School of Computation, Information and Technology - Informatics
Technical University of Munich

Bachelor's Thesis in Informatics: Games Engineering

Blockchain and Games: Evaluating Practicality and Impact

Lukas Jonsson

School of Computation, Information and Technology - Informatics
Technical University of Munich

Bachelor's Thesis in Informatics: Games Engineering

Blockchain and Games: Evaluating Practicality and Impact

Blockchain und Spiele: Bewertung der Anwendbarkeit und Wirkung

Author: Lukas Jonsson
Supervisor: Prof. Gudrun Klinker, Ph.D.
Advisors: Sandro Weber, Ph.D.
Submission Date: March 15, 2024

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich diese Bachelor's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, April 24, 2024

LUKAS JONSSON

Abstract

This thesis discusses the practicality and impact of blockchain technologies on game development. Due to the increasingly popular topic of blockchain, so-called blockchain games have been published in the past. This raises the question of whether using this technology as a foundation for games is a valid design choice. In order to understand the arguments, a theoretical basis must first be created. To achieve this, we first summarize the historical background of blockchain technology and the development of Bitcoin to understand why it exists and what it is used for. After looking closer at the Bitcoin blockchain implementation, we introduce Ethereum and its smart contract capabilities, which are needed to write simple Turing-complete programs that the Ethereum Virtual Machine can execute. In addition, examples of smart contracts and their usage in the Unity game engine are presented. The evaluation chapter of this thesis then discusses various arguments for and against blockchain as a tool in software and especially game development. By reading this work, the reader will gain a deeper understanding of blockchain technology and thus be able to follow and evaluate the arguments in this ongoing discussion.

Contents

Eidesstattliche Erklärung **iii**

Abstract **iv**

1 Motivation **1**

2 History **2**
2.1 Problems of digital cash . . . 2
2.1.1 Double-Spending . . . 2
2.1.2 Inflation . . . 2
2.1.3 Byzantine General's Problem . . . 3
2.1.4 Sybil Attack . . . 3
2.2 Cryptography . . . 3
2.3 Blind Signatures . . . 4
2.4 Hashcash . . . 6
2.5 B-Money . . . 7
2.5.1 Protocol Details . . . 7
2.5.2 B-Money creation . . . 7
2.6 Bit Gold . . . 8
2.7 RPOW . . . 9

3 Related Work **11**
3.1 The Bitcoin Blockchain . . . 11
3.2 Ethereum . . . 11

4 Bitcoin Blockchain **13**
4.1 Peer to Peer Network . . . 13
4.2 Digital coins . . . 14
4.3 Storage management . . . 15
4.4 Proof of Work Consensus . . . 16
4.5 Nodes . . . 18

5 Ethereum and Smart Contracts **20**
5.1 Ethereum Virtual Machine . . . 20
5.2 Smart Contracts in Game Development . . . 20
5.2.1 Smart Contract Implementation: Virtual Pets . . . 21
5.2.2 Unity Implementation: Using Virtual Pets in a Game . . . 24

6 Evaluation: Blockchain and Games **26**
6.1 Difference between Bitcoin and Ethereum . . . 26
6.2 The oracle problem . . . 28
6.3 Blockchain as a Buzzword . . . 29

6.4 Problems of Smart Contracts for Software Engineering 31
6.4.1 Problem: Randomness 31
6.4.2 Problem: Immutability 31
6.4.3 Problem: Blockchain Scalability 32
6.5 Opportunities of Smart Contracts for Software Engineering 33
6.5.1 Opportunity: Code Transparency 34
6.5.2 Opportunity: Code Reusability 34

7 Conclusion **35**

8 Future Work **36**
8.1 Scaling Bitcoin with the Lightning Network 36
8.2 Trusted Timestamping 36
8.3 BitVM 37

1 Motivation

Computers and the Internet play an essential role in today's world. Many things in modern society would only work with digital technology. Even back in 2004, Hoffman, Novak, and Venkatesh raised the question of whether the Internet has become indispensable, meaning the integration into one's daily life [1]. Nearly two decades later, this question is still relevant. With a modern computer or mobile phone, one can access publicly available information quickly because every bit of information is copyable on a digital device. Viewing a website, sending an email, or creating a post on X is only possible by copying data. However, the fact that every bit is duplicatable creates a new untrivial problem of digital information. Financial systems rely on the immutability of completed transactions. No one would buy a stock if it were just a file that everyone could easily copy. Spending cash online would not make sense if anyone could duplicate a dollar bill before using it for an online transaction. The standard way to solve this problem is to establish a trusted third party that would monitor everything and log the rightful owner of the file. Still, new problems arise when using a system based on trust.

In 2008, Satoshi Nakamoto published a whitepaper to a mailing list of cryptographers on metzdowd.com [2]. The paper is called "Bitcoin: A Peer-to-Peer Electronic Cash System" [3] and proposes a solution to this problem. It includes the concept of a digital payment system that uses a so-called blockchain. This new technology should solve issues of digital cash like double-spending [4], privacy, and the need for trusted third parties [5]. The blockchain would not only be used to run the Bitcoin protocol but was also the discovery of digital scarcity. A bitcoin is not copyable because of the immutability of the transaction record made possible by the blockchain [6]. Although, are there other fields of application for this solution?

More than a decade later, many industries are trying to use the blockchain to enhance existing systems, one being the gaming industry [7]. As a game developer, the evaluation of the practicality and the impact of blockchain technology in game development raised my interest.

2 History

It is essential to look at the technological pioneers to understand how blockchains work, what led to their invention, and what they are useful for. In the 1980s, the **Cypherpunk movement** emerged. Inspired by the ideas of David Chaum, they believed in cryptography to weaken the governmental influence in the digital realm. The use of cryptographic algorithms could enhance people's privacy. These ideas consequently result in a concept called crypto-anarchism. While a government enforces laws, crypto-anarchism solely relies on laws written in code and provable by math. These technologies are essential to guarantee privacy, especially in countries with poor human rights or corruption. Hence, digital money is also an important research field since money transactions reveal much about the person behind it. [8]

2.1 Problems of digital cash

Over the years, many different concepts were designed to solve the problems of digital cash. Before looking into these protocols, the problems need to be defined.

2.1.1 Double-Spending

Double-spending describes the situation where money is spent twice simultaneously. However, When someone receives money, they must be sure that it belongs to them now and that it was not spent twice at the same time. This problem is only severe in decentralized networks because a centralized database would know the rightful owners of a money unit at any given moment, thus solving the problem. Accordingly, when a trusted third party must be excluded, the problem is not trivial anymore. The network must agree on a network state that contains every user's balances. Whenever someone performs a money transaction, the state of the network defined by all the data it shares changes. When the network is distributed across substantial geographical distances, users might issue two transactions simultaneously and spread them to different network parts. These two parts would then accept different states, which makes reaching a network consensus impossible. The network must find a solution to problems like this to prevent any double-spending attempts. [4]

2.1.2 Inflation

When dealing with digital information, every bit is copyable. Given that the Cypherpunks did not want to use a central database to control the money supply, a solution had to be found to create a form of digital scarcity that no one could control. If one money unit were a regular file, everyone could copy it before spending it online. This action would not only inflate the money supply and thus make it worthless but also make double-spending possible. A suitable protocol must find a way to control the money supply on

a decentralized basis. Therefore, the money creation process must have two properties. Firstly, it must be unaffected by technological progress. With faster computers in the future, the algorithm to create money cannot generate more money the faster or more often it is executed. Secondly, it cannot grant selected network participants exclusive control over it.

2.1.3 Byzantine General's Problem

Inflation and double-spending are solvable problems by agreeing on a network state. However, in a decentralized network, every user has the same rights. Consequently, bad actors are also allowed and cannot be banned. Instead of excluding them, the protocol must be resilient against manipulation or tampering. The Byzantine Generals Problem is a thought experiment that illustrates this situation. It describes a scene where the Byzantine army besieges an enemy castle. The army consists of many small groups of soldiers, each controlled by a general. In order to win the battle and take the castle, the generals must devise a plan. Nevertheless, they can only communicate via messengers because they are geographically separated. Since some generals may be traitors, a consensus mechanism must be found that makes it possible to find a valid strategy as long as most generals are loyal. In a computer context, the generals are nodes in a distributed network. The nodes want to reach a consensus on a network state while bad actors are possible. The resilience against those bad actors influencing a network consensus is called byzantine fault tolerance. [9]

2.1.4 Sybil Attack

The Byzantine general's problem is solvable if there are enough honest nodes in a decentralized network. Thus, an attacker would try to create many fake identities, make them join the network, and force a different consensus. This process is called a Sybil attack. The problem can be solved by designing the protocol to make Sybil attacks very difficult or uneconomical for the attacker. If the cost of performing an attack would be much higher than the reward, it is much less likely that someone would try to carry out such an attack. [5]

2.2 Cryptography

Besides problems related to digital cash, digital privacy is also an important topic, achievable using modern cryptography. It describes the information encoding process to a ciphertext so only designated entities can decrypt it back to plaintext.

The beginnings of cryptography go back to early human history. Even the Greeks used special sticks with tape that would be used to encode a message. The message could not be recovered if the tape was removed from the stick. A second example is the Caesar Shift Cipher of the Roman Empire. [10]

In 1918, Arthur Scherbius invented the Enigma machine to encrypt a message automatically. The device had three disks. The first would encode a plaintext letter into a cipher letter. Consequently, the other two disks would repeat the encoding process so that a letter would be encoded three times. The Allied forces tried to decrypt the machine since

the Third Reich used it for military communication during the Second World War. Alan Turing then cracked the code by inventing machines that could simulate the enigma to decrypt the enemy's communication. A great help would be that the messages contained military keywords that could be used to search for patterns. [11] However, modern-day cryptography is more advanced than the enigma machine.

First, there is **symmetrical cryptography**. It is called symmetrical because a symmetric cryptographic algorithm uses one key to encrypt and decrypt plaintext. Both the sender and the receiver know the key.

