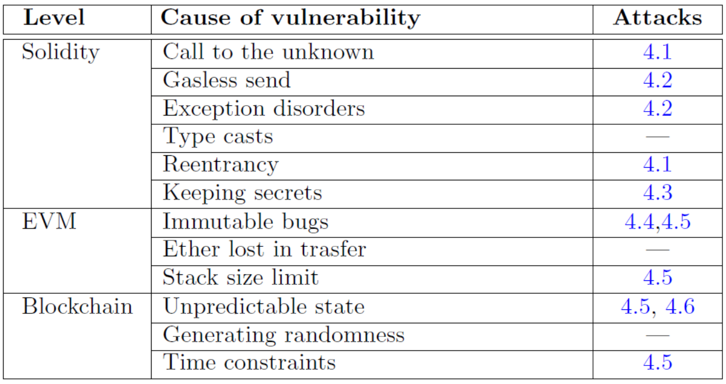
We can understand how fast complexity can easily lead to development mistakes if one for instance wants to technically clone the Bitcoin token or implement any other asset structure. Since smart contracts are (1) permanent and (2) implement currency-like tokens, small errors can entail huge financial impacts and limitations, that cannot be reversed with simple updates. In fact, a survey of attacks by (Atzei, Bartoletti, & Cimoli, 2016) revealed most causes of

[18] Cf. (RLP, 2016) for details.

[19] Function calls are compiled into transaction data. The EVM technically does not support functions. It rather uses uniquely identifiable signatures based on the function's name and parameters, which then links to the compiled bytecode of the function.

[20] Cf. (Solidity, 2017) for full specification.

vulnerabilities stem directly from the high-level language solidity with something as basic as type casts.



Level	Cause of vulnerability	Attacks
Solidity	Call to the unknown	4.1
	Gasless send	4.2
	Exception disorders	4.2
	Type casts	—
	Reentrancy	4.1
	Keeping secrets	4.3
EVM	Immutable bugs	4.4,4.5
	Ether lost in trasfer	—
	Stack size limit	4.5
Blockchain	Unpredictable state	4.5, 4.6
	Generating randomness	—
	Time constraints	4.5

Figure 18 *Ethereum: Taxonomy of Vulnerabilities in Smart Contracts (Atzei, Bartoletti, & Cimoli, 2016)*

The most famous exploited vulnerability was the attack on a bundle of contracts called 'The DAO' in June 2016. Technically it was recursively calling a function within itself before falling back to the next statement. The impact led to a hard-forked split of the entire Ethereum blockchain initiated by the developing community. (Atzei, Bartoletti, & Cimoli, 2016, pp. 13,14)

To prevent severe attacks with possibly huge financial consequences, (Jentzsch, 2016) recommends implementing standards of security patterns such as updateable contracts, time delays for reactions and regular invariant checks. Mathematical proves to verify contracts with minimal complexity would even close some typical security holes completely.

Process Overview

To conclude this sub-chapter, we summarize the general implementation process from Solidity to EVM-bytecode.

Not on the blockchain:

(1) Developer implements high-level contract code in Solidity

(2) Compiler compiles high-level code to binary data format

On the blockchain:

(3) Human initiated transaction publishes binary data, i.e. deploys it on the Ethereum blockchain. It happens via the *init* field according the RLP-encoding into EVM-specific bytecode

(4) EVM transcribes the bytecode to an account object, the new contract

(5) The contract can now repeatedly interact with other contracts via their ABIs using the EVM as interpreter for their messages

4.5 Blocks and Mining

Ethereum's mechanism for chaining blocks sequentially together with hash pointers is technically almost the same as in Bitcoin. But there are several subtle unique design decisions a standard user would probably not notice. We identify them, investigate why they are chosen and compare them to Bitcoin.

4.5.1 Tokens and Inflation Model

The Genesis block of Ethereum creates account states with a positive token balance for investors and developers.[21] This funds development and helps bootstrapping the system in a distributed manner. This initial coin offering (*ICO*) scheme is typical for projects that are conceptualized much later than Bitcoin since there is already an established blockchain community. Another difference is, that the fixed mining rewards from coinbase transactions follow a constant increase of total coins instead of a slowly halving increase tending to zero in Bitcoin. The relative inflation rate drops gradually too. It disincentivizes storing tokens, since the total supply rises steadily creating higher pressure on the value. Contrariwise it diminishes the cost gap between transacting tokens relative to not transacting.

4.5.2 A Block

Blocksize and -Interval

Ethereum targets a much lower block creation time of 17 seconds, which caps right above the mean time for a node to see a block in the Bitcoin network. This reduces the transaction-to-execution delay. But it escalates the danger of a chain split due to simultaneous mining on different blocks since network latency still exists. A blocksize limit is also vanished and instead indirectly covered via a Gas limit for each block. The maximum Gas is a much fairer indicator for the overall services of the system nodes since it not only includes the data size, but rather all done computational operations.

Body - Data

Besides the list of transactions as in Bitcoin, a list of *ommer* headers is included in the block data.[22] Ommers are valid stale blocks not in the main chain having the same parent's parent block as the current one. It countersteers the splitting desire if nodes create blocks simultaneously since a part of the normal mining reward is granted to the node who calculated an ommer block.

[21] Cf. (etherscan.io, 2015) for full block data.

[22] Uncles is used synonymously for Ommers.

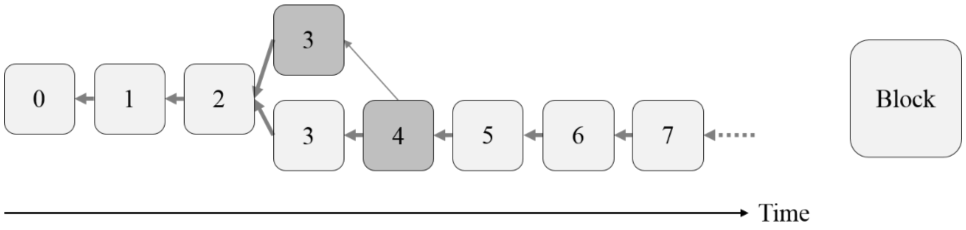


Figure 19 *Ethereum Block #4 granting a stale Block at Time 3 as Ommer to de-incentivize a Race*

Header

We draw an overview of an Ethereum block in the figure below. Looking at an Ethereum block header, we observe similar data fields as in a Bitcoin header, most importantly including the *previous hash* and a *nonce* for mining. However, Ethereum includes more data. A *beneficiary address* field substitutes the function of a coinbase transaction in Bitcoin. The header also includes some check fields, such as the block's *Gas limit* and *Gas used*, the *block's height* and a *mix hash*, which is the result of the intermediate step of the hashing function to allow faster block validation for light clients. Further, an Ethereum header must connect all most recent state and transition data as input for the hashing function. These are the *root hashes* (1) of the account-address mapping, i.e. the *world state*, (2) of the block's *transactions* and (3) of the block's transaction *receipts* including logs as history data. (Wood, 2014, pp. 5-7) This is a necessary alteration to be able to trivially revert to old Ethereum states. In Bitcoin, the state, namely UTXOs, could be recalculated much faster if needed. Lastly, the *ommers hash* links all ommers stabilizing the main chain.

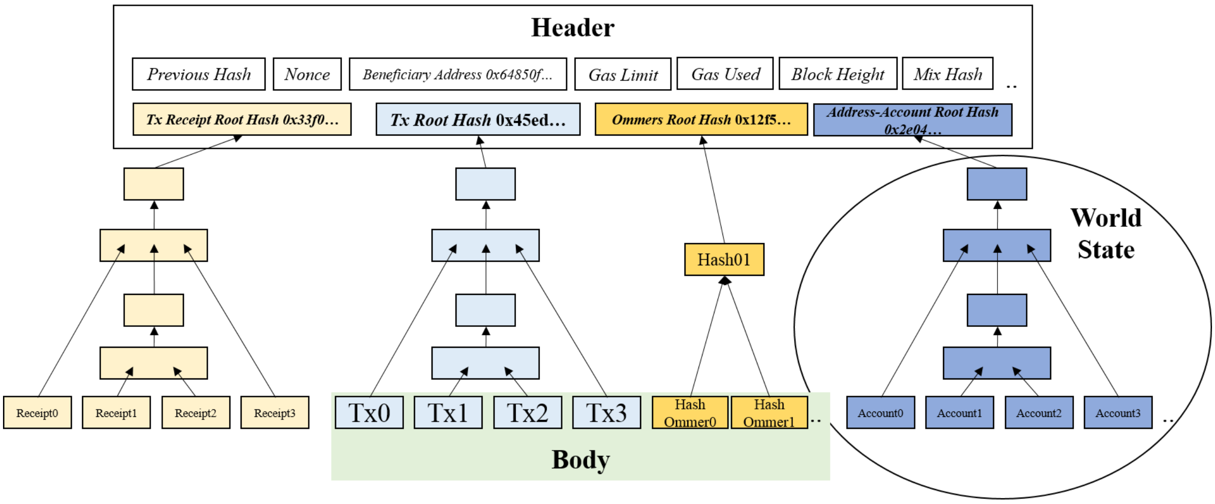


Figure 20 *Ethereum Block Overview*

Currently all full nodes store states, only a fraction though saves history - transactions and receipts. Storing the entire state with every single block obsoletes saving the full blockchain history, but may seem inefficient, which is why Ethereum modifies the structure of a Merkle tree for the state root into a Merkle Patricia tree, a radix trie. The root then comprises all world state data.