Second, there is **asymmetric cryptography**. The critical difference is that it uses key pairs instead of one key. A key pair consists of a public key that is publicly known and a private key that is only known to its owner. They are both mathematically related to one another, which makes several calculations possible. First, a private key can sign data. The signature is then verifiable with the public key. Like a hand signature, cryptographic signatures can be used to verify that the person holding the private key has taken a specific action. For example, the owner can sign a message and prove that it came from them. It is further possible to encrypt data with a public key, making it only decryptable by the respective private key.

Third, there are cryptographic functions, called **hash functions**, that are used for one-way encryption and do not use keys at all. A primary property of hash functions is that their output, called hash, is unrelated to the input data. Nevertheless, the same input always produces the same output while avoiding collisions, meaning two identical inputs that result in the same output hash. One use case for such functions is saving user credentials in a database. Instead of saving passwords as plaintext, only their hashes are stored. When a user tries to log into their account, the system recalculates the hash and compares it to the one saved in the database. [12]

By using the hash function sha256 with the input Hello world, the following hash value can be generated.

64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c (2.1)

These hashes can be combined into an immutable binary hash tree, called Merkle tree [13]. The result of a Merkle tree creation is the tree's root hash. The root can be used to verify whether a transaction was added during the tree creation. Hence, the Merkle tree root serves as a digital fingerprint over the hashes it includes because it cannot be altered after its creation.

2.3 Blind Signatures

In 1982, David Chaum released a paper about a new concept called blind signatures. He described a payment system that could provide cryptographic proof of payment. Two main goals were to make payments untraceable and unlinkable. These targets are achieved by making a bank create cryptographically signed numbers that are usable as monetary units. A user with a bank account can then withdraw the signed number from the bank and send it to a shop owner as a payment. At that point, the shop owner would deposit the signed number into their bank account. As a trusted third party, the bank keeps track of the signed coins without saving personal information and denies the trans-

action if a coin has already been spent. [14]

Since every money unit is only a signed number, the bank cannot link it to the user [15]. Further, some mathematical properties of this concept are defined. First of all, a trusted third party is needed. In a digital payment system, a bank would portray this role. In addition, the three following functions are needed to make the system work.

- A signing function **s'** that is only known to the bank and its publicly known inverse function **s**.
- A commuting function **c** and its inverse function **c'** which are both only known to the payer.
- A predicate **r** used to check for redundancy

Chaum describes the system as follows. In order to send money from the payer to a shop, they need a certificate **x** signed by the bank that contains the amount of money they want to spend. Since the system is designed to provide anonymity to the payer, they encrypt **x** using their commuting function **c**.

$$c\left(x\right) = enc\left(x\right) \ . \tag{2.2}$$

They then send it to the bank, which uses its signing function to sign the amount using **s'**.

$$s'\left(enc\left(x\right)\right) = signed\left(enc\left(x\right)\right) \ . \tag{2.3}$$

The payer receives the signed certificate from the bank afterward. They now use the inverse function **c'** to get the signed value of **x**.

$$c'\left(signed\left(enc\left(x\right)\right)\right) = signed\left(x\right) \ . \tag{2.4}$$

This works because **c** and **c'** share commutativity with **s'** such that

$$c\left(s'\left(x\right)\right) = s'\left(c\left(x\right)\right) \ . \tag{2.5}$$

$$encrypt\left(signed\left(x\right)\right) = signed\left(encrypt\left(x\right)\right) \ . \tag{2.6}$$

As a result, the payer can spend **signed(x)** in a shop that only needs to verify the signature by using the publicly known inverse **s**.

$$s\left(signed\left(x\right)\right) = x \ . \tag{2.7}$$

The shop then sends the **signed(x)** back to the bank, checking whether the transaction was spent twice. The bank now credits the amount to the shop's account. Cryptography makes it possible to verify that the payer cannot spend money they do not own. It also grants anonymity to the users of the system. However, the system relies on a trusted third party. [14]

Problem	Solved	Not Solved
Double-spending	✓	
Decentralization		X
Privacy	✓	

2.4 Hashcash

One of the fundamental technologies that led to the creation of Bitcoin was presented by Adam Back in 1997 [16] to the Cypherpunks mailing list. Hashcash was designed as a **denial-of-service counter measure technique** [17]. This goal is achieved by using the CPU to compute **partial hash collisions** while using a certain amount of electrical energy for the calculations. The following example explains Beck's idea in a more precise way.

Let us assume a company that wants to acquire new customers for its new product. Usually, they can send advertising emails to millions of people daily. Since no computational effort is needed to send an email, there is no reason not to use this spam technique. Even with a minimal success rate, profits can still be made when a customer buys one of their products due to the advertising.

Hashcash solves this spam issue by requiring such computational effort. Before sending an email, the sender must solve a cryptographic puzzle using a hash function. Given a text like an email header and a timestamp, a nonce has to be found that, when concatenated to the text, results in a hash that contains a certain amount of leading zero bits.

sha256(*Hello world*) =

64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c

sha256(*Hello world****600028382***) =

00000*979c7e32777a365712a1bd24ec33c6ddc98bdba259a3a57529bca5665e4*

The nonce **600028382** leads to a hash with 20 leading zero bits.

Since secure hash functions are non-predictable, one-way functions, the best way to compute a valid solution is to use a brute-force strategy by trying out nonce values. If a solution does not contain the right amount of leading zeros, the computation restarts with the nonce incremented by one. This process is repeated until a valid hash is found.

Each required leading zero bit doubles the difficulty of finding a valid hash, as the chance is reduced by half with each bit. This also means that twice as much computing power is required to randomly find a hash in the same amount of time on average. Thus, the difficulty of a valid hash is adjustable by raising or lowering the required bit amount.

While finding a partial hash collision can be time-consuming, verifying whether a given input text results in a particular output hash takes little to no effort. The recipient can verify the hash and confirm the computational effort by providing the found nonce in the email. [16] Consequently, by providing a nonce, it is verifiable that the nonce was generated with a particular amount of work. Hence, the process is called **Proof of Work** (POW). The higher the amount of leading zero bits, the longer it took to find it.

Although the system is called Hashcash, it would be unsuitable as standalone money. The amount of leading zero bits denominates the value of one hash produced by Hashcash. As a result, an increasing amount of money will be created by better computers in the future because better computers produce hashes with more leading zero bits faster. Accordingly, this would result in hyperinflation in the long term. In addition, the system does not prevent double-spending.

Nevertheless, using real-world energy to produce valid hashes lays the fundamentals of digital scarcity later used in the Bitcoin protocol. [18] The Cypherpunks saw the potential

for Proof of Work to form a foundation for digital money. Since digital money should not be duplicatable like regular data, scarcity could be achieved by using Proof of Work in some form. They concluded that a Hashcash Proof of Work token was the direct proof of real-world energy usage needed to create the hash, thus making it a digital representation of energy. [19]

Problem	Solved	Not Solved
Double-spending		X
Decentralization	✓	
Hyper-Inflation		X
Privacy	✓	

2.5 B-Money

After Hashcash was published, it took about one year for Wei Dai to publish his ideas about a peer-to-peer money system in 1998 [20]. He was also inspired by the Cypherpunk movement of the 1990s and tried to create a concept that would fully exclude the government. To preserve the privacy of its users, every user would get a public key as a pseudonym. Dai knew his implementation had flaws, so he presented two approaches to achieve the same functionality. The paper also contained the process of digital contracts and how to enforce them without needing a third party. However, these ideas are not relevant to the cash system he proposed.

2.5.1 Protocol Details

The first protocol needs every participant to store the whole database of balances on their computer. In order to establish a way of communication between the participants, the computers would send and receive messages over messaging channels. Whenever someone wants to send money, they must sign a money-spending message. After that, the message is published to the peer-to-peer network. Verifying that a transaction message is valid and was not created by someone other than the money owner is done by every network participant since each possesses a local copy of the database. Although this protocol is similar to a modern blockchain, Dai did not come up with a mechanism to prevent double-spending. The second protocol is very similar to **Proof of Stake**, an alternative consensus mechanism some blockchains use today. [21]

2.5.2 B-Money creation

A problem Dai had to face while creating his ideas was the process of creating units of money. The first idea described in his paper required the network participants to publish solvable computational problems to the network. These problems are not only solvable, but the computational effort of a valid solution is also verifiable. The network would then compensate the user who solved the problem with an equivalent amount of b-money by crediting it to his account.

Dai knew that the evaluation of the right amount of compensation for the problem solver could be a problem. As a consequence, he proposed a second idea for money creation.

The concept relied on four phases. The participants propose a new money creation rate in the first phase, followed by a bidding phase. Every user who wants to create money has to bid with a computable problem with a defined difficulty. After the bidding is done, they start to solve the problem. Solving the most challenging problem would result in newly created money for the solver.

As an alternative, Dai proposed a second protocol requiring only some servers to store the database. Since this requires trust, they must lock some money as insurance. If they try to cheat the system, they lose their insurance. Even if every server tried to cheat the system, the attempt would still fail because they need to publish their stored data to the network of users occasionally. The network would notice and punish the servers if their data contained falsified information. [20]

Problem	Solved	Not Solved
Double-spending	?	?
Decentralization	✓	
Backed by energy	✓	
Privacy	✓	

2.6 Bit Gold

In 2001, Nick Szabo concluded that trusted third parties are security holes. According to Szabo, personal wealth should not depend on trusted third parties. This also applies to money. [22]

Consequently, in 2005, he revisited an idea he proposed in 1998 about a concept he called Bit Gold. It was inspired by the properties of precious metals that people have historically used as money. Szabo would later use these attributes to create a digital version of gold called Bit Gold. He summarized his idea as a protocol that could transfer and store bits safely without depending on a trusted third party. A Proof of Work function is used to create those bits used as units of Bit Gold. Szabo further described the following steps on how the system works.