Ethereum's Merkle Patricia Tree

Because all data in Ethereum is serialized using its recursive length prefix (*RLP*) method, a prefix-oriented radix trie serves as a more space efficient data structure compared to the bare

Merkle trees in Bitcoin. The tries bring the RLP data in a canonical form. They are immutable, fully deterministic in storing all key-value bindings and offer log(n) inserts, lookups and deletes. Basically, four Bits compose a nibble which are the prefix differentiator for sorting the data leaves into the node structure. Hence, changing the world state from one block to the next only requires updating a few entries or pointers whereas the rest of the trie in the backend database stays untouched. (Wood, 2014, pp. 17,18)

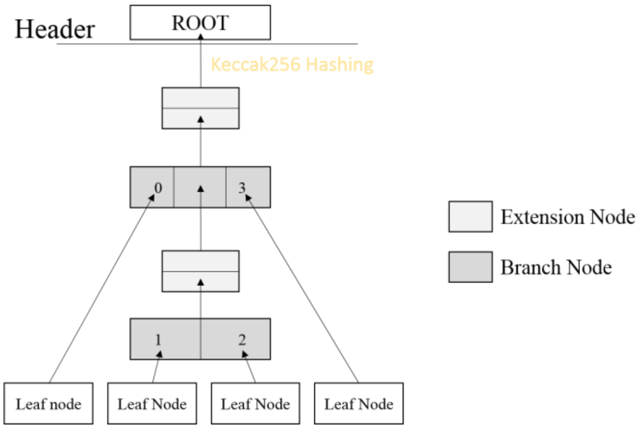


Figure 21 *Ethereum Sample Trie*

Leaf nodes store the data. A branch node in the radix trie can for example be a byte-form indicator for a data type branching all children according their type. In practice, a trie consists of thousands of leaf nodes and to flatten the trie the RLP design enables branches with 16 nibbles (hexadecimal) to branch from. Four trees utilize this structure to link their root with the header: (1) the key-value storage mapping of every single account, (2) all account-address mappings forming the world state, (3) the block's transactions and (4) the block's transaction receipt. (Merkle Patricia Trie Specification, 2017)

4.5.3 PoW Mining - Ethash

The Proof of Work mechanism with brute-forcing hash values in a target range remains the same as in Bitcoin, but with more input variables from the block header. Miners select transactions because of the included Gas fees converted as Ethers, where a commercial miner would favor transactions with a higher Gas price. In Bitcoin mining, the SHA-256 based Proof of Work algorithm allows for application-specific integrated circuits (*ASICs*), which means a node can significantly accelerate calculating hashes by using adapted hardware circuits. To avoid outperforming a standard home computer, Ethereum tries to be ASIC-resistant by implementing a proper hashing algorithm called Ethash. Its hashing function builds upon a multi-level directed acyclic graph (*DAG*) to achieve memory hardness. Memory hardness is if the time to compute a new hash primarily requires holding large amounts of data and therefore is bound to memory instead of pure processing power. In general GPUs perform best with Ethash. (Ethash, 2017)

4.6 Updating Outlook

With the current design, every single peer-to-peer node validates every block individually. Keeping the balance between the size of each block and the target time to create a block considering network latency is the bottleneck problem for scalability. To address scaling with higher transaction demand per second, Ethereum plans to introduce mainly two new approaches.

Proof of Stake

Instead of using Proof of Work as a costly way to distribute mining power, another protocol called Proof of Stake (*PoS*) will gradually replace Proof or Work. There are competing versions for Proof of Stake, but the common idea is to let nodes stake value by means of Ether tokens for a specific period in a smart contract. Then the contract makes the token return dependent on the nodes lining up for creating a new block with rewards and voting to validate other blocks. (Proof of Stake FAQ, 2017) Further benefits of this approach are that it cuts the energy cost used for Proof of Work, it distributes mining power two all Ether holders equally and it gains autonomy for validation inside the system, which impedes majority attacks since the system majority would only harm themselves.[23]

Sharding

Another desired concept to implement is sharding. Like sharding of centralized databases, an Ethereum node would only keep track of a small portion of transaction data, but can verify the entire state by Merkle-verifying everything if needed. Essentially, a partitioned shard would act as an independent blockchain with cross-shard transactions and a unified policy to create a common market of tokens. A successful realization would drastically increase the number of transactions handled per second since the normally linear blockchain would add a second vertical dimension. (Sharding FAQ, 2017)

4.7 Wrap-up

Ethereum's self-set objective to build a general-purpose platform for distributed state machines is the driver to redesign approaches form Bitcoin. Introducing account objects with contract code advances the complexity of the state to solve a variety of non-trivial problems. Key-value storages and a consumable unit for computational steps enable quasi Turing-completeness with loops. Finalizing valid state transitions continues to involve economic measures in the form of mining.

[23]"Majority" in PoW equals hashing power that can be used for other purposes, whereas majority in PoS means Ether tokens, thus are an internal part of the blockchain system.

5 Ripple Protocol

5.1 Introduction

In this section, we introduce the Ripple payment protocol based on blockchain technology and contrast it with Bitcoin and Ethereum. Having a backing central authority for responsibility and accountability, namely the Ripple company, we decide their influence and released data as the system's primary truth. Our structural and evaluating analysis builds upon their official developer documentation (Developer Center, 2017) as our technical source if we do not indicate it otherwise. Further, we examine their published source code (Ripple GitHub, 2017) and the initial whitepaper by (Schwartz, Youngs, & Britto, 2014).

5.2 Background and Application Purpose

The Ripple start-up existed several years before the first blockchain system, Bitcoin, arose in 2009. In 2012, still early days for blockchain systems, the company switched to blockchain technology as their database and network foundation by conceptualizing their individual Ripple protocol, a vastly distinctive approach to other blockchain systems.

The application purpose is to connect banks, payment providers, digital asset exchanges and corporates globally via their Ripple network to provide scalable and secure payments while reducing transaction cost and facilitating access. (Ripple Inc., 2017) In more technical words, the goal is to keep a peer-to-peer network running a blockchain with a low-latency consensus algorithm while still maintaining robustness in the face of Byzantine failures. (Schwartz, Youngs, & Britto, 2014)

5.3 The Ledger - *State*

5.3.1 An Account – *The Individual Ledger*

Ripple also makes use of an account-balance model just as Ethereum does, but the idea and structure diverge heavily. Similarly, a Ripple account object contains an outgoing transaction *counter* and a *balance*. In opposition, it directly stores the *address* as an attribute instead of the external mapping in Ethereum and it includes a *history* of transaction hashes that affected the account. These two attributes essentially define every account as a self-contained ledger, where it is possible to iterate through the history of a single account's states. Every account must always be controlled by a private key or similar construction[24], hence not allowing for code-based smart contracts. The reason for this is because Ripple does not pursue to be of general purpose, but rather a purely financial payment system mostly influenced by human actions. Below we see the fields of an account object how it is stored in a block.

[24] Multiple private keys or a passphrase.

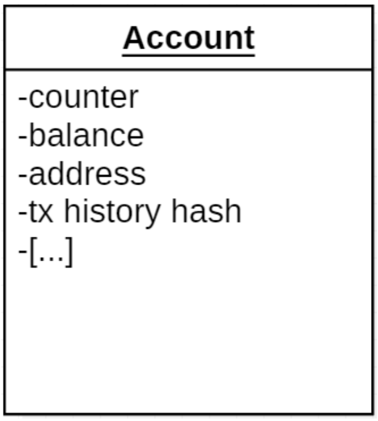


Figure 22 *A Ripple Account*[25]

Addresses

Address creation is almost identical to Bitcoin with ECDSA for private and public keys, then SHA-256 of RIPEMD160 and base58 encoding for addresses. SHA-512 half, which is 512-bytes hashing and truncating to the first 256 bytes is alternatively supported hashing, which is considered as secure as SHA-256, but slightly faster on 64-bit processors, which is a wanted characteristic for a payment system settling transactions.

5.3.2 A Block – *The World Ledger*

Alike Bitcoin and Ethereum, Ripple tracks all state transitions of each block by including a list of transactions in the data section of that block. Resembling Ethereum, it strictly saves the entire state - the sequence of account objects - in every single block indexed as a combination of radix and Merkle tree. The same data structure is also used for the transactions.

Header

We show the fields of a Ripple header in detail:

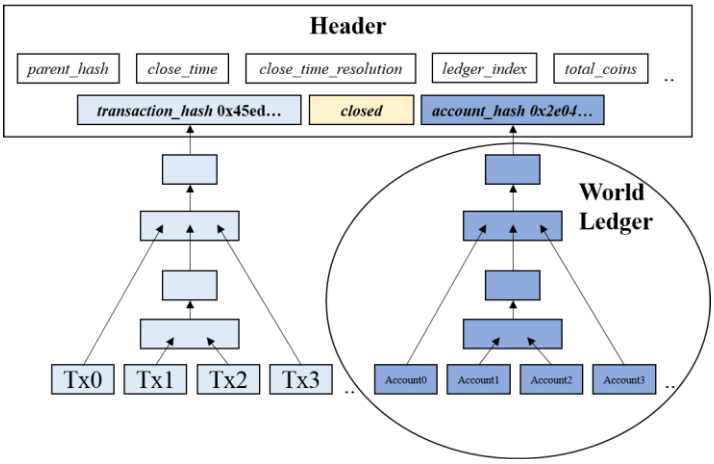


Figure 23 *Ripple Block Header*

[25] We oversimplified this UML object of a Ripple account and altered names on purpose to fit our analogy with Ethereum.

We observe the same block connecting mechanism by referencing the previous header hash (*parent_hash*), including a timestamp (*close_time* plus rounding specification in *close_time_resolution*) and the tree root hashes of the current state (*account_hash*) and transistions (*transaction_hash*). For security measures, a header tracks its own block height (*ledger_index*) and all tokens in circulation (*total_coins*).

The crucial distinction to Bitcoin and Ethereum lies in the absence of a nonce for Proof of Work mining, but instead relying on a Boolean flag (*closed*) indicating whether the block is still in the mining process or already finalized. After analyzing the Ripple consensus algorithm for mining new blocks and the unique network structure, we will fully understand why this simplification can work in practice.

5.4 Transaction System

5.4.1 XRP Tokens and Inflation

The native protocol token is called a Ripple (Symbol: *XRP*). In scalar values, it is always denominated as a quantity of Drops, the protocol's atomic unit (One XRP = 1,000,000 Drops). Within the Genesis block 100,000,000,000 XRP were spawn, whereby about two thirds of these are time-locked for the Ripple company's future development. Every transaction needs to destroy at least ten Drops as global fee to be valid. Furthermore, every ever-created account must lock a deposit balance of at least twenty XRP.[26] This reserve requirement encourages reusing an account and protects against spamming and Denial-of-Service attacks. It further de-anonymized the account steadily.

There will never be any additional XRP mined for circulation. This de facto makes the number of total tokens static or slightly deflationary in the long-term instead of the inflationary models for Bitcoins and Ether.

5.4.2 A Transaction

The Ripple protocol allows to issue thirteen pre-defined types of transactions. For now, we consider the standard type regarding XRP balances, a *payment* transaction. The only obligatory data fields are the sender's address, the receiver's address and of course, the token amount to be send. Formatted in JSON, the sender signs the full data to authorize the transaction to be relayed onto the ledger. This is the most basic transaction type possible for a payment system. Most other transaction types regard to trust lines.[27]

5.5 Consensus Algorithm – *State Transitions*

The Ripple Protocol consensus algorithm (*RPCA*) is fundamentally different from Bitcoin's or Ethereum's Proof of Work mining. According to the whitepaper by (Schwartz, Youngs, &

[26] An account gets created by sending XRP to an address for the first time.

[27] Cf. chapter 5.7.

Britto, 2014) it builds upon three goal pillars. First, *correctness*, which means it is impossible for a fraudulent transaction to be included in the ledger. Second, *agreement*, that says all honest nodes should reach consensus on the same state transition. And third, *utility*, which implies consensus will eventually terminate.