At first, a string is created that is used as an input of a Proof of Work function like Hashcash [16] to create an expensive timestamped hash. In addition, a distributed ledger has to store them. If another user wants to create a new hash, the last saved hash in the ledger will be used as a string to create the new one. Thus, a chain of hashes results. Since this concept also relies on Proof of Work functions like hashcash, the value of a unit of Bit Gold is directly tied to the computational effort used to produce the hash. In the context of a Hashcash POW token, the value grows the more leading zero bits the token has. To counter the fact that technological progress results in better computers and thus in more valuable hashes over a shorter period, every Bit Gold has a timestamp. The value of one hash depends not only on its difficulty of creation but also on its creation timestamp. As a result, units of Bit Gold are not fungible since every unit has its unique combination of value and time of creation. A hash with 30 leading zero bits from 2010 would likely be valued higher than one with the same amount of zero bits created in 2023 because creating the hash in 2010 was more challenging.

Szabo concludes that a banking layer is needed to create bills of equal value backed by units of Bit Gold. Having multiple banks would result in a form of decentralization to

preserve the system's integrity. The network nodes solve the double-spending problem by keeping a local copy of the immutable chain of ownership of a Bit Gold unit. [23]

However, two additional security threats exist that Bit Gold, by design, is not secure against. The first is the potential threat of Sybil attacks. Like b-money, Bit Gold stores its database across multiple servers in a decentralized network. The ownership of hashes is secured by every network participant who owns a copy of the ledger and agrees on its integrity. Nevertheless, there is no protection against multiple dishonest network nodes joining the network and publishing a manipulated version of the database to change its state. The second problem is the Byzantine general's problem. How should a distributed network reach a consensus about its state while potential enemies inside the network could try to sabotage it? In conclusion, Bit Gold has the following properties. [24]

Problem	Solved	Not Solved
Double-spending	✓	
Byzantine generals problem		X
Decentralization	✓	
Sybil attacks		X
Backed by energy	✓	
Privacy	✓	

2.7 RPOW

As already stated, Hashcash does not work as standalone money. The proposals of Dai and Szabo relied on Proof of Work to back their money with energy. To continue the research in this field, Hal Finney created an implementation that made Hashcash hashes, which he refers to as a Proof of Work token, reusable, called **Reusable Proof of Work** (RPOW). The reason for this is that a user could create a hash with great difficulty and reuse it for new recipients he has not contacted yet. Since the Hashcash implementation does not prevent double-spending, every user has a private double-spending database to prevent anyone from sending him a token twice. Yet, the database is only known to the user and is not publicly available. Although Hashcash counters this fact by assigning a specific purpose to a POW token, there is still the need for a recipient to store received tokens to prevent someone from reusing the same token. Finney's idea was to provide an RPOW server that exchanges POW tokens for RPOW tokens. Additionally, it stores all exchanged tokens to avoid duplicates of the same hash. Inspired by Nick Szabo's Bit gold proposal from 1998, Finney considered RPOW tokens a digital rarity like gold.

Even though the RPOW Concept used a centralized server to provide its functionality, Finney designed it to be tamperproof by hosting the program on an IBM 4758 chip. The chip uses a concept called **trusted computing** [25] to verify the system's integrity by providing remote attestation to the users. This makes it impossible for Finney to cheat the system, even though he hosted the program himself.

The benefit of this system is that when sent to another user, a POW token will not be discarded after a one-time use. Instead, it can be sent to the RPOW server by the recipient immediately. In turn, they get a fresh RPOW token with the same value reusable for a new action. The old token will be blocked for further use by storing it in the double-spending database. This process also allows for splitting or combining tokens. Since

the computational effort to create a hash collision with a specific difficulty doubles by incrementing the required leading zero bits by one, two RPOW tokens with difficulty 27 equal one RPOW token with difficulty 28. [26] Nonetheless, Finney did not design RPOW as a store of value but to exchange proof of energy consumption and thus digitally represented energy. [19]

Problem	Solved	Not Solved
Double-spending	✓	
Decentralization		X
Backed by energy	✓	
Privacy	✓	

3 Related Work

3.1 The Bitcoin Blockchain

None of the technologies and ideas could solve all the problems of digital cash simultaneously. Yet, the need for such a system grew even further due to the financial crisis starting in 2007. The crisis had several causes. Some of them were a housing price bubble in the US caused by the banking sector, a flawed regulatory environment, and the downfall of Lehman Brothers. [27] Consequently, the Cypherpunks again saw the flaws of a centralized credit-based money system. This led Satoshi Nakamoto, an anonymous programmer, to devise an implementation to solve the problems that made former concepts unsuitable as digital money. As stated in the Bitcoin talk forum, Nakamoto's ideas were greatly influenced by former concepts.

> *Bitcoin is an implementation of Wei Dai's b-money proposal [...] on Cypherpunks [...] in 1998 and Nick Szabo's Bitgold proposal[...].* [28]

Although the Bitcoin whitepaper was released on the thirty-first of October in 2008, the software and its code were made open source on the third of January in 2009. It is the same day Satoshi Nakamoto launched the program for the first time and mined the first Bitcoin block. This so-called genesis block also contained a quote referencing the crisis.

> *Chancellor on brink of second bailout for banks.* [29]

3.2 Ethereum

In 2014, Vitalik Buterin, a programmer and co-founder of Bitcoin Magazine, published a whitepaper about a protocol he named Ethereum. In his whitepaper, the author first points out different projects that use the decentralized design of Bitcoin to create new projects. He concludes that to build a new decentralized protocol, one could either build a new network with new consensus rules or a second layer on top of Bitcoin. While Bitcoin has a scripting language, it is not Turing-complete. Essential concepts like loops do not exist to prevent infinite loops from crashing network nodes. While this design choice makes Bitcoin resilient against attacks, it also limits the possibilities of second-layer solutions. Consequently, he envisioned a new blockchain-based system to execute Turing-complete byte code. Thus, developers could implement so-called smart contracts, first described by Nick Szabo [30], that are stored and run on the blockchain. [31] Szabo described a smart contract as a digital representation or implementation of legal contracts formalized in code. The potential of so-called non-fungible tokens in games implemented using smart contracts is further discussed in [7].

Ethereum would be the base layer for so-called decentralized applications (dapps). Over 42 days in 2014, Ethereum was funded via a presale. One could buy 2000ETH for around 500 us dollars. However, the price was raised step by step later. Sixty million Ether were

created for the presale, and only 80 percent were sold to the public, while the Ethereum Foundation kept the rest. [32]

Although the Bitcoin blockchain is only capable of executing basic operations, there are attempts to change this. Robin Linus released a paper about BitVM in October of 2023. While Ethereum executes the code on its network nodes, BitVM only uses the Bitcoin Blockchain to verify them. As a result, Bitcoin could entirely replace Ethereum's smart contracts, thus making them obsolete. [33]

4 Bitcoin Blockchain

The following chapter will show how the Bitcoin blockchain works and that it has the following properties.

Problem	Solved	Not Solved
Decentralization	✓	
Privacy	✓	
Double-spending	✓	
Byzantine generals problem	✓	
Backed by energy	✓	
Sybil attacks	✓	
Inflation resistant	✓	

4.1 Peer to Peer Network

Most pioneer concepts described in this thesis either relied on a trusted third party or were never implemented due to their susceptibility to double-spending, inflation, Sybil attacks, or the lack of byzantine fault tolerance. The Bitcoin core protocol is designed as a peer-to-peer network. Every computer running the software acts as a node. However, a node can act as an active or passive network node. Active nodes are identifiable by their public IP address, while private ones use their local IP address to connect to the network. This results in public nodes acting as peers accepting incoming connections from around the globe.

In the following, the term peers will reference publicly available nodes, while the term node references a network node inaccessible over the internet. Both peers and nodes can maintain eight outgoing connections at the same time. In addition, a peer accepts up to 117 incoming connections.

If a new node or peer is to join the network, a bootstrapping protocol is used to connect it to other peers. The protocol uses a domain name server to acquire a list of active peers. Otherwise, it connects to hardcoded addresses. After successfully connecting to a peer, it receives even more addresses to establish more connections. In addition, the peer also sends all verified transactions and blocks. The new participant verifies and accepts the data according to the network consensus rules programmed into its software. [34]

Problem	Solved	Not Solved
Decentralization	✓	

4.2 Digital coins

Former implementations of digital cash, like Chaum's ecash, used databases with accounts to track every user's balance. Satoshi chose a different approach, called **Unspent Transaction Outputs** (UTXO). This idea is very similar to Finney's concept of tracking the chain of ownership of RPOW tokens. Instead of units of Bitcoin just being a number in a database, they are tracked in UTXOs. They act like digital cash with a static value comparable to a dollar bill with a particular value.

Every user controls a Bitcoin wallet with a cryptographic keypair. The public key serves as a pseudonym for privacy purposes and is used to derive a wallet address.

Problem	Solved	Not Solved
Privacy	✓	

A money transfer requires the creation of a hash using its latest transaction and the public key of the next owner. This hash is then cryptographically signed by the sender with their private key, thus making the transfer verifiable with their public key.

Similar to Szabo's bit gold, a chain of verifiable ownership is formed. Consequently, a Bitcoin wallet has access to UTXOs with different values. However, one transaction does not only consist of one UTXO. Instead, it has multiple inputs and outputs.