We transfer to our scheme from chapter 3.5 and see the objectives for distributed consensus are identical:

(1) ***Termination ≙ Utility***

The Ripple processes for persistent data storage are writing a block and reading a block.

(2) ***Validity + Integrity ≙ Correctness***

If a block is valid, it needs to be accepted as valid and added as part of the chain. All invalid blocks must be denied being accepted in the chain.

(3) ***Agreement***

There can never be more than one valid block referencing to the same previous block or in other words the chain must be linear without branches.

The RPCA is round-based. We specify,

- *valid transaction*: a transaction correctly signed by the sender and with a sending amount of XRP, such that the sender's account balance does not fall below twenty XRP - conflicting transactions are treated as FIFO
- *candidate set*: list including all collected valid transactions having a later timestamp than the latest valid block
- *validator* or *validating node*: a network node running the server software of the Ripple protocol to take part in consensus
- *unique node list (UNL)*: a validator's list of validators connected to him

and aggregate the process algorithm:

- *for each* consensus round (re-starting circa every ten seconds)
    - *each* validator compares his candidate set with the ones from his UNL via the transaction hashes (lasting approximately two seconds)
    - *if* a supermajority[28] X of validators from the UNL have the same candidate set
        - *then* finalize consensus by creating a block of all transactions in the candidate set and adding the block to the main chain - this implies a state transition by applying all transactions to calculate the world ledger and setting the closed flag in the block header to true before hashing

[28] Supermajority X refers to >50%, except if the last round was unsuccessful finding a block, then the threshold adjusts stepwise upwards until >80%.

- *else* delete valid transactions from candidate set having a lower than *X* support rate - this defers a transaction to the next round

Due to the tendency for correctness we could call the algorithm Proof of Correctness (*PoC*) borrowing the PoW term scheme. As long as connectivity between two UNL cliques[29] surpasses *(1-X)*n*, a fork between two cliques is prevented.[30] (Schwartz, Youngs, & Britto, 2014, p. 5) In practice the UNLs are not structured clique-wise, which hardens forking. More on the structure will follow in the network chapter.

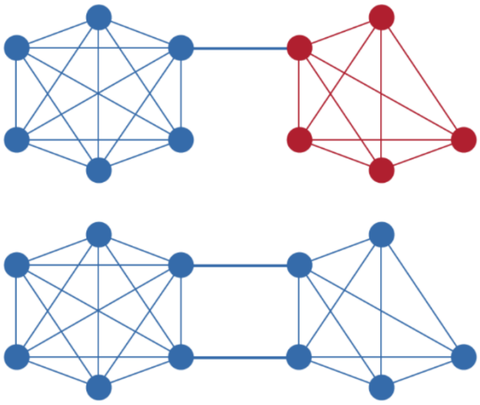


Figure 24 *An example of the Connectivity required to prevent a Fork between two UNL Cliques (Schwartz, Youngs, & Britto, 2014, p. 5)*

A problem case becomes clear, when we think about what happens without global consensus, which cannot be guaranteed. The consequence would be that the protocol falls into an infinity loop. Whereas, in the event of not having a global consensus, Bitcoin and Ethereum mining continues to work by tolerating to fork into several chains forming consensuses for each subset of network nodes ignoring each other. With the supermajority threshold peaking at 80%, the RPCA tolerates an upper limit of byzantine failures of $\frac{n-1}{5}$ in a network of n nodes, compared to the *<50%* in Bitcoin and Ethereum.

We embrace the algorithm and conclude

(1) ***Termination***

The RPCA decides for creating a block in finite time if a supermajority of validators in the network act according the source code rules and there are no independent cliques. Reading is trivial and done locally on each server on demand.

(2) ***Validity***

Transaction validations are one-time work done at each server and the candidate set equal for a supermajority becomes valid in a block.

(3) ***Integrity***

[29] A clique is a set of validating nodes k, that when combining all their UNLs form a k-connected graph with only at most one edge per clique-validator to validators outside the clique.

[30] N is the number of validating nodes in the total network.

Validators deny creating a block without supermajority of all its transactions. Permanent misbehaving leads to their exclusion from the UNLs of others.

(4) ***Agreement***

Again, validators only create a block with supermajority of all its transactions. Permanent misbehaving leads to their exclusion from the UNLs of others.

We rely on the constraint of having a supermajority of validating nodes, which would be an impossibility with the open unstructured peer-to-peer network of Bitcoin or Ethereum. Besides, reaching consensus is not directly incentivized with a mining reward. To comprehend how this works for Ripple to be deterministic, we need to take the peer-to-peer network into account.

5.6 Network

5.6.1 UNLs and Subnetworks

The last piece on why the RPCA succeeds without token reward incentives is inherent to the network setup of the Ripple nodes. The fact of having a unique node list (*UNL*) resembles a form of whitelisting your own IP-connections inside the peer-to-peer network. It is vitally different from the open and anonymous network of Bitcoin or Ethereum, but rather a permissioned model. This relocates trust between nodes form inside the protocol - as in Bitcoin or Ethereum - back to the outside real world. The operator of a validating node has already established a trustful relationship with another owner operating one or more servers of the former's UNL. The more other entities trust someone by having him in their UNLs, the higher impact that someone has on the network-wide consensus, which incentivizes him to honestly ensue the source code rules. This UNL mechanism effectively clusters collectively-trusted subnetworks within the larger network with the main cluster having the highest influence on consensus. (Schwartz, Youngs, & Britto, 2014)

A validator trusts other validators on its UNL not to collude against him, which gives each validator an urge to incorporate several operators with different interests in his UNL. But each added operator diffuses a validator's view on consensus and augments the risk of not reaching consensus - especially if he weights the votes of all operators on his UNL distinctively outside the system. This balances out the number of members in each UNL and aligns they each have an influencing role of similar weight. Therefore, the mechanism prevents larger cliques from evolving and levels out partial clusters running system-wide.

5.6.2 Gateway Servers vs. Clients

There are essentially two types of nodes in the Ripple network running their respective software. First, we have the server nodes, which all stock a local copy of the blockchain and follow the network. Some of them supplement their role with being a validator for consensus. The Ripple company started to operate the first validator and decided on a UNL consisting of banks, payment providers and corporate merchants, which induces the main cluster to continue to be solely of institutional nature with sparsely permitting new parties. This is in big contrast to the thousands of nodes in Bitcoin or Ethereum predominantly run by private individuals. The Ripple setup excludes the average Joe from maintaining the blockchain. To include these end users, servers may act as a *Gateway*. A Gateway is a middle layer bank-wise business bridging between the blockchain and the real world with its traditional financial system. They are usually restricted by a country's law to fulfil anti-money laundering and know your customer requirements, which permits almost no user anonymity albeit asymmetrical cryptography for addresses.

As an alternative to having a user account on a Gateway's website, an individual can run the second type of network nodes, a client node. However, it must always connect to Gateway servers and it is limited to only use the transaction system without participating in consensus or any blockchain administration. Even though it appears to resemble the SPV clients from Bitcoin or Ethereum, a Ripple client can only verify if a transaction is well-formed with valid signature, but never a block on its own. Since there is nothing like Proof of Work in a header hash, a server could easily counterfeit a mass of blocks tricking the connected client.

The trust dependency of client nodes to server nodes transforms the peer-to-peer network into a two-layered peer-to-peer network resembling a decentralized client-server architecture rather than the distributed network structure of Bitcoin or Ethereum. We render the two layouts:

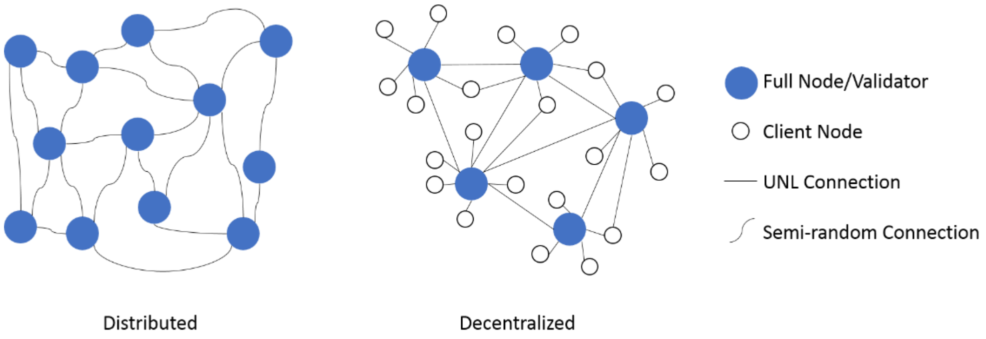


***Figure 25** Bitcoin and Ethereum's Network (left) vs. Ripple's two-layered Network (right)*

5.7 Enabling Issuance Tokens

5.7.1 Rippling Issuances over Trust Lines

We finally want to introduce a unique feature supporting applications, that are extremely important for the objective of Ripple. We keep this sub-chapter slim though, because the application type is not a direct implication of the blockchain data structure but rather of the custom network design with a system interface to the outside. Apart from the native token XRP, the bank-wise Gateways can issue assets themselves. These issuances, also called *IOUs* (short for "I owe you"), are not debit-based like the cryptocurrency XRP. Instead they are credits representing assets outside the Ripple ledger. These assets could be anything a Gateway bails a guarantee for, from fiat money such as Euro or US-Dollar to gold or stocks. If a Gateways for example allows USD-IOUs, a client or customer can send USD externally via the traditional financial system to the operator of the Gateway and get a USD issuance recorded on the blockchain as asset in his account. The raw transaction format, JSON, can easily adapt any text-based description of the asset to be signed afterwards.

Gateways acting as the entrance hubs for non-Ripple-ledger assets is part of the bigger picture displaying how one can transfer IOUs between accounts. Technically, a transfer of IOUs can just be a standard payment transaction having a nested balance of that asset, but for the recipient to redeem that credit from the issuing Gateway the blockchain protocol embodies a method called Rippling, hence the name Ripple. This is where most of the transaction types recorded in blocks are used for. Two accounts - for instance belonging to two befriended Gateways - can establish a *trust line* via a *TrustSet* transaction specifying the maximal amount of one or more IOUs they assure to credit each other in the future. We could name that an accounting relationship. Now, if there are unused units of a trust line between A and B plus available units of the same IOU of a trust line between B and C, then A can effectively send C IOUs not exceeding any of the two trust line limits. This *transitive trust* mechanism is automated via several transaction types altering the IOU balances of the three respective accounts. Having several trust lines with a variety of issuances between two entities allows for cross-asset transfers where XRP can always act as the universal currency bridge if needed. Since most Gateways know each other and their customers, there is a wide-spread and interlaced mesh of trust lines in existence. Therefore, sending IOUs over the network with multiple hops becomes simply a problem of finding a valid path in a graph. At first glance, this seems to degrade the meaningfulness of XRPs, but XRP supports the system with a liquidity role we state more precisely in the very next chapter. Additionally, trust lines with their issuances can get frozen by the issuer restricting transfers for security measures, which can never happen to XRP balances.

5.7.2 Interledger - Connecting Ledgers for Currency Exchange

The process of sending issuances of real world assets in transactions on the blockchain becomes worthless if the backing Gateways do not comply with the obligation to pay back the actual asset in return for an IOU token. May it because of unwillingness, bankruptcy or any arbitrary reason, there is no way of enforcing redemption on the data structure of the blockchain - because the assets are tangible without a de facto purely digital equivalent. To mitigate that risk, Ripple deploys a technology for connecting different ledgers in a standardized manner, the Interledger Protocol (*ILP*). We adapt the architecture model defined by (Interledger Architecture, 2017) on how it complements the Ripple blockchain:

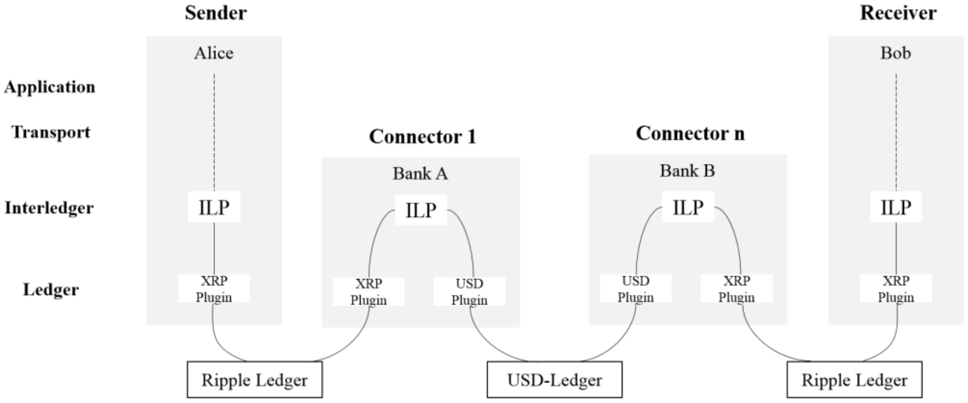


Figure 26 *XRP-USD-XRP Transaction with Interledger Protocol*

Focusing on the blockchain aspect, we do not care about the application and transport layer, where the parties would define payment conditions and other details on an end-to-end level. In our case Alice would send Bob 1,000 USD via the Ripple blockchain. Alice and Bob use a Ripple client dependent on their respective Gateways A and B, which happen to be Bank A for Alice and Bank B for Bob. Both banks settle USD payments over the traditional financial system with each other, thus acting as connectors. The ILP then acts as a middleware to settle all transactions at once with conditional payments. Analog to a two-phase commit, the ILP prepares the USD transaction and the three atomic transactions in Ripple, Alice to Gateway A to Gateway B to Bob.[31] If successfully prepared, it locks all accounts, applies all transactions and then unlocks the accounts again. The source code of ILP is thereby the coordinator to achieve data integrity. To allow faster settlement, Bank A would have pre-funded Bank B a collateral in USD as trust line counterpart, which means the money is already at its destination before the transaction was triggered. The ILP allows for multi-hop transactions even with currency conversion - usually inside the same entity’s two ledgers - on its way and uses collaterals at liquidity providers to enhance finding links. For Ripple, the default liquidity case converts to XRP, transfers XRP and then converts the money back.

5.8 Protocol Updating

The process of updating the Ripple source code in consensus resembles the model of BIPs and EIPs in the way that new source code is published and then participants vote on the proposed change. Unlike Bitcoin or Ethereum miners who express their vote by including data in a block they mine, Ripple validators vote by submitting pseudo-transactions including their vote as data. This model enables fast ad hoc decisions, currently cycled every *256th block*. This is impossible within Bitcoin or Ethereum since a transaction could come from anyone in the network and PoW hash votes in blocks require many more blocks to adequately represent a fair distribution on the overall opinion. The Ripple company leads developments with easier collaboration and discussions outside the protocol since they know the major players, the server nodes. This affords to manage updates more smoothly and efficiently, but also more centralized and biased.

5.9 Wrap-up

The Ripple blockchain protocol aims to found one connected global payment and settlement network. To accomplish that, it is highly interweaved with the current traditional payment processes of banks. The permission-based network structure shifts back trust to corporate stakeholders running Gateways. Cynically one could compare it to a multi-nepotism network. However, this layered structure facilitates transactions on the platform of any asset in the form of issuances with higher transaction throughput than Bitcoin or Ethereum. Not allowing for smart contracts and instead relying on a simple account and transaction format, tailors the network specifically for financial applications with high security requirements.

[31] The actual setup of Ripple uses multiple transaction, but for demonstration purpose we reduce them to two different transaction concepts.

6 Further Blockchain Concepts

6.1 Introduction

To gain a broader and all-embracing view of existing blockchain systems, we extend the horizon beyond our selected top three by shortly introducing examples fitting in the four categories mentioned in chapter 2.3.3. We omit a deeper technical study of them to keep our scope.

6.2 Cryptography

6.2.1 PoW Hashing Algorithms

We notice there is a variety of hashing algorithms used for calculating Proof-of-Work. Aside from the covered double SHA-256 in Bitcoin and the Ethash for Ethereum there are for instance *Scrypt* used in Litecoin, *X11* for DASH and *CryptoNight* developed by (Saberhagen, 2013) and later adapted by the more prevalent Monero. But why would a system elect one and not the other and what is their common ground?

As we have seen for Ethash, relying on functions with memory hardness to avoid the use of *ASICs* and benefit GPU mining satisfies a better distribution of hashing power to individuals. This reduces the risk of a few mining companies overtaking PoW consensus. Scrypt and CryptoNight have the same intent. CryptoNight's hashing function goes even further and achieves cache hardness by pegging a buffer-intense voting mechanism for the right order of transactions. This emphasizes latency and paves the way for CPU-based mining having a similar performance as GPU-based. (Saberhagen, 2013, pp. 11,12) However, the experience with Bitcoin's SHA-256 has shown it could only be a matter of time and cost to introduce better performing hardware like ASICs for every hashing algorithm. As a reference, Bitcoin's prominent mining hardware switched from being CPUs to GPUs in 2010 and to ASICs around 2012. (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016, pp. 137-145) The X11 approach of DASH is not primarily to harden ASIC mining, but to have a higher level of security if a sudden breakthrough would threaten one specific hashing function like SHA-256. For it, X11 connects eleven different hashing functions in series. (DASH X11, 2017) Because the hashing algorithms are all used exclusively to calculate block header hashes in a certain range, a system - if unanimous - can easily change the source code rules to adapt a new algorithm leaving the old blocks unaffected.

Apart from being deterministic, cheap to verify and distributed with high min-entropy, the common desired properties for all PoW hash functions *H(x)* are:

- **Hiding**

  Given *H(x)*, it is infeasible to find *x*.

- **Puzzle-friendliness**

Given *k* and *y*, such that *H(k || x) = y*, it is infeasible to deduct *x*.[32]

- **Collision-resistance**

  Given *H(x)=H(y)* and *x*, it is infeasible to deduct y for *y != x*.

(Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016, pp. 23-31)

Puzzle-friendliness restricts the best performing solving strategies for mining to brute-forcing and therefore is essential for PoW. Relating to the block connecting mechanism, hiding and especially collision-resistance resolve the issue of illicit copies for a found hash. That is, if a miner found and broadcasted a valid block, others should not be able to use that information to compute another input - with their coinbase transaction for example - resulting in the same valid hash.

6.2.2 Digital Signature Schemes

Hash functions are also used within the digital signature algorithms of blockchain systems. If a system wants to change its signature scheme it would be much harder than altering the PoW function. This is due to the service as a pseudo-identity and access to the system. Updating the source code rules would not suffice, but rather every single existing address would need a new counterpart. Since private keys for addresses should be secret, every user owning a private key would need to manually initiate an update process for his key-address pair.

As of 2013 the National Institute of Standard and Technology (*NIST*) assures the discrete logarithm problem for *ECDSA* and the prime factorization problem for *RSA* as secure standards to deploy for digital singing. (Information Technology Laboratory, 2013) Further, *Lamport* signatures could retain security in the event of a breakthrough in quantum computers. Most current blockchain systems still rely on ECDSA with diverging inputs as we have mentioned for Bitcoin and Ethereum. However, a security hole detected in ECDSA even in the long-term would highly derogate blockchain systems in their functionality as a transaction system if they rely on a hard-coded rule for signing.

6.3 Distributed Consensus

A blockchain as a distributed database crucially needs a reliable consensus algorithm to achieve data integrity. We already know about PoW, a planned PoS in Ethereum and the RPCA. There are numerous approaches implemented or under research. We present a selection in table, which we briefly define and categorize in two major groups. One where the decision-making resource sprouts from outside the blockchain system, the other from within. This classification covers all possible approaches since a consensus algorithm is nothing but a decision function, hence cannot have no input resource.

[32] K should be random, i.e. chosen from a distribution with high min-entropy.

Consensus Approach	Basic Description	Example Implementations
External Resource		
1. Proof of Work (*PoW*)	Perform a costly computational task to verify expenses.	Bitcoin, Litecoin, Ethereum, Zcash, Monero
2. Proof of Storage Capacity (*PoSC*)	Provide hardware storage to verify expenses.	Burst, StorJ
3. Proof of Activity (*PoA*)	Provide a permanently participating network node to verify expenses.	
4. Proof of Correctness (*PoC*)	Aggregate supermajority of a known set of network nodes to verify agreement.[33]	Ripple
Internal Resource		
5. Proof of Stake (*PoS*)	Have a significant monetary interest in the ongoing of the system…	
A. Random Selection	…via lucky draw with probability of own tokens divided by total tokens.	Nxt
B. Coin Age	…via lucky draw following a probabilistic function with two inputs, an address's token amount and timed transaction inactivity.	Peercoin
C. Delegated Voting	…via lucky draw from a sub-set elected by the sum of tokens assigned from other addresses.	Bitshares, Steem, NEO

(Poelstra, 2015) argues, pure Proof of Stake concepts with mining rewards have one shortcoming, namely that they depend on the very history they are aiming to form. Therefore, they should better be combined in hybrid systems with an external resource to some extent.

All presented consensus algorithms have a mutual objective principle. The block creation time is Poisson distributed, because every atomic resource r ought to have an equal probability of $\frac{r}{sum(r)}$ of deciding consensus, meaning a Bernoulli trial. However, pooled mining or unequal hybrid systems deviate from the distribution model and constitute attack vectors for the decision process.