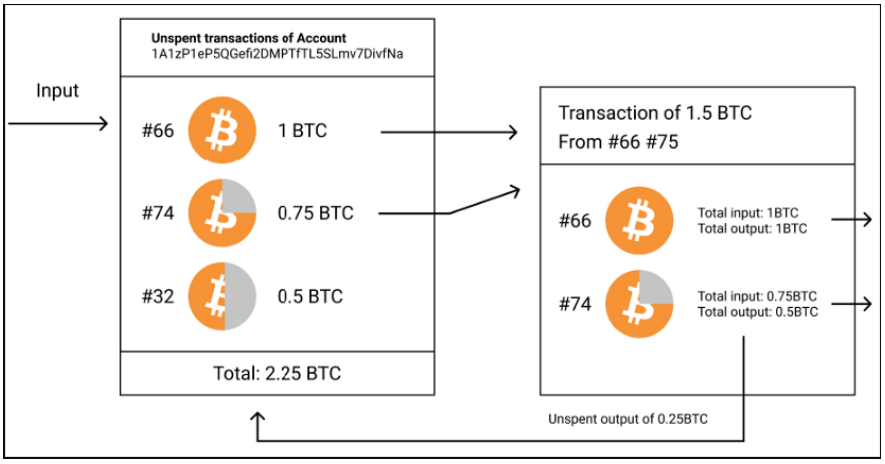

Figure 4.1: Sketch of the UTXO transaction format [35]

Consider a scenario with three users named Alice, Bob, and Greg. Greg owns two UTXOs, each valued at 1 Bitcoin. In total, he owns two bitcoins. If he wants to send 1.2 bitcoins to Alice and 0.3 to Bob, he puts his two UTXOs into the input field of a newly created transaction. The input field is already valid since he wants to spend less than he owns. In addition, Greg must create two new UTXOs in the output field of the transaction to send the money to Alice and Bob. The first new UTXO contains Alice's public key and the amount of 1.2 bitcoins. The second new UTXO contains Bob's public key and the amount of 0.3 bitcoins. Consequently, Greg creates a third UTXO referencing his wallet to receive the remaining value of 0.4 bitcoins as a change, leaving 0.1 bitcoin as a payment fee.

Nevertheless, Satoshi faced a significant problem referenced in his whitepaper. How could a payee verify that the owner did not spend their coins in an earlier transaction?

This double-spending problem resulted in the need for a mechanism to reach a consensus among all network participants. The solution had to be cryptographically verifiable and resilient to Sybil attacks or the byzantine fault tolerance problem. Hence, Satoshi envisioned a distributed timestamping server that kept track of all transactions ever made. These transactions are collected in so-called blocks. Each block is a list of transactions. Every transaction has a list of input and output UTXOs. Besides, a block has a timestamped block hash to save the block confirmation date. Accordingly, all transactions inside a particular block are considered to have happened simultaneously. Ultimately, a chain of blocks is formed by using the previous block hash as input for the next block hash. This system is called blockchain and serves as a decentralized solution to the double-spending problem.

Problem	Solved	Not Solved
Double-spending	✓	

4.3 Storage management

Since every new block results in more storage space taken by the software, the constant growth of the blockchain must be limited. That is why new blocks are created every ten minutes. [3] Using such a period also ensures the network has enough time to spread a newly found block to all network nodes until the next block is found.

Every block in the chain has a block header that is 80 bytes in size. It contains the following parts.

Size	Name
4 Bytes	Version
32 Bytes	Previous block hash
32 Bytes	Merkle root hash
4 Bytes	Timestamp
4 Bytes	nBits
4 Bytes	Nonce

The first four bytes represent the block version. This allows future soft forks to upgrade the block validation rules over time. Next to them are 32 bytes that contain the previous block hash to ensure the immutability of the chain. This hash is built with the SHA256 algorithm. A Merkle tree is built to save the transactions inside a block from manipulation. The root of the Merkle tree can be used to validate a particular transaction. It is 32 bytes in size and also a part of a block header. The next four bytes are used to store the timestamp, followed by four bytes for the mining difficulty target and four bytes for the mining nonce. [36] Mining will be discussed in the Proof of Work chapter of this thesis.

In addition, the number of transactions per block is limited due to a maximum size of one megabyte per block [37]. This results in a maximum potential blockchain growth of:

$$1MB * 6 * 24 * 365 = 52,560MB = 52.560GB. \text{ per year} \tag{4.1}$$

The slow storage increase ensures that running a Bitcoin full node is affordable for everyone, leading to more potential peers on the network. [3] Consequently, even if a central authority owned thousands of nodes, it would still be unable to control the network because users could still build a node and choose the consensus rules themselves. Accordingly, when new versions of the software are published, every user who owns a node decides on their own whether to install it. A new version of the code that would change consensus rules and break compatibility would have to be accepted by nearly every user who runs a node to become a reality in the network. This procedure solves the byzantine fault tolerance problem, as every node can verify the transactions independently. The information is only spread if it is validated. Otherwise, it will be discarded.

Problem	Solved	Not Solved
Byzantine generals problem	✓	

4.4 Proof of Work Consensus

Reaching consensus in a peer-to-peer network is a challenging problem. Since the network is freely usable, it must be resilient against bad actors. Whenever a new block is needed to validate transactions and reach a new network state, someone has to select transactions for the next block and find a valid hash. Therefore, if a bad actor were to create all upcoming blocks, he could censor specific wallets by excluding their transactions forever. If it were predetermined who gets to validate the next block, it would leave room for corruption or censorship. Therefore, the next person to validate a block must be randomly selected. Satoshi used Adam Backs Hashcash to implement a Proof of Work consensus algorithm to achieve that. Each block hash needs to meet specific requirements to be accepted by the network. Like Hashcash, a miner needs to find a nonce so that the block hash is smaller than a particular target. The target is chosen by the network and adjusted after every period of 2014 valid blocks. The adjustment ensures blocks are found every ten minutes on average by raising or lowering the required target for a valid hash. This process is called difficulty adjustment.

Every miner has a certain probability of finding a valid hash for the block header. The more computing power the miner has, the more likely they will be the first to find a valid hash. Nevertheless, miners with lower hash rates also have a statistical chance of validating a block. If one miner controls 20 percent of the network's hash rate, their chance of finding a valid hash and receiving the block reward is also 20 percent. This process is comparable to a lottery, where the contributed hash rate is a lottery ticket. The higher the hash rate, the more lottery tickets. As a result, it is random who gets to verify the next block based on their hash rate contribution. However, whenever a new miner joins the network, the percentage of the total network's computing power changes. Theoretically, a miner that controls at least 50 percent of the network's total hash rate could partially control the mining process by validating more blocks than any other miner. Consequently, he could censor particular wallets or transactions. Ultimately, even if any miner reaches this state, they must preserve their power by accumulating more hashing power as the network grows with new miners joining.

Using this technique solved a problem former electronic cash systems faced: the guaranteed inflation of digital money resulting from technological progress and faster computers in the future. Still, that is not its only purpose. Satoshi described Proof of Work

as one-CPU-one-vote [3]. The more leading zero bits it has, the more hashing power it takes to find the hash. In addition, the block's timestamp further proves how long it took to find it. Thus, A block's hash proves the global accumulated energy used to validate it because creating the hash took time and consumed energy.

Problem	Solved	Not Solved
Backed by energy	✓	

If a new computer is to enter the network, it needs to receive the chain from other peers. Without Proof of Work, malicious peers could create fake chains and send them to new nodes. Only the network difficulty and the block hash provide information that makes identifying the honest chain possible. The chain with the greatest difficulty and, thus, the highest computational effort is most likely the honest chain [3].

In theory, creating a fake chain is only possible if one has more hashing power than the honest part of the network. First, an attacker would have to recalculate all blocks that happened since the block they want to fake. Then, they must calculate new blocks faster than the rest of the network to convince the network nodes that the fake chain is the honest chain. If two chains exist in parallel, every network node will choose the faster-growing chain with the same difficulty and discard the other. Thus, a central authority could only take over the consensus algorithm if it had more energy and hardware in control than the rest. The Sybil attack is prevented by making it uneconomical and difficult to perform a system takeover.

Problem	Solved	Not Solved
Sybil attacks	✓	

Furthermore, the chain uses the Proof of Work consensus mechanism to issue new coins into the ecosystem. Whenever miners find a valid block hash, they receive a transaction that pays a so-called block reward consisting of newly created coins and transaction fees. In turn, miners are incentivized to secure the network with their hashing power, while the amount of new coins rewarded is halved every 210,000 blocks to prevent money supply inflation. As a result, there is an upper supply limit of nearly 21 million bitcoins. Bitcoin uses Proof of Work to solve the double-spending problem and to issue new coins into the system without the risk of hyperinflation or a central authority controlling the supply [18].

Problem	Solved	Not Solved
Inflation resistant	✓	

4.5 Nodes

Bitcoin is a peer-to-peer network consisting of computer nodes running the Bitcoin program. There are four different types of node programs in total that serve different purposes.

- Full node
- Header-only node
- Signing only (Wallet)
- Mining nodes

Full nodes run the original Bitcoin software. Thus, they include all protocol functionalities and keep a full copy of the blockchain. Their main task is to verify incoming transactions or newly found blocks from the miners. If a block is valid, they propagate it to other nodes to spread the information through the network. In addition, they also store a full copy of the blockchain. The first user-friendly program to interact with the Bitcoin blockchain was the Satoshi Client, named after the Bitcoin creator. It had a wallet address and private key to sign and receive transactions and could also participate in mining. Yet, the mining capability was removed to build a separate, more efficient mining client. [38]

Next to full nodes, Satoshi introduced another client that could verify transactions without a full blockchain copy. These clients are called **header-only nodes**. As the name suggests, it only stores the block headers without knowing a block's transactions. It only knows the Merkle tree root hash that contains all transactions. Hence, it relies on full nodes sending block headers to the client. To prevent some malicious full node from sending a manipulated block header, it asks different nodes until it has found the longest Proof of Work chain. As already stated, the difficulty of each block suggests how much effort went into computing the whole chain, which acts as proof of energy consumption. Since these nodes do not store a local copy of the blockchain, they are only partially trustless like a full node. Regardless of being unable to verify new transactions, they can still tell if a particular transaction has happened by calculating whether the Merkle tree root of the transactions in the blocks header contains a particular transaction hash. [3]

Besides these two implementations, there are also **signing-only clients**. They do not even store block headers but only possess a private key. Accordingly, they can only sign new transactions and keep a local copy of their transaction history.