6.4 Interfaces and Access

Data Interfaces - Oracles

Blockchain database relies on every node performing changes on their replica independently from each other. Therefore, they cannot poll data via APIs of centralized servers because a multitude of individual requests would not guarantee a deterministic outcome. A working solution to use external state is if the external source itself submits data into the blocks. Smart contract or transaction scripts exhibit entry points for arbitrary data. Actively pushing the data by an external agent solves the deterministic restriction, but is prone to errors or manipulation

[33] The UNLs are the resource, which operators dictate from system-external decisions.

by that entity. Hence, consensus-based oracles using multiple servers are an advancement. Further, financial prediction markets where humans bet on the outcome of an event and market economics determines the price and therefore the correct data are an alternative in research. Augur and Gnosis are two leading models based on Ethereum. The infused data then persists and allows for conditional checks to trigger events.

Access and Identity Interfaces

The second interface we discuss is not based on data communication, but rather on identity communication to access a transaction-based blockchain system from the outside. Digital signatures schemes provide address anonymity. But, when transactions are publicly and permanently visible on a blockchain, they allow to transitively link them together. This denudes anonymity to pseudonymity. To preserve real anonymity, (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016, pp. 165-169) define three domains of desirable *unlinkability*:

- **Addresses**

  Hard to link different addresses of the same user

- **Transactions**

  Hard to link different transactions of the same user

- **Payments**

  Hard to link sender and recipient of a payment

One existing solution to all three domains is coin mixing used by external Bitcoin services or DASH. The approach is to break transaction outputs into standardized pieces and re-aggregate them together with arbitrary pieces of other transactions in joint transactions. To further obfuscate a clustering of addresses and transactions, this shuffling process repeats. (Duffield & Diaz, 2017, pp. 5-8)

The main issues are performance, since higher intransparency results from incrementing the transaction amount and repeating cycle usually with using some middle-men service. To avoid that, the Zerocoin approach mixes on protocol level by minting a base unit and then pouring tokens back by destroying the mint. Going even further, the Zerocash design implemented in Zcash relies on *zk-SNARKs*, cryptographic zero-knowledge proofs, to only proof a commitment to virtual minting and destruction, thus avoiding the base unit transactions. (Ben-Sasson, et al., 2014)

Monero assimilates the entire blockchain transaction system to assure privacy. A transaction output is not recorded in blocks, but rather a unique cryptographically derived key image of it masking the actual output for miners, but ensuring to prevent a double spend. (Saberhagen, 2013, pp. 4-11) Ring signatures, where the signed output is mingled with arbitrary past output signatures as decoys, prevent attaching sender and recipient to a transaction. Also enforcing one-time stealth addresses hides the recipient and complicates clustering of addresses. Albeit solving the linkability domains fully, Monero is not a perfect solution either because the cryptographic transversions consume higher storage too. (Courtois, 2016)

We assess that the problem of linkability is effectively solved by the chosen methods, but with the drawbacks of performance and additional data. Besides the anonymity on database level, de-anonymization condenses on network-layer with IP sniffing or other attacks. Integrating a TOR-like network structure would be an effective solution.

6.5 Network Structure

Bitcoin and Ethereum use purely equipotent and anonymous peer-to-peer networks, whereas Ripple nodes enter the network and influence consensus only on exclusive permission of identified peers, thus layering the network. Another multi-tiered construct is the network of DASH, where a network alike Bitcoin's gets expanded with a Masternode role. These Masternodes require to bound a token collateral (=*PoS*) and provide services to the network including mixing transactions, faster verifying transactions between each other and voting on how to use an associated mining budget. There incentive for operating is an allocation of the mining reward. (Duffield & Diaz, 2017, pp. 1-5)

More centralized and permissioned networks enable more features and maximize scalability for a blockchain, but always bear the risk of exploitation by the controlling entities. The Hyperledger Fabric project as an extreme sample offers a fully private and centralized blockchain version to run.

7 High-Level View and Design Space

7.1 Introduction

In this chapter, we consolidate our findings from the three individual blockchain analyses to construct a uniform high-level view of blockchain systems. We omit considering private blockchain systems on a centralized server or other yet to be implemented concepts due to the lack of established prototypes.

7.2 Crucial Components

We identified four distinct domains to categorize blockchain elements: state, transition, network and access and interface. We reason why components are *crucial* for a viable blockchain, but first include common terms we consider *partly crucial* or *not-cruc*ial to clear misunderstandings.

Partly and not crucial Components

Component	Reason and Purpose
Partly Crucial	
Native Token	A native token may seem logical to exert a transaction system. Ripple’s IOUs of system-foreign currencies raise the inquiry how it would behave without the role of a native token and instead implement spam prevention and liquidity otherwise.
Not Crucial	
PoW Mining	Just a sub-form of a consensus algorithm. Ripple is an established counter-example without it. Even a race condition is not needed.
Token Rewards and Inflation	Ripple relinquishes both, rewards and inflation via coinbase transactions.
Account-balance System	Bitcoin’s UTXOs work without.
Unique Scripting Language	Ripple is specialized as a pure payment system without smart contracts. Hence, it does not require a unique scripting language.

Crucial Components

Component	Reason and Purpose
State	
Hashing	Hash pointers with their respective hash functions are the essential mechanism to connect block headers into the logical data structure of a blockchain. Hash values further allow to efficiently exchange information about the state between nodes to synchronize data thereafter.
Start State & Genesis Block	A unique Genesis block with a start state is relevant for a blockchain's identity. In case of a fork, the forking block supports the role of a Genesis block.
State Replication	Data replication on different places keeps an auditable view on the state. PoW-based hashes are hard to forge, though.
Transition	
Transaction System	A transaction system is the first and viable application of every blockchain. The usefulness of others without are yet to prove.
Consensus Algorithm	A distributed consensus algorithm is the premise to write valid data to the persistent storage following a deterministic model.
Hash Trees	Hash trees (Merkle, Radix) are needed to space-efficiently combine transactions into a block by including the tree root in the block header.
Network	
Peer-to-Peer Structure	Peer-to-peer nodes are a foundation to replicate the state data in a distributed fair manner, either with an unstructured or layered Design.
Entrance Point	For a new node to participate in the network, it needs to have an entrance point by knowing at least one IP address of an active node.
Access and Interface	
Protocol Rule Set	To communicate in the network and maintain state and transitions, participants need to agree on a set of protocol rules. They do not have to be open source code or uniform yet compatible with each other.
Protocol Update Mechanism	Breaks of the cryptographic fundamentals (Hashing, Digital Signing) may occur eventually. To customize the system and prevent failures, the system needs to have a mechanism to update its source code rules.
Digital Signature Scheme	Asymmetrical cryptography for digital signing allows to include identity via addresses for outside users.

7.3 Architectural Ontology Model

We consolidate the crucial entities and their relationships into a high-level architectural diagram to overview an entire blockchain system in an abstract manner.

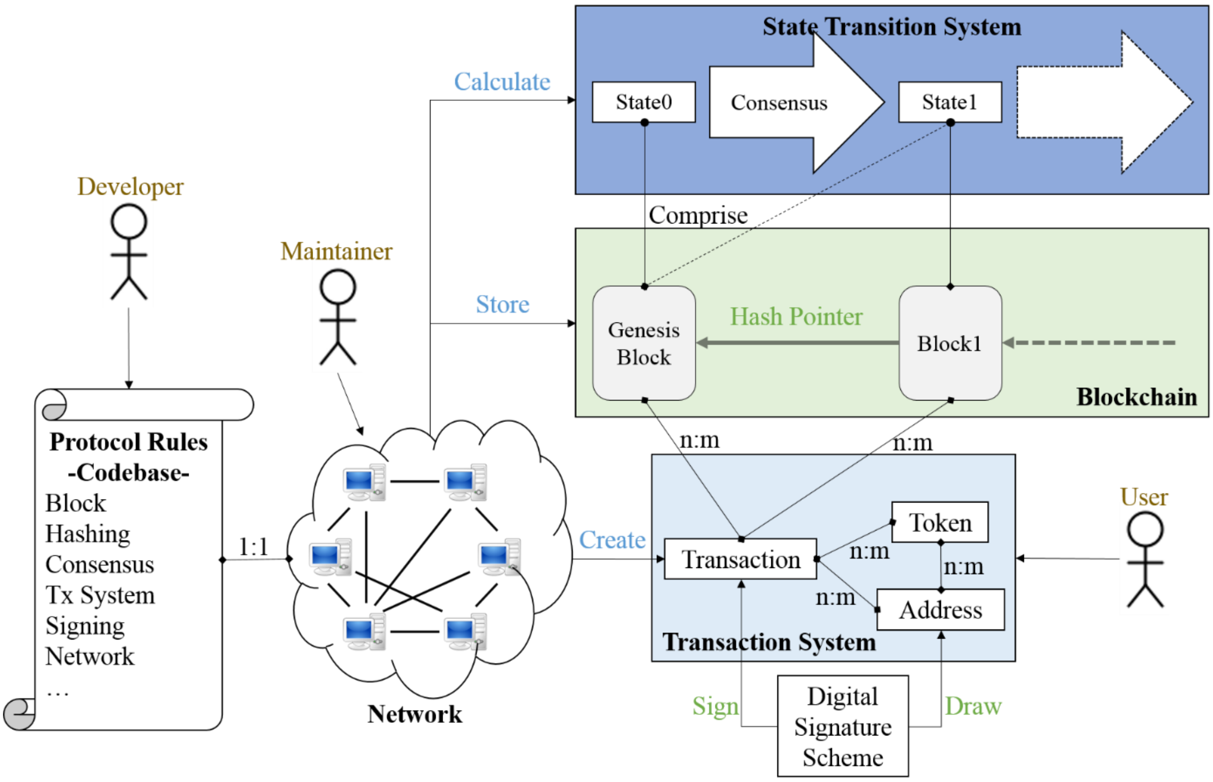


Figure 27 *Generic Blockchain System*

We identify three human roles. Developers who compose and update the codebase, maintainers run the system and users access it. All roles are stakeholders and influence each other. Lastly, we want to clarify, that state and block are two different things. In Bitcoin for example the UTXO state is calculated from all blocks in the past together even though nodes usually just store the set of UTXOs separately to not re-calculate it every block.

7.4 Morphological Analysis

We conduct a morphological analysis to reduce complexity when exploring possible variants of a blockchain system and its characteristics. Step one is the definition of attributes a blockchain system adopts. Step two is the categorization and the array in a morphological box with possible parameters of the attributes. In step three we map our three established blockchains to the scheme. This process should provide a clear guidance opportunity to classify all existing blockchain solutions.

7.4.1 Attribute Definitions

[1] Hashing Algorithm

The algorithm used to connect two subsequent block headers.

[2] Start State

Implies whether the blockchain's first block is uniquely created leading to a full state or a shared block with some other blockchain system, thus forked. A fork of Bitcoin for example inherits the full state with the same history as Bitcoin itself even before the fork.

[3] State Replication

A blockchain obligates to replicate state data.

[4] Smart Contracts

The fact of supporting any smart contracts and distinguishing between complexity of Turing-completeness or not.

[5] Transaction System

A blockchain obliges to use a transaction system for state transitions in our categorization.

[6] Native Token

Having a native token without reference to outside the system is partly crucial. Nonetheless we assume specific blockchain systems could work without and refer to our analysis about issuances in Ripple.

[7] Issuances

A blockchain allows transferring other tokens than the native token.

[8] Consensus Algorithm

The algorithm used to validate new blocks and add them to the chain.

[9] Meta data structure

The structural assembly of data in the blocks. Data in our space refers mainly to transactions.

[10] Network Admission

Participation freedom in the underlying network.

[11] Network Type

The communication type of the network. If there is no communication we have a centralized blockchain.

[12] Network Model

The allocation of roles for nodes in the network.

[13] Network Structure

The assembly of node connections in the network.

[14] Codebase

The publicity of the source code rules.

[15] Unique Scripting Language

Support for a unique scripting language when deploying smart contracts.

[16] Digital Signature Scheme

The mathematical foundations for signing a message or transaction.

[17] Ownership Model

The way to guarantee token access and redemption.

[18] Data Transparency

The outside visibility of data in the blocks without specific access rights.

We induced crucial components such as a transaction system and state replication from the last chapter to be obligatory selected. We allude to many more attributes a blockchain system can have we did not cover. Either because they are specific or not related to the technology, such as having a supporting company or built applications. Due to the novelty and pluralism of the technology we continually observe new systems testing further design attributes.

7.4.2 Morphological Box with Parameters[34]

#	Attribute	Possible Parameters						
State								
1	Hashing Algorithm	SHA-256 (double)	Ethash	SHA-512 (half)	Scrypt	X11	Crypto-Night	
2	Start State	New Genesis Block	Forked					
3	Replication	Yes						
4	Smart Contracts	Turing-Complete	Stack-Based	None				
Transitions								
5	Transaction System	Yes						
6	Native Token	Yes			No			
7		Inflationary	Static	Deflationary				
8	Issuances	Yes	No					
9	Consensus Algorithm	PoW	PoC	PoSC	PoA	PoS		
10						Ran dom	Coin Age	Dele gated
11	Meta data structure	Merkle-Hash-Tree	Radix-Tree	Merkle-Patricia-Tree				
Network								
12	Admission	Public	Permissioned					
13	Type	Peer-to-peer	No Network					
14	Model	Unilayer	Mulitlayer	Single Server				
15	Structure	Unstructured	Structured	Client-Server				
Access and Interface								
16	Codebase	Open Source	Closed Source					
17	Unique Scripting Language	Yes	No					
18	Digital Signature Scheme	ECDSA-based	RSA-based					
19	Ownership Model	Transaction-based	Account-based					
20	Transparency	Full	Some Hidden Data					

[34] We introduced all abbreviations in this thesis. Cf. list of abbreviations if needed.

We propose how frequently cited loose attribute terms for a blockchain fit into our model. They are a multi-dimensional combination of individual parameters.

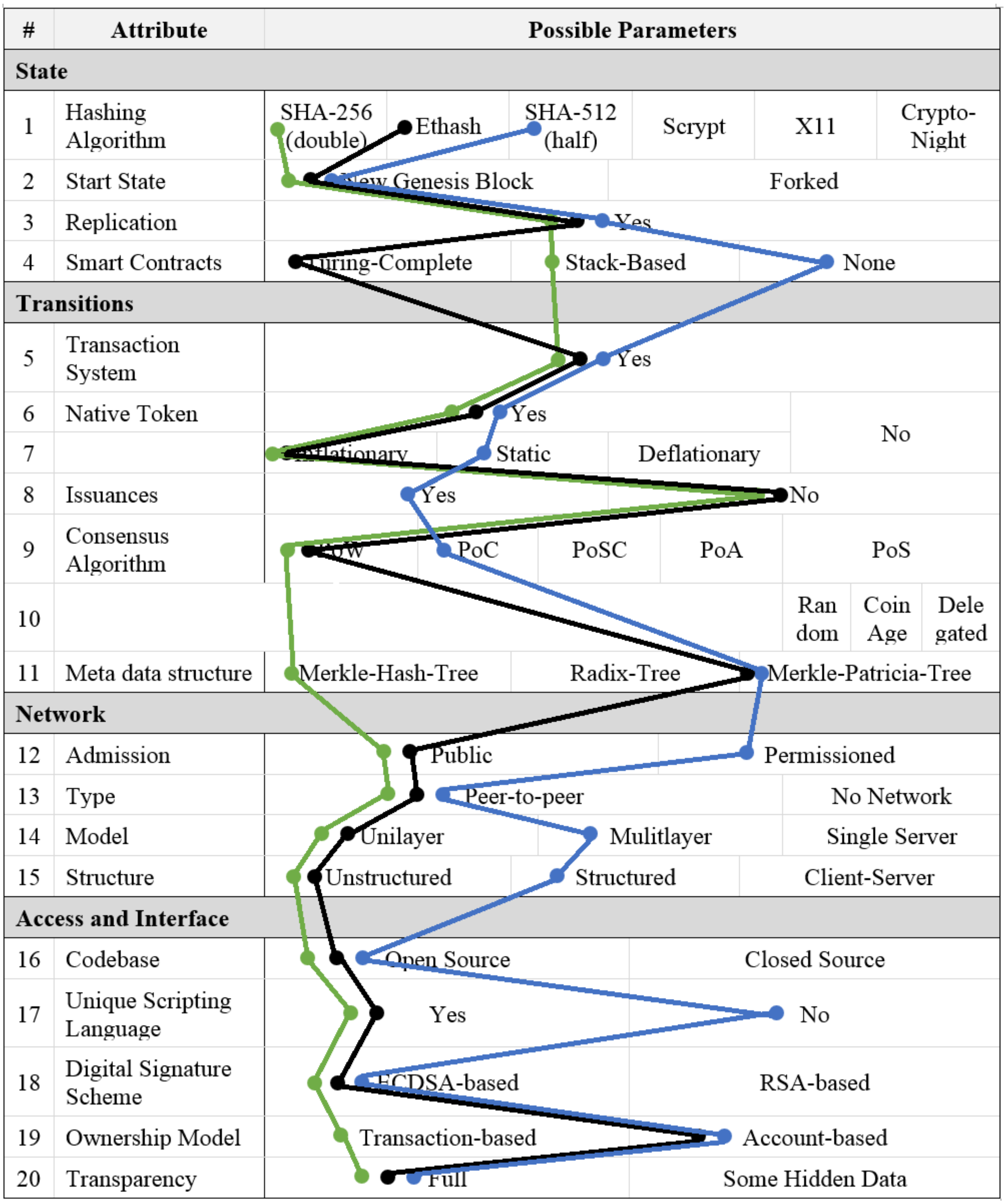
Is this blockchain system …

…distributed?	peer-to-peer (*13*) ∧ unilayer (*14*) ∧ unstructured (*15*)
…private?	permissioned (*12*) ∧ no network (*13*) ∧ single server (*14*) ∧ client-server (*15*)
…transparent?	full (*20*)
…(pseudo-)anonymous?	[*18*] signing ∧ [*20*] transparency
…trustless?	[*1*] hashing ∧ [*3*] replication ∧ [*9*] consensus ∧ public (*12*) ∧ peer-to-peer (*13*) ∧ open source (*16*)
…irreversible/immutable?	[*1*] hashing ∧ [*3*] replication ∧ [*9*] consensus ∧ peer-to-peer (*13*)
…energy costly?	[*1*] hashing ∧ PoW (*9*)
…autonomous?	Turing-complete (*4*) ∧ transaction (*5*) ∧ token (*6*) ∧ [*9*] consensus ∧ peer-to-peer (*13*) ∧ scripts (*17*)

Even broader terms like governance model or accountability would inherit more complex combinations also explicitly excluding parameters. We relate the privacy algorithms from chapter 6.4 in Monero, Zcash or DASH to [*18*] signing ∧ some hidden data (*20*).

[..] Attribute reference (any parameter)   (..) Parameter reference (specific)

7.4.3 Classifying Bitcoin, Ethereum and Ripple



#	Attribute	Possible Parameters						
State								
1	Hashing Algorithm	SHA-256 (double)	Ethash	SHA-512 (half)	Scrypt	X11	Crypto-Night	
2	Start State	New Genesis Block			Forked			
3	Replication	Yes						
4	Smart Contracts	Turing-Complete	Stack-Based	None				
Transitions								
5	Transaction System	Yes						
6	Native Token	Yes			No			
7		Inflationary	Static	Deflationary				
8	Issuances	Yes	No					
9	Consensus Algorithm	PoW	PoC	PoSC	PoA	PoS		
10						Random	Coin Age	Delegated
11	Meta data structure	Merkle-Hash-Tree	Radix-Tree	Merkle-Patricia-Tree				
Network								
12	Admission	Public	Permissioned					
13	Type	Peer-to-peer	No Network					
14	Model	Unilayer	Mulitlayer	Single Server				
15	Structure	Unstructured	Structured	Client-Server				
Access and Interface								
16	Codebase	Open Source	Closed Source					
17	Unique Scripting Language	Yes	No					
18	Digital Signature Scheme	ECDSA-based	RSA-based					
19	Ownership Model	Transaction-based	Account-based					
20	Transparency	Full	Some Hidden Data					

Figure 28 Design Classification Bitcoin, Ethereum and Ripple

● Bitcoin ● Ethereum ● Ripple

7.5 Model Alignment

To classify blockchain systems regarding their capabilities as databases and as distributed systems, we test the technology against typical metrics from standard models of both areas.

7.5.1 Databases

CRUD

On an operational level create, read, update and delete are the four basic functions of persistent storage in a database. We map blockchains to that pattern:

- ***Create*** A user creates a transaction by signing. Nodes create a block by consensus, e.g. mining. A transaction becomes persistent only if added to a valid block.
- ***Read*** Every node tries to synchronize as swift as possible to maintain a full replica. It then can read blocks and transactions at will.
- ***Update*** Updating is very expensive. It means adding new valid blocks to a block that already has one or more successors and out-racing the other previously created chain. This is a consensus or mining-fork and not supposed to happen.
- ***Delete*** Neither a single signed transaction, nor a block with its enclosed transactions can effectively be deleted after their broadcast. Indirectly a stale block can at most get ignored. Single transactions get ignored if a competing transaction persists in a block.

The fact, that updating and deleting values is by design extremely costly in a blockchain, renders a blockchain to be a create and read-only databases.