The fourth client is called **mining client**. While initially implemented into the Satoshi Client, they were replaced by more efficient clients using the GPU to achieve more hashes per second. The more hashes a client can generate, the higher the chances of finding a valid hash in the Proof of Work consensus algorithm. [38]

However, more than GPUs are needed to find a valid block today. Over the years, special mining hardware capable of hash rates up to 100 terra hashes per second was designed by several companies, such as Bitmain. [39] Nevertheless, even with this kind of specialized hardware, it is doubtful for one device to find a valid block hash. That is why a concept called pool mining emerged over time. Instead of mining as a single entity, many miners worldwide try to find a valid block hash together. Whenever a valid block is found, all participants get a share according to their hash rate contribution. It is comparable to lottery players splitting the prize whenever any player wins something. [40] According to hashrateindex.com [41], the mining pools with the greatest hashrates are Foundry USA

with 30.1% and AntPool with 26.2%.

5 Ethereum and Smart Contracts

The Ethereum blockchain allows code execution designed in a smart contract language like **Solidity**. As a result, a game can execute primarily defined methods of the smart contract. Using these contracts makes it possible to store in-game items, weapons, or character statistics on Ethereum. [31]

5.1 Ethereum Virtual Machine

Like Bitcoin, Ethereum is a decentralized network that agrees on a network state. A state is a collection of accounts with a 20-byte address. Every account stores its Ether balance, a contract code if one was provided in bytecode form, and a contract storage, which is divided into three types. First, there is long-term storage that uses key-value pairs to store data. To create variables temporarily, developers can use the stack or the memory.

Further, two types of accounts are described. The first one is a standard user account controlled with a private key. They only issue external transactions, meaning transactions from outside the blockchain. The second one is a contract account that controls its smart contract. They can also issue internal transactions from one smart contract to another. Receiving a transaction as a contract account always leads to a code execution and, thus, to a state change. [42] The fact that the blockchain executes Turing-complete code leads to the term **Ethereum Virtual Machine** because the network acts as one computer system.

However, code execution does not happen instantaneously. Like regular transactions, a code execution transaction needs verification in a block. Thus, standard money transfers and code executions compete for block space, resulting in transaction fees, called gas, measured in wei. One wei equals 0.000000000000000001 Ether. Depending on the network usage, the cost of gas changes. In addition, a user can provide a gas limit when issuing a transaction to prevent unintended execution of poorly programmed smart contracts. As a result, developers of smart contracts are heavily incentivized to write high-performance code that uses storage effectively. Consequently, infinite loops are avoided. Crashing the whole network would cost a tremendous amount of Ether, which makes it a wasteful effort.

5.2 Smart Contracts in Game Development

Since the EVM bytecode is Turing-complete, many different things can be done using programming languages like Solidity. This chapter will show some approaches to implementing game-related data using Solidity, the most commonly used smart contract language. It is a high-level, object-oriented, statically typed programming language. It supports various concepts of object-oriented programming, such as inheritance, interfaces, abstract classes, and structures.

5.2.1 Smart Contract Implementation: Virtual Pets

Implementation: Virtual Pet

Used Contract: ERC-721 Non-Fungible Token Standard

Details: This example shows how to implement so-called Virtual Pets using Solidity. The contract is based on the ERC-721 NFT standard. Everyone can create a new pet by using the createNewPet() function. The function mints a new ERC-721 token and randomly draws properties for the new pet.

First, some pet properties are defined in the contract.

```
1  contract VirtualPet is ERC721 {
2      uint256 private _tokenIds; bool existAny;
3      enum PetSize{VERY_SMALL, SMALL, MEDIUM, LARGE, HUGE}
4      enum Personality{Brave, Shy, Curious, Loyal, Playful}
5      enum Elemental_Type{Fire, Water, Earth, Air, Lightning}
6      enum Race{Dragon, Unicorn, Phoenix, Turtle, MagicRabbit}
7      enum AttackType{Melee, Ranged, Magic, Poison, Curse}
8      enum FoodPreferences{Carnivore, Herbivore, Omnivore, Magic_eating,
           Not_eating}
9
10     struct PetProperties{ uint256 date_of_birth; Race race; Elemental_Type
           elementary_type; PetSize pet_size; Personality personality;
           AttackType attackt_type; FoodPreferences food_preferences;}
11
12     mapping(uint256 tokenId => PetProperties) private
           petIdToPropertyMapping;
13
14     constructor() ERC721("VirtualPet", "VP"){}
```

Listing 5.1: Solidity pet properties

However, every pet should have different properties, so functions to generate them randomly are needed. In addition, pet sizes should have different probabilities.

```
1      function drawRandomPetProperties () public view returns(
2          Race race, Elemental_Type elemental_type,
3          PetSize pet_size, Personality personality,
4          AttackType attack_type, FoodPreferences food_preferences){
5
6          race = Race(randomNumber(10) % uint256(Race.MagicRabbit) + 1);
7          elemental_type = Elemental_Type(randomNumber(11) % uint256(
               Elemental_Type.Lightning) + 1);
8          pet_size = drawRandomPetSize();
9          personality = Personality(randomNumber(15) % uint256(Personality.
               Playful) + 1);
10         attack_type = AttackType(randomNumber(20) % uint256(AttackType.
               Curse) + 1);
11         food_preferences = FoodPreferences(randomNumber(4) % uint256(
               FoodPreferences.Magic_eating) + 1);
12     }
```

Listing 5.2: Solidty pet property creation

```
1     function drawRandomPetSize () private view returns (PetSize){
2         uint256 percentage = randomNumber();
3         if(percentage < 1)
4             return PetSize.HUGE;
5         else if(percentage < 5)
6             return PetSize.LARGE;
7         else if(percentage < 15)
8             return PetSize.MEDIUM;
9         else if(percentage < 45)
10            return PetSize.SMALL;
11        else return PetSize.VERY_SMALL;
12    }
```

Listing 5.3: Solidty pet property random function

The random number functions used for this example are deterministic due to the nature of the EVM not having any byte code functionality to create true randomness.

```
1     function randomNumber (uint256 seed) public view returns (uint) {
2         return uint(keccak256(abi.encodePacked(block.timestamp, block.
              difficulty, msg.sender, msg.data, seed)));
3     }
4     function randomPercentage (uint256 seed) public view returns (uint){
5         return randomNumber(seed) % 100;
6     }
7     function randomPercentage () external view returns (uint){ return
          randomPercentage(0); }
8     function randomNumber () public view returns (uint) { return
          randomNumber(0); }
```

Listing 5.4: Solidity random functions

Finally, a function to create a new pet is implemented. In addition, a function to obtain the pet properties is also needed so they can be used in a game.

```
1     function createNewPet() public returns (uint256 petID){
2         _tokenIds++;
3         _mint(msg.sender, _tokenIds);
4         existAny = true;
5
6         uint256 date_of_birth = block.timestamp;
7         (Race race,
8         Elemental_Type elemental_type,
9         PetSize pet_size,
10        Personality personality,
11        AttackType attack_type,
12        FoodPreferences food_preferences) = drawRandomPetProperties();
13        petIdToPropertyMapping[_tokenIds] = PetProperties(date_of_birth,
              race, elemental_type, pet_size, personality, attack_type,
              food_preferences);
14        return _tokenIds;
15    }
```

Listing 5.5: Solidty pet creation function

Then, a getter for external function calls from the Unity game implementation is created.

```
1     function getProperties(uint256 petId) public view returns (
          PetProperties memory){
2         require(existAny && petId <= _tokenIds, "A pet with this id does
              not exist.");
3         return petIdToPropertyMapping[petId];
4     }
```

Listing 5.6: Solidity pet property getter

5.2.2 Unity Implementation: Using Virtual Pets in a Game

After publishing the virtual pet smart contract on an EVM blockchain like Ethereum, a game engine like Unity can communicate with it. For this purpose, the Thirdweb SDK will be used [43].

Code to set up the player wallet for further smart contract interactions.

```
1   ThirdwebSDK sdk;
2   private const string petsContractAddress = "0x....."; // The wallet
        address of the deployed pet contract
3   private const string contractAbi = "..."; // The contract abi (Json
        interface of the contract)
4
5   // Setting up the player wallet
6   public async void playerSetup()
7   {
8       sdk = ThirdwebManager.Instance.SDK;
9       const AuthProvider provider = AuthProvider.Google;
10      var connection = new WalletConnection(
11          provider: WalletProvider.SmartWallet,
12          chainId: 421614,
13          personalWallet: WalletProvider.EmbeddedWallet,
14          authOptions: new AuthOptions(authProvider: provider)
15      );
16
17      await sdk.wallet.Connect(connection);
18  }
```

Listing 5.7: Setup Code

Example code to call a read function on the smart contract and do something afterward.

```
1   // Transaction showing a read call to the pet contract
2   public void readTransaction()
3   {
4       const int tokenID = 0;
5       var petProperties = getPetProperties(tokenID).Result;
6
7       //TODO: Do something with petProperties
8   }
```

Listing 5.8: Read interaction

Example code to call a write function on the smart contract and do something afterward.

```
1     // Transaction showing a write call to the pet contract
2     public async void writeTransation()
3     {
4         var lastTokenID = (await getPetsContract().Read<int>("
              getLastTokenID"));
5         createNewPet();
6         var petProperties = await getPetProperties(lastTokenID);
7
8         // TODO: Do something with petProperties
9     }
```

Listing 5.9: Write Interaction

Helper function to obtain an object representation of the smart contract.

```
1     private Contract getPetsContract()
2     {
3         return sdk.GetContract(petsContractAddress, contractAbi);
4     }
```

Listing 5.10: Smart contract loading

Helper function to call the getPetProperties() function of the smart contract.

```
1     private async Task<PetProperties> getPetProperties(int petID)
2     {
3         return await getPetsContract().Read<PetProperties>("getProperties",
              petID);
4     }
```

Listing 5.11: Get per properties

Helper function to call the createNewPet() function of the smart contract.

```
1     private async Task<TransactionResult> createNewPet()
2     {
3         return await getPetsContract().Write("createNewPet");
4     }
```

Listing 5.12: Call create new pet function

6 Evaluation: Blockchain and Games

Besides the technical possibilities that result from inventions such as Ethereum, what they are useful for is questionable. The blockchain was initially designed to solve the problems of digital money proposed earlier. Ethereum then created a blockchain that could execute Turing-complete code instead of simple operation codes. According to the Ethereum whitepaper, the use case is that decentralized apps or organizations could be built using smart contracts. However, are these apps decentralized? What are the benefits or drawbacks?

6.1 Difference between Bitcoin and Ethereum

Although Ethereum is also based on a blockchain, Bitcoin has critical differences. [44] Some of them lead to the question: Is Ethereum decentralized? This chapter will first show the differences and afterward present some arguments for and against Ethereum.

First, there are technical differences. Besides the fact that Ethereum can run Turing-complete smart contracts, it also has a much shorter block time of only 17 seconds compared to Bitcoin's ten-minute block time. While Ethereum was initially launched with Proof of Work, the consensus was changed to **Proof of Stake** in 2022. [45] Proof of Stake does not rely on finding partial hash collisions. Instead, nodes are selected to verify the next block based on a staked amount of Ether compared to the total staked amount in the network. This amount is locked inside a smart contract and serves as insurance against fraudulent behavior. [46] Earlier in this thesis, a significant problem that comes with the absence of Proof of Work was discussed. New nodes joining the network require other nodes to send them the honest chain. However, without Proof of Work, a node cannot verify that it received the honest chain. In a proof of work context, the honest chain is described as the one with the greatest difficulty, resulting from the fact that an attacker cannot compute a fake chain faster than the rest of the network [3].

A second consequence of Proof of Stake is the centralization of power and wealth. While Proof of Work needs the miner to constantly compete with the rest of the network, Proof of Stake lacks this mechanism. The node with the highest stake has the highest chance of verifying blocks and thus receives most of the block reward. Consequently, the node could stake the Ether it gets to preserve its power in the network. The more coins it has, the more power it has in the network.

Another problem leading to centralized Proof of Stake is that a minimum amount of Ether must be locked to participate in the consensus. By requiring a minimum amount, only some users can participate in the consensus mechanism. Only those who have enough Ether can start staking. However, central services, including crypto exchanges, offer staking services to customers who cannot afford the minimum amount. This results in the service controlling the staking nodes and the network consensus. In contrast, in the Proof of Work consensus, everyone can participate with every hardware that can execute the SHA256 algorithm. Thus, third parties are not needed, which prevents centralization.

Accordingly, centralizing Proof of Work is more difficult because it relies on a central authority being more energy and resource-efficient than the rest of the network. Proof of Stake centralization is possible, with users giving their coins to these services to participate in the mechanism.

However, there are also social differences between Ethereum and Bitcoin. When looking at the first days of Ethereum, one significant difference is that Ethereum was initially funded via a presale. This event led to the creation of a significant amount of Ether that was sold to selected investors. In contrast, the Bitcoin software was published before Nakamoto announced the start of the network. Technically, everyone had the opportunity to participate in the Bitcoin network from day one. Nakamoto never created coins without doing the Proof of Work on his machine. By doing this presale, the Ethereum Foundation took a position with significant influence on the network. Investors that bought Ether from the Foundation expect the Foundation to create the network they announced. Hence, when the Foundation decides to take a different path and change the software, many investors would potentially follow their decision. [32]

Although Bitcoin and Ethereum are designed to be decentralized networks, we conclude that preserving this state is more of a **social consensus**. The narrative and idea of the concept play an essential role. As a result, a decentralized network must also be socially decentralized, meaning that no central entity has enough influence to change the network or influence the network's majority on its own. This is one of the reasons why Nakamoto left the Bitcoin project in its early days. As the creator of the initial protocol, he had a social influence on the network.

An event that further strengthened the critique of the role of the Ethereum Foundation was the **DAO hack** in 2016 [47]. The hack used a so-called reentrancy attack that requires an attacking contract and a vulnerable smart contract.

The following code snippets are taken from [48]. In order to perform the attack, two functions need to be set up. First, the attack function deposits and withdraws funds.

```
1     function attack() external payable {
2         require(msg.value >= 1 ether);
3         depositFunds.deposit{value: 1 ether}();
4         depositFunds.withdraw();
5     }
```

Listing 6.1: Attack function

The vulnerability can be found in the withdrawal function of the vulnerable contract. The caller of the function first receives their funds, and then the contract's internal balance of the caller is updated. However, the caller is a smart contract that changed their fallback function, which is called whenever a payment is received.

```
1     function withdraw() public {
2         uint bal = balances[msg.sender];
3         require(bal > 0);
4
5         (bool sent, ) = msg.sender.call{value: bal}("");
6         require(sent, "Failed to send Ether");
7
8         balances[msg.sender] = 0;
9     }
```

Listing 6.2: Vulnerable withdraw function

The attacking contract now calls the withdraw function again, resulting in a recursion that makes the vulnerable contract send funds to the attacker while never updating its internal balance.

```
1     fallback() external payable {
2         if (address(depositFunds).balance >= 1 ether) {
3             depositFunds.withdraw();
4         }
5     }
```

Listing 6.3: Attackers fallback function

This attack is easily prevented by first updating the caller's balance and then sending the funds.

Vitalik Buterin proposed a soft fork to censor the attacker's wallets in response to this hack. However, because of a software vulnerability in the soft fork, a hard fork was proposed instead, which would roll back the blockchain to an earlier state. Due to the protocol's technical decentralization, Vitalik could not force the hard fork by himself. However, the node operators and miners agreed to update their software. Consequently, the impact of the Ethereum Foundation on this network decision should be questioned. Without the Foundation initially proposing a fork, it might have never led to that outcome. Besides, the DAO hack was not the consequence of a bug in the Ethereum blockchain. Instead, it was a software bug in the DAO's smart contract. According to the attacker, he technically just used the smart contract functionalities and did not commit a crime. [47] The rollback of the blockchain broke the narrative of it being immutable. Payments that had already been made were removed from the chain.

A second event that split the Ethereum community again was the so-called **merge**, which changed the network's consensus rules from Proof of Work to Proof of Stake. It is questionable whether such decisions would become a reality without the influence of the Ethereum Foundation.

As a result, Ethereum's narrative and use case is very different from that of Bitcoin. Accordingly, Ethereum cannot replace Bitcoin, which is often claimed to happen in the future [49]. Given the threat of potential rollbacks or changes in the network rules, it does not work as a digital store of value, such as Bitcoin. However, it has the Turing-complete smart contract functionality, which Bitcoin lacks.

6.2 The oracle problem

The last chapter concluded that Ethereum serves a different purpose than Bitcoin. This thesis also worked out that the purpose of blockchain technology is to solve the problems of digital cash. Before making a statement about the use case of smart contracts on Ethereum, the so-called **oracle problem** has to be defined. The term **oracle** first appeared in a paper by Alan Turing in 1937 [50]. It described a black box that provided noncomputable information to a Turing machine. In 1978, William Howden stated in [51] that software testing also needs oracles. According to Howden, a software test relies on an oracle to tell whether it has passed. Usually, this oracle is portrayed by a human being, a programmer, who checks if the tests are implemented correctly and produce correct results.

If real-world assets such as gold are traded digitally, they are represented by certificates that promise that the respective asset backs them. However, only an oracle, in this case, the bank that stores the gold, can assure that it is in their possession. As a regular owner of such certificates, they have no option but to trust the bank because they cannot enter the bank's safe and check on themselves.

In conclusion, an oracle is an abstract entity that provides information. The problem is that it must be trusted whenever the oracle provides data. It is further complicated to audit the data repeatedly to verify their integrity. Besides, an auditor would be an oracle as well.

Bitcoin solves this problem by not implementing it in the first place. The Proof of Work serves as proof of energy consumption, resulting in a digital representation of a natural resource. No oracle is needed to prove the work, making the blockchain trustless. This detail causes many discussions about using the blockchain to store data about supply chains [52], health care [53], or video game data. However, storing external data on a blockchain does not guarantee integrity. Besides, the initial use case of stored data is subject to change in the future.

Giulio Caldarelli discussed the blockchain oracle problem in [54]. He concluded that oracles are the only way to provide data integrity for external data stored in a blockchain. Besides, oracles are most often centralized and can be compromised or manipulated. While the impact and the possible damage depend on the data, it is still a relevant problem that requires further research. In addition, the author proposes new questions that need a proper answer before using the blockchain in various areas.

> *SUPPLY CHAIN: Can a firm reputation alone counter the oracle problem? If oracles are unable to prevent the upload of unwanted information, who will benefit from blockchain implementation?*
>
> *HEALTHCARE: Can patients themselves be oracles? Can a distributed system also guarantee privacy and security?*
>
> *LAW: What is the legal role of oracles? How can smart contracts be enforceable? How does one prevent illegal smart contracts?* [54]

6.3 Blockchain as a Buzzword

The invention of smart contracts on blockchains, namely Ethereum, led to many games that utilized this technology. One of the first games to actively integrate NFTs and cryptocurrencies was Cryptokitties. The game launched in late 2017 and was published by Dapper Labs. [55] According to their whitepaper, one of their main goals is to make blockchain technology more accessible for the average person. Although the game is marketed as a blockchain game, it has no direct gameplay. It occurs on the Cryptokitties website, where the user has to buy, breed, and sell cats. Consequently, the game is more comparable to a card-collecting game than a classical video game. According to the Cryptokitties team, using digital collectibles in games has vast potential for the gaming industry. However, they identified why these digital items failed to succeed. First, they cannot rely on a trusted third party to issue them. Second, their functionality should be independent of the founder's existence. What is the purpose of a digital collectible when the issuer turns off the servers and makes them inaccessible? [56] These statements are very close to what Szabo concluded in his paper about Third parties being security holes

[22].

Nevertheless, there are a few contradictions between the Cryptokittie's whitepaper and how the game is implemented. One can be seen in the smart contracts developed for the game. [57] Some specific wallet addresses are granted administrative permissions in the code. Explicit function modifiers make certain functions exclusively executable by these addresses. As a result, Dapper Labs could interfere with the Cryptokittie's ecosystem and break the promise of digital scarcity by creating new kitties. They could even upgrade or replace the existing contracts entirely. Moreover, the pictures of the cats are neither stored nor created on the blockchain. Without the official Cryptokitties website and its central services, the cats would lose their personality without these pictures. This would be comparable to a Pokemon card losing the image of the Pokemon. Hence, the game depends on its maintenance by the publisher.

In summary, the development team concludes that digital collectibles should not depend on the creator but implement such dependencies into the game's code. The blockchain makes the peer-to-peer transfer of kitties possible. However, the only solution it provides is preventing double-spending without relying on a trusted third party. Since the game, by design, relies on the founder, a trusted third party is still needed, thus making the blockchain obsolete. By choosing blockchain as a database, the ecosystem benefits from blockchain functionality at the expense of scalability and immutability of data. According to [58], these limitations led to the game's failure.

In 2018, another blockchain game was released that was heavily inspired by Cryptokitties. It was named Axie Infinity and was published by Sky Mavis. The game also relied on collectible entities. This time, they were called Axies, which could be used to fight against other players' Axies. This so-called **Play-To-Earn concept** would reward their players with Axie Infinity's cryptocurrency AXS. Sky Mavis then implemented the Ronin Sidechain because of the number of smart contract calls executed daily to keep the game running. A sidechain runs parallel to a main blockchain while implementing different consensus rules, block size limits, and transaction speeds. [59] In 2021, the game reached significant popularity. With the rising price of the cryptocurrency, it became more difficult to join the game since playing requires a player to own Axies, which are priced in AXS. Thus, a scholarship program was implemented that added the functionality to borrow Axies from other players. Consequently, the early players that already owned Axies lent them to new players, made them play the game, and gave them a small fraction of the earnings. Especially in poor countries such as the Philippines, players depended on their Axie-lenders and the game. However, the value of AXS dropped significantly in 2022. In addition, the bridge between the Ronin Sidechain and the Ethereum Blockchain was hacked in March of 2022 [60]. The hacker stole about 600 million dollars worth of mostly user funds. Therefore, many players who invested in the game to play it and reach a better financial situation lost everything. [61]

Given that a central software to run such games would be a better alternative, the usage of a blockchain works marketing-wise but not from a software engineering perspective. In addition, the Play-To-Earn concept of Axie Infinity, which is possible through blockchain infrastructure, created a Ponzi scheme. Accordingly, the players who owned Axies got richer until they sold their assets, leaving worthless Axies and desperate Axie borrowers behind. This conclusion begs the question, what are the benefits of using a blockchain compared to central services?

6.4 Problems of Smart Contracts for Software Engineering

6.4.1 Problem: Randomness

Since Solidity is executed on a blockchain, there is one problem a programmer might face when implementing game data, meaning randomness. This problem was further discussed in [62]. The reason for this is that smart contracts are deterministic due to the nature of the blockchain. Since all nodes that execute the smart contract code must reach consensus, byte-code does not support random number generation. Although, there are some approaches to achieving deterministic randomness. While they are unsuitable for sensitive algorithms such as casinos or financial systems, they might be enough for game data.

The first approach is to use an oracle. As already described in this thesis, oracles are external sources of data a blockchain can use. Since the Oracle code is not performed on the blockchain, it cannot be manipulated. However, the oracle is also a security risk since it is compromisable.

A second way to generate random numbers is to use information from the current block. An example of this can be seen below. It uses the hash function keccak256 and the abi interface to encode the data to minimal using space data. A deterministic number will be created using block data and the contract caller's address. In addition, a random number seed is also provided.

```
1    // #### Random Number Functions ###
2    function randomNumber (uint256 seed) public view returns (uint) {
3        return uint(keccak256(abi.encodePacked(block.timestamp, block.
             difficulty, msg.sender, msg.data, seed)));
4    }
```

Listing 6.4: Deterministic random number generation in Solidity

6.4.2 Problem: Immutability

Another critical problem that contrasts with modern software engineering concepts is the immutability of smart contracts. Software engineering is a fluent process of implementing features and fixing bugs that occur when testing the application. However, it is not sure whether an application is bug-free. Since smart contracts are often used in a context where wallets own specific assets or rights, making them as secure as possible is crucial. This makes the threat of potential hacking attempts severe since there is much incentive to steal valuable user funds. In addition to intentionally exploiting code vulnerabilities, there is also the possibility of bugs not being found during the testing process.

OpenZeppelin, a software company specializing in smart contract development, proposed a solution to this problem. [63]. They described a proxy upgrade pattern. A proxy contract would be used to delegate calls to a logic contract. The benefit of this design is that the logic contract can be swapped in the background. The user will constantly interact with the proxy. Thus, the contract address of the proxy always stays the same. However, implementing such a concept takes time and effort. First, there are potential storage collisions resulting. A storage collision occurs when both the proxy and the implementation use the same address in storage to store their data.

In this example, the proxy stores the implementation address in the same variable slot where the implementation contract stores an address. To counter this, OpenZeppelin proposed a concept called "unstructured storage," where a random slot is chosen for the proxy variable to ensure no collisions.

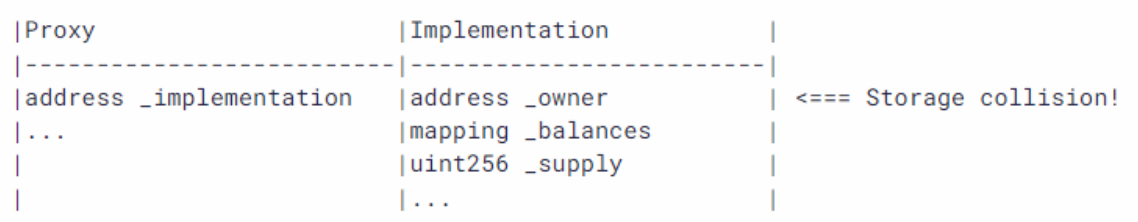

Figure 6.1: A storage collision between proxy and logic data [63]

Another example from OpenZeppelin shows how a collision happens when a contract upgrade is not performed correctly. The order of variables inside the source code must be preserved. If not done correctly, the already saved data clashes with the new field saved at that particular slot.

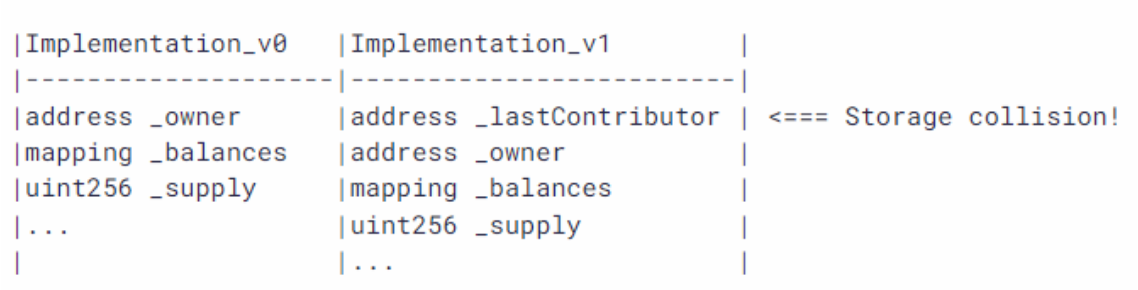

Figure 6.2: A storage collision after a contract update [63]

In addition to storage collisions, there are also function clashes. They are described as two functions named the same or sharing the same 4-byte code identifier, resulting in a collision. A transparent proxy solves this issue by picking the function based on the address of the function caller.

> *If the caller is the admin of the proxy (the address with rights to upgrade the proxy), then the proxy will not delegate any calls and will only answer any messages it understands. If the caller has any other address, it will always delegate a call, whether it matches one of the proxy's functions.* [63]

6.4.3 Problem: Blockchain Scalability

Blockchains' scalability issues come with using blockchains to store external data or execute smart contract logic. Every state-changing action requires a transaction, while all transactions compete for block space with fees. The Bitcoin blockchain processes about seven transactions per second, which is neither suitable as a protocol to use for daily transactions nor would that grant sufficient speed for code execution. Ethereum already has a block time of a few seconds compared to Bitcoin's block time of ten minutes. Yang, Long, Xu, and Peng analyzed the scalability problem of blockchains in [64] and named potential solutions.

One of these solutions is called **Sharding**. A concept initially designed to split large databases into smaller ones to distribute the workload and enhance efficiency. In the blockchain context, smaller chains, called shards, produce blocks that are appended to the main chain. While this process allows for faster transaction speeds and improved network performance, some drawbacks could lead to potential security problems. One of them is a Single Shard Takeover Attack. Taking over the Bitcoin blockchain requires an attacker to have more processing power than the rest of the network. However, taking over a shard requires a percentage of the original amount. Besides, managing the shards and efficiently balancing the workloads requires complex code that could lead to new vulnerabilities and bugs. [65]

Another concept is the use of off-chain payment networks. Instead of changing the blockchain or its consensus rules, the use of the protocol is changed. Accordingly, the underlying blockchain serves as a settlement base layer. Smaller transactions are performed in the payment network. The settlement is done by publishing one transaction containing billions of off-chain transactions. One example of such a network is the **Bitcoin Lightning network** [66]. It creates multi-signature wallets between two parties that serve as payment channels. Transactions between these two entities are not tracked on-chain. Instead, the settlement transaction is done when closing the channel. As a result, network channels can be used to route payments through the network. If Alice has a payment channel with Bob and Bob has a channel with Tom, Alice can send funds to Tom. However, the network also comes with some drawbacks. Since two parties share a payment channel, users are advised to keep their lightning node online to prevent fraudulent behavior. Else, it would be possible to steal funds from the channel. Besides, the network needs liquidity to route payments. If Alice wants to send 0.5 bitcoin to Tom, but the channel between Tom and Bob does not have 0.5 bitcoin for routing, it cannot be done. The only way to transfer liquidity into the network is by issuing main-chain transactions to lock the coins into a payment channel.

Besides the Lightning network, there are other implementations for other blockchains, further described in [64]. While there are more solutions for the scalability problem, more scalability always comes with drawbacks. This consequence is called **blockchain trilemma** [67], meaning the trade-off between decentralization, security, and scalability. Bitcoin was designed concerning decentralization and security, resulting in a low transaction speed and, thus, less scalability. This design choice serves the narrative of a blockchain being a secure payment network without relying on a trusted third party. As a result, using blockchains to store data or execute code is always limited by blockchain technology. The chain would lose its initial purpose if a scaling solution leading to less decentralization or security problems were used. Hence, using blockchains over centralized services will always be less efficient.

6.5 Opportunities of Smart Contracts for Software Engineering

Besides these drawbacks, there are also opportunities. Before evaluating these arguments, it is crucial to note that every argument for blockchains and smart contract usage should exclude any oracles or third parties. Otherwise, the blockchain would lose the decentralization features it was made for, while the software engineering problems would stay. When implementing solutions or concepts using smart contracts, the Oracle problem is the most crucial question to ask. Even if a full game or program is implemented

using only smart contracts, the programmer might always have special administrative rights to control certain contract parts. For example, the administrator could issue virtual items with admin functions instead of using the normal contract functions. Alternatively, some variables of the contract could be changed. If not fully solved or even addressed, centralized services will always be a better design choice to implement data or code logic.

6.5.1 Opportunity: Code Transparency

Due to the nature of public blockchains, all data is available for everyone at any time. Hence, all smart contracts and their byte code are public as well. This transparency could be used as a feature to implement specific game program codes into the chain, such as random number generators for loot drops or crucial game mechanics. However, it is not guaranteed that any game uses these contracts. Furthermore, the company producing a game could change its source code and not use smart contracts anymore. Thus, users again face the oracle problem. However, the argument might be valid if the whole game is programmed with smart contracts, with all the software engineering drawbacks. Games that use this approach can only implement a few game mechanics due to the scalability issues. [68]

6.5.2 Opportunity: Code Reusability

Since anyone can execute smart contracts, different games can use them simultaneously. Character stats or virtual items could be beneficial in different games. In addition, if more than one game uses the virtual item, the value is not dependent on only one-third party. While this argument does not solve the oracle problem, it at least distributes the counterparty risk to many third parties. [68]

Besides, using existing code and a functioning blockchain reduces the workload. With blockchains, an existing infrastructure that requires neither maintenance nor running costs can be used. When game items or logic are already implemented on the chain, games can adopt them and connect their product to an existing ecosystem. However, the oracle problem will always be a part of these solutions, which results in the question of whether centralized services would also work to achieve the same outcome.

7 Conclusion

Blockchain technology was created to solve problems many digital cash protocols failed on. After years of research and proposals of several Cypherpunks, Satoshi Nakamoto presented a whitepaper about Bitcoin. The main feature of this solution is a decentralized peer-to-peer network that does not rely on any third party. Accordingly, it was designed only to process seven transactions per second to focus on decentralization and security in the blockchain trilemma. Every attempt to change the blockchain design by other projects like Ethereum to achieve more scalability leads to less security or centralization. However, both are needed to make the chain a tamper-proof digital cash solution, which it was designed for. Further, the social consensus of the Bitcoin network is the backbone of its value proposition. The decentralized open-source character, the origin story, and its network effects differentiate it from every other cryptocurrency implemented afterward. If other cryptocurrency and blockchain implementations do not contribute any improvements, they might just be trying to imitate Bitcoin and its narrative for marketing purposes.

As a closed ecosystem, the blockchain only contains data secured by its consensus rules. Accordingly, it cannot prove the integrity of external data in any way. This so-called blockchain oracle problem leads to data only being valid when an oracle grants its validity. Ingame items on the blockchain are only valuable if the game's code uses them. If an entity does not abide by the terms of a smart contract, it always needs an external authority to enforce it. In addition, there is no guarantee that a smart contract that is created today will be used for its initial purpose tomorrow. Consequently, third parties are needed again, even though the entire technology should exclude them, and therefore, they must accept significant disadvantages to achieve that. These drawbacks also come into play when using smart contracts instead of a centralized database to store game-related data. Although there are some benefits, it is questionable whether they are significant enough to justify the software engineering problems that come with them.

As a result, we conclude that blockchain technology is a superb implementation for solving the problems of digital cash. However, we do not see any use case for blockchains as a distributed database to store external data. The only data that should be stored on a blockchain is data that was created through a decentralized consensus that is very hard to compromise. Proof of Stake removes the thermodynamical properties provided by Proof of Work, making it impossible to identify the honest chain by its mining difficulty. Therefore, blockchain networks, namely Ethereum, that use the Proof of Stake consensus algorithm cannot store digital wealth. While it is primarily used to execute smart contracts, which are subject to the Oracle problem, it is even questionable whether Ethereum was designed to solve an existing problem or if the need for smart contracts in a blockchain was constructed to justify Ethereum's existence.

8 Future Work

This thesis might serve as a foundation to further expand on what blockchains can be used for. While storing external data on a blockchain does not guarantee the truthfulness of the information, there might still be other use cases for blockchains. We propose a different view to concentrate on second-layer solutions that do not change a blockchain or store any third-party information directly. Instead, layer two protocols could use the blockchain as a distributed timestamping server, which it was initially called by Nakamoto [3], or as a base layer to prove the state of their network. These layers could also choose a more scalable design since the underlying chain is still secure. Storing third-party data on these new systems has no drawbacks like storing them on the main chain, making it a better software engineering approach.

8.1 Scaling Bitcoin with the Lightning Network

The Bitcoin Blockchain creates a fundament for a whole new ecosystem to be built on top. Since its first launch over a decade ago, new systems have been created with Bitcoin as a base settlement layer. One of them is the Bitcoin Lightning network [66], which is capable of routing payments through a network of payment channels that are settled on the Bitcoin blockchain. Every payment channel is a wallet that is controlled by two entities. They can transfer coins inside this channel without waiting for the Bitcoin blockchain to verify it in a new block. This results in billions of daily transactions compared to about seven transactions per second on the blockchain. Thus, games could use Bitcoin Lightning as an alternative to a traditional payment infrastructure for micro-payments.

8.2 Trusted Timestamping

Timestamping is used to prove data's existence at a particular moment. Thus, the services that provide such timestamps must be trustworthy. In addition, the timestamps must be cryptographically saved from being altered afterward. That is where Bitcoin could play an essential role as a distributed time-stamping server. The idea of using the Bitcoin blockchain like that is further discussed in [69]. The author proposes to store hashes of documents or data on the Bitcoin blockchain, which proves the existence of the data at the time of the block validation. As already discussed, storing data on the main chain comes with drawbacks. However, the only property a blockchain can prove about third-party data is when it was stored. If this is the only property needed and the transaction costs are worth the data, it might be a valid use case that needs further research.

8.3 BitVM

Instead of storing data directly on a blockchain, BitVM could be used as a second layer that replaces implementations like Ethereum. We have already concluded that smart contracts on the Ethereum platform are subject to its scalability issues. Further, the contracts need oracles in most cases, creating a contradiction between the initial idea of excluding third parties and the need for trusted data providers and contract enforcers. However, when a second layer is used instead, the software engineering drawbacks are not a problem anymore. While the use case of smart contracts is still questionable, the blockchain is no longer used to execute them. Second-layer smart contract platforms might serve new application areas that need to be discovered by further research in this field.

List of Figures

4.1 Sketch of the UTXO transaction format [35] 14

6.1 A storage collision between proxy and logic data [63] 32
6.2 A storage collision after a contract update [63] 32

Listings

5.1 Solidity pet properties 21
5.2 Solidty pet property creation 21
5.3 Solidty pet property random function 22
5.4 Solidity random functions 22
5.5 Solidty pet creation function 22
5.6 Solidity pet property getter 23
5.7 Setup Code 24
5.8 Read interaction 24
5.9 Write Interaction 25
5.10 Smart contract loading 25
5.11 Get per properties 25
5.12 Call create new pet function 25

6.1 Attack function 27
6.2 Vulnerable withdraw function 27
6.3 Attackers fallback function 28
6.4 Deterministic random number generation in Solidity 31

Bibliography

[1] Donna L. Hoffman, Thomas P. Novak, and Alladi Venkatesh. Has the internet become indispensable? *Commun. ACM*, 47(7):37–42, jul 2004.

[2] Bitcoin p2p e-cash paper. https://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html [Accessed: (05.01.2024)], 2008.

[3] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[4] Usman W. Chohan. The double spending problem and cryptocurrencies. Available at SSRN: https://ssrn.com/abstract=3090174 or http://dx.doi.org/10.2139/ssrn.3090174 [Accessed: (25.11.2023)], 2021.

[5] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[6] G. Soussou L. Popovski and P. B. Webb. A brief history of blockchain. https://pbwt2.gjassets.com/content/uploads/2018/05/010051804-Patterson2.pdf [Accessed: (08.12.2023)], 2014.

[7] Allan Fowler and Johanna Pirker. Tokenfication - the potential of non-fungible tokens (nft) for game development. In *Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '21, page 152–157, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Arvind Narayanan. What happened to the crypto dream?, part 1. *IEEE Security and Privacy*, 11(2):75–76, 2013.

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. 2019.

[10] Tony M. Damico. A brief history of cryptography. http://www.inquiriesjournal.com/a?id=1698 [Accessed: (20.11.2023)], 2009.

[11] Nuno Crato. *The Enigma Machine*, pages 101–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[12] Muharrem Tuncay Gençoğlu. Importance of cryptography in information security. *IOSR J. Comput. Eng*, 21(1):65–68, 2019.

[13] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[14] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[15] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, Jun 2000.

[16] Adam