Data Integrity Model

Data integrity as the maintenance of data to assure its accuracy and consistency, is a critical database choice to pursue. They are loosely defined terms, but blockchains generally try to enforce both. Data consistency in blockchains refers to checking the intrinsic validity of transactions and blocks. In contrast, data accuracy connotes the system-validity of a block reached by consensus.

ACID

The ACID model is a set of properties to test whether transactions in a database satisfy them. The term transaction in this case is not the transaction of a payment system in a blockchain, instead it refers to a bundle of operations to perform in the database, such as a validity-check or adding a block to storage.

- ***Atomicity*** All operations of a database transactions will complete or none. A block can only be added if all block’s operations are verified and all hashes match. Otherwise nodes would deny it. The same refers to a payment transaction since they persist exclusively in a block. Therefore, database transactions of a blockchain comply to atomicity.

- ***Consistency*** Uniform validity checks according the protocol rules achieve data consistency. They prohibit inconsistent states changes.

- ***Isolation*** It means concurrent operations will lead to the same result as if they were performed isolated. Blockchains do not fully correspond to that scheme, since two competing payment transactions invalidate each other if they get validated in a block first. A blockchain guarantees a sequential model for concurrent operations.

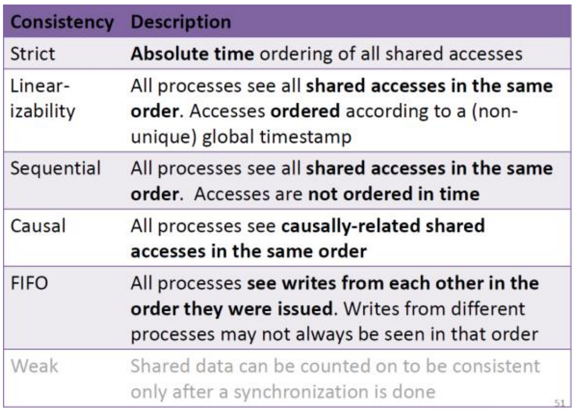
- ***Durability*** Once a block commits to the main chain, it remains there. Forks and consensus attacks are an exception leading effectively to a tamper-resistant instead of durable scheme.

It is difficult to fully project a blockchain database to the ACID model and we do not consider it suitable. The main reason is because in a typical database a central administrator conducts all tasks and participant trusts it to comply. Blockchain systems however do never guarantee any of the four properties, but rather convince participants with high probability to rely on economic incentives.

7.5.2 Distributed Systems

Consistency Models

Data replication on distributed peer-to-peer networks inherit a problem of consistency. This is system wide consistency, not data consistency as used in the last chapter. A consistency model is a contract between participants on the result of read and write operations in the presence of concurrency. We evaluate five consistency types of distributed systems following the scheme by (Jacobsen, 2015):



Consistency	Description
Strict	**Absolute time** ordering of all shared accesses
Linear-izability	All processes see all **shared accesses in the same order**. Accesses **ordered** according to a (non-unique) global timestamp
Sequential	All processes see all **shared accesses in the same order**. Accesses are **not ordered in time**
Causal	All processes see **causally-related shared accesses in the same order**
FIFO	All processes **see writes from each other in the order they were issued**. Writes from different processes may not always be seen in that order
Weak	Shared data can be counted on to be consistent only after a synchronization is done

Figure 29 *Summary of Consistency Models (Jacobsen, 2015)*

In the case of Bitcoin and Ethereum, two nodes can read a block at height X at concurrent time appearing valid to them. These could be two competing blocks with one of them becoming invalid after further consensus. Therefore, Bitcoin and Ethereum do not guarantee *strict consistency, linearizability, sequential* nor *casual consistency*. Similar with transactions where one single nodes could broadcast two competing transactions arriving at different nodes concurrently. One transaction would get included in the making of a block at node A and the other in the making of a block at node B. Hence, they do not ensure *FIFO consistency* either.

Bitcoin and Ethereum do not fit in a system-wide consistency model. They support a client-centric model, where consistency is an illusion at every node and eventually confirms under probability assumptions or if no updates were made. Both blockchains seem to fall into a weaker category of *eventual consistency* only supporting a liveness guarantee without safety guarantees. Therefore, we want to test the trade-offs against the BASE model and the CAP theorem.

BASE

The BASE semantics are what an eventual consistent distributed system provides.

- ***Basically Available***

  An interval between two blocks averages at the target time with permanent nodes storing state and history. Fulfilling both requirements a blockchain system indeed appears to work most of the time.

- ***Soft State***

  Different node replicas are not mutually write-consistent all the time. A blockchain database does not have a guarantee on what the current state is, thus called a soft state.

- ***Eventual Consistency***

  At some later point, high consistency is reached by consensus about states in the past. Blockchain databases follow eventual consistency.

A blockchain system fits into the BASE criteria, but these are much looser with less strict assurances than ACID. The BASE model acts as lower bound assumptions.

CAP Theorem

The CAP theorem of distributed Systems in general states, that a distributed data store cannot simultaneously provide:

- ***Consistency*** At the same time, all nodes read the same data.
- ***Availability*** Every request receives a response about its success.
- ***Partition-Tolerance*** The system continues to operate despite arbitrary partitioning of the network due to failures.

(Gilbert & Lynch, 2002)

Which guarantee do blockchain systems sacrifice? We already debated Bitcoin and Ethereum being eventual consistent, a very weak form of consistency. Cases like the rollback decision for Ethereum leading to Ethereum Classic after the DAO hack substantiate the loose consistency. However, that only applies for the most current blocks and nodes practically consider older data as finalized - with high network latency as the price to pay. Using the Proof of Correctness consensus approach with supermajority, Ripple performs much better in *consistency*. In general, one property of transactions supports consistency: A transaction is idempotent, meaning applying the same transaction twice in a row has the exact same result as applying it once. Bitcoin restricts to this property via the unique mapping of output-input scripts, Ethereum and Ripple with an increasing transaction counter. Sacrificing system *availability* would mean, that a blockchain system somehow pauses when it cannot commit transactions or blocks. For Bitcoin and Ethereum that is not the case. Ripple on the other hand, pauses and repeats a consensus round when node's candidate sets dissent to a specified degree. Abdicating *partition-tolerance*, a blockchain would only run if the entire network reliably communicates. Lacking a master administrator, a node is not aware of the network size. In Bitcoin and Ethereum, one way of still detecting if nodes encapsulated from the main network, thus being a partition, is to check for a sudden increase in time between blocks. However, a blockchain system is just partition tolerant to a certain extent where a larger size of the partition relative to the biggest partition and a longer duration of exclusion indicate partition intolerance. Image China would block all internet packages relating to Bitcoin for a year. This could permanently split the network in two versions even after reunion.

Blockchains handle the CAP trilemma situation in diverse ways, where a delayed consistency suffices Bitcoin and Ethereum and Ripple bears a risk of losing availability. Losing partition tolerance is no severe threat because of the competence to detect partitions. We conclude, that the design choices of blockchain systems carry their distinct top priorities and may drift apart.

8 Conclusion

8.1 Wrap-up of Findings

We want to condense our insights by resuming to our research questions from the beginning. We wrap-up the key findings separately.

RQ1 **Which are established Blockchain Systems?**

We isolated and defined the very basic terms existing in the blockchain sphere. We used a combination of six quantitative and two qualitative criterions and obtained a clear leadership role of *Bitcoin* with two emerging systems for alternative applications, *Ethereum* and *Ripple*. Several concepts from less important systems completed our research foundation.

RQ2 and RQ3 **What is the respective Setup of established Blockchain Systems?**

How do established Blockchain Systems differ?

We structured our insights along the model of a distributed deterministic state machine.

The systems' *state* data are different: Bitcoin sets on UTXOs, Ethereum maps balance and key-value-based storage of accounts to addresses and Ripple's world ledger is the concatenation of single account ledgers. To obliviate history states, Ethereum and Ripple both store the full state in each block, Bitcoin does not. As necessary for the definition of a blockchain, all three systems connect their block headers via hash pointers, although with varying functions.

State *transitions* are in all three systems triggered by transactions of native tokens with Ripple additionally supporting issuances as tokens. Ripple restricts smart contract creation for simplicity, security and scalability, whereas the other two implement a unique scripting language pursuing automatization. Bitcoin with a focus on simple stack-based scripts and Ethereum with a Turing-complete design for general purpose DAPPs. All three systems reach distributed consensus. Bitcoin and Ethereum with an economically incentivized PoW mining for higher byzantine fault tolerance and Ripple with an individual algorithm opting for correctness.

A peer-to-peer *network* underlies the blockchains with data replication on every full node. Bitcoin and Ethereum have an unstructured equipotent approach with a model of semi-random connections and communication. They currently struggle with latency and therefore scalability of their data. Ripple runs a two-layered permissioned network with selected corporates backing the top layer where consensus is agreed upon.

A pseudo-anonymous digital signature scheme is the essential way of participating in the transaction system. Ripple's Gateways weaken anonymous *access* to potentially control or restrict unwanted parties.

RQ4 and RQ5 **What are crucial Components and Characteristics of all established Blockchain Systems?**

How can a Design Space of Blockchain Systems be defined?

We identified eleven crucial parts for a blockchain system to be viable and depicted their relationships and roles. Evaluating the achievements of blockchain technology we aligned them with models for *database* and *distributed systems* including CRUD, ACID, BASE and the CAP theorem.

We embedded the common ground of all established blockchains systems in a morphological analysis and classified them accordingly. Attributes with their possible parameters was our proposed frame we defined beforehand.

8.2 Outlook

To date Bitcoin and other blockchains keep their promise to be a tamper-resistant database verified by a distributed timestamping server powered by a consensus of peers. The remaining questions are whether they can overcome their hurdles to mainstream adoption and serve as a protocol platform for future applications. The most problematic technical issue for that is the dilemma of scalability in a distributed environment without falling into a tendency of centralization. Furthermore, the outside ecosystem of each blockchain system plays a major role to drive development and build a community from technology-interested users to application-interested users. The right influence of all stakeholders should also be balanced out to avoid a collapse. All in all, the technology has a potential to thrive, but perhaps in another direction researchers think of nowadays.

Bibliography

amazon.com. (2017). Alexa Rank. Retrieved 07 08, 2017, from https://www.alexa.com/siteinfo

amazon.com. (2017). Alexa Rank. Retrieved 10 08, 2017, from https://www.alexa.com/siteinfo

Apostolaki, M., Zohar, A., & Vanbever, L. (2017). *Hijacking Bitcoin: Routing Attacks on Cryptocurrencies.* IEEE Symposium on Security and Privacy 2017.

Atzei, N., Bartoletti, M., & Cimoli, T. (2016). *A survey of attacks on Ethereum smart contracts.*

Back, A. (2002). *Hashcash - A Denial of Service Counter-Measure.*

Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. (2014). *Zerocash: Decentralized Anonymous Payments from Bitcoin.* 2014 IEEE Symposium on Security and Privacy.

Bertoni, G., Daemen, J., Peeters, M., van Assche, G., & van Keer, R. (2012). *Keccak implementation overview.*

Bitcoin - Developer Documentation. (2017). Retrieved 11 10, 2017, from https://bitcoin.org/en/developer-documentation

Bitcoin Core. (2017). Retrieved 11 10, 2017, from https://github.com/bitcoin/bitcoin

Bitcoin Technical Wiki. (2011). Bitcoin Transaction. Retrieved 11 10, 2017, from https://en.bitcoin.it/wiki/File:TxBinaryMap.png

Bitcoin Technical Wiki. (2017). Retrieved 11 10, 2017, from https://en.bitcoin.it/wiki/Main_Page

bitcoinstats.com. (2017). Data Propagation. Retrieved 11 10, 2017, from http://bitcoinstats.com/network/propagation/

bitinfocharts.com. (2017). Retrieved 10 08, 2017, from https://bitinfocharts.com/index_v.html

bitinfocharts.com. (2017). Retrieved 07 08, 2017, from https://bitinfocharts.com/index_v.html

blockchain.info. (2017). Average Number Of Transactions Per Block. Retrieved 11 10, 2017, from https://blockchain.info/charts/n-transactions-per-block

Buterin, V. (2013). *Ethereum White Paper - A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM.*

Buterin, V. (2015). Understanding Serenity, Part I: Abstraction. Retrieved 11 10, 2017, from https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/

Buterin, V. (2016). *The Mauve Revolution.* Devcon2.

CodeTract. (2017). Inside an Ethereum transaction. Retrieved 11 10, 2017, from https://medium.com/@codetractio/inside-an-ethereum-transaction-fa94ffca912f

coingecko.com. (2017). Retrieved 07 08, 2017, from https://www.coingecko.com/en

coingecko.com. (2017). Retrieved 10 08, 2017, from https://www.coingecko.com/en

coinmarketcap.com. (2017). Cryptocurrency Market Capitalizations. Retrieved 11 10, 2017, from https://coinmarketcap.com/charts/

Courtois, N. T. (2016). *Stealth Address, Ring Signatures, Monero.* University College London.

Dai, W. (1998). B-money. Retrieved from http://www.weidai.com/bmoney.txt

DASH X11. (2017). Retrieved 11 10, 2017, from https://dashpay.atlassian.net/wiki/spaces/DOC/pages/1146918/X11

Decker, C., & Wattenhofer, R. (2013). *Information Propagation in the Bitcoin Network.* 13-th IEEE International Conference on Peer-to-Peer Computing.

Donet, J. A., Pérez-Solà, C., & Herrera-Joancomartí, J. (2014). *The Bitcoin P2P network.*

Driscoll, S. (2013). How Bitcoin Works Under the Hood. Retrieved 11 10, 2017, from http://www.imponderablethings.com/2013/07/how-bitcoin-works-under-hood.html

Duffield, E., & Diaz, D. (2017). *DASH Whitepaper.* Retrieved 11 10, 2017, from https://github.com/dashpay/dash/wiki/Whitepaper

Ethash. (2017). Retrieved 11 10, 2017, from https://github.com/ethereum/wiki/wiki/Ethash

ethereum.org. (2016). Retrieved 11 10, 2017, from https://ethereum.org/token

etherscan.io. (2015). Retrieved 11 10, 2017, from https://etherscan.io/txs?block=0

Gilbert, S., & Lynch, N. (2002). *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services.*

Google Trends. (2017). Retrieved 11 10, 2017, from https://trends.google.com/trends/explore?date=2014-10-10%202017-11-10&q=%2Fm%2F0138n0j1

Information Technology Laboratory. (2013). *Digital Signature Standard (DSS).* National Institute of Standards and Technology.

Interledger Architecture. (2017). Retrieved 11 10, 2017, from https://interledger.org/rfcs/0001-interledger-architecture/

Jacobsen, H.-A. (2015). Consistency & Replication. Technical University of Munich.

Jentzsch, C. (2016). *Smart Contract Security.* Devcon2.

Kshemkalyani, A., & Singhal, M. (2008). *Chapter 14: Consensus and Agreement.* Cambridge University Press.

Merkle Patricia Trie Specification. (2017). Retrieved 11 10, 2017, from https://github.com/ethereum/wiki/wiki/Patricia-Tree

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.*

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies.*

Poelstra, A. (2015). *On Stake and Consensus.*

Proof of Stake FAQ. (2017). Retrieved 11 10, 2017, from https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

Reuters. (2017). JPMorgan's Dimon says bitcoin 'is a fraud'. Retrieved 11 10, 2017, from https://www.reuters.com/article/legal-us-usa-banks-conference-jpmorgan/jpmorgans-dimon-says-bitcoin-is-a-fraud-idUSKCN1BN2PN

Ripple Inc. (2017). Retrieved 11 10, 2017, from https://ripple.com/

Ripple Inc. (2017). Developer Center. Retrieved 11 10, 2017, from https://ripple.com/build/

Ripple Inc. (2017). Ripple GitHub. Retrieved 11 10, 2017, from https://github.com/ripple

ripple.com. (2017). XRP Charts. Retrieved 07 08, 2017, from https://xrpcharts.ripple.com/#/metrics

ripple.com. (2017). XRP Charts. Retrieved 10 08, 2017, from https://xrpcharts.ripple.com/#/metrics

RLP. (2016). Retrieved 11 10, 2017, from https://github.com/ethereum/wiki/wiki/RLP

Saberhagen, N. v. (2013). *CryptoNote v 2.0.*

Schwartz, D., Youngs, N., & Britto, A. (2014). *The Ripple Protocol Consensus Algorithm.* Ripple Labs Inc.

Sharding FAQ. (2017). Retrieved 11 10, 2017, from https://github.com/ethereum/wiki/wiki/Sharding-FAQ

Solidity. (2017). Retrieved 11 10, 2017, from https://github.com/ethereum/solidity/blob/develop/docs/index.rst

Swan, M. (2015). *Blockchain: Blueprint for a New Economy.*

Szabo, N. (2008). Bit gold. Retrieved from https://unenumerated.blogspot.de/2005/12/bit-gold.html

Tschorsch, F., & Scheuermann, B. (2016). *Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies.* IEEE Communications Surveys & Tutorials (COMST).

Wood, D. G. (2014). *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER.*

Appendix

A. List of Mentioned Blockchain Systems with Websites

Augur	https://augur.net/
Bitcoin	https://bitcoin.org/en/
Bitshares	https://bitshares.org/
Burst	https://www.burst-coin.org/
DASH	https://www.dash.org/
Ethereum	https://ethereum.org/
Gnosis	https://gnosis.pm/
Hyperledger Fabric	http://hyperledger.org/projects/fabric
Litecoin	https://litecoin.org/
Monero	http://monero.org/
NEO	https://neo.org/
Nxt	https://nxtplatform.org/
Peercoin	https://peercoin.net/
Ripple	https://ripple.com/
Steem	https://steem.io/
StorJ	https://storj.io/
Zcash	https://z.cash/

B. Snapshot Tables

Data Snapshot 2017-07-08

Criterium	Metric *[Unit]*	Bitcoin	Litecoin	Ethereum	Hyperledger Project	Ripple	Zcash
Supporting Community	***Reddit Subscribers [#]***	255,744	39,602	85,636	--- Linux Foundation, IBM	14,882	3,459
Development Support	***Activity in Public Source Code Repos [#]***	14,090	1,484	5,671	--- Linux Foundation, IBM	1,428	2,556
Longevity	***Age since Initial Release Date [Years]***	8.5 (01-2009)	6 (10-2011)	2 (07-2015)	1.5 (12-2015)	4.5 (10-2012)	1 (10-2016)
Network Activity	***Transactions [# per Day]***	212,140	17,300	248,060	---	665,304	---
Investor Evaluation	***Market cap of native currency [Bn$]***	42	2.5	25	---	10	0.4
Public Awareness and Interest	***Alexa Rank [-]***	6,880	62,478	7,156	128,476	12,697	20,214
Technical Uniqueness of Protocol	***Ordered Attribute Scale [1..5]***	5 Very High (First Mover)	2 Low (Bitcoin Clone)	4 High (Parts of Bitcoin)	5 Very High (First Mover)	5 Very High (First Mover)	2 Low (Bitcoin Clone)
Application Ecosystem	***Ordered Attribute Scale [1..5]***	5 Very Many	3 Middle	4 Many	2 Few	3 Middle	2 Few

Resources

Ripple network activity retrieved from (XRP Charts, 2017).

Longevity retrieved from corresponding websites.[35]

Alexa ranks retrieved from (amazon.com, 2017) with corresponding websites.

All other raw data retrieved from (coingecko.com, 2017) and (bitinfocharts.com, 2017) and aggregated or averaged if conflicting.

[35] Cf. Appendix A.

Data Snapshot 2017-10-08

Criterium	Metric *[Unit]*	Bitcoin	Litecoin	Ethereum	Hyperledger Project	Ripple	Zcash
Supporting Community	***Reddit Subscribers [#]***	346,057	61,877	128,984	--- Linux Foundation, IBM	28,921	6,035
Development Support	***Activity in Public Source Code Repos [#]***	17,257	1,832	7,819	--- Linux Foundation, IBM	1,601	2,846
Longevity	***Age since Initial Release Date [Years]***	8.5 (01-2009)	6 (10-2011)	2 (07-2015)	1.5 (12-2015)	4.5 (10-2012)	1 (10-2016)
Network Activity	***Transactions [# per Day]***	218,908	18,429	282,203	---	958,252	4,576
Investor Evaluation	***Market cap of native currency [Bn$]***	75	2,8	29,5	---	9,9	0,6
Public Awareness and Interest	***Alexa Rank [-]***	5,170	59,237	8,890	94,791	15,236	22,488
Technical Uniqueness of Protocol	***Ordered Attribute Scale [1..5]***	5 Very High (First Mover)	2 Low (Bitcoin Clone)	4 High (Parts of Bitcoin)	5 Very High (First Mover)	5 Very High (First Mover)	2 Low (Bitcoin Clone)
Application Ecosystem	***Ordered Attribute Scale [1..5]***	5 Very Many	3 Middle	4 Many	2 Few	3 Middle	2 Few

Resources

Ripple network activity retrieved from (XRP Charts, 2017).

Longevity retrieved from corresponding websites.[36]

Alexa ranks retrieved from (amazon.com, 2017) with corresponding websites.

All other raw data retrieved from (coingecko.com, 2017) and (bitinfocharts.com, 2017) and aggregated or averaged if conflicting.

[36] Cf. Appendix A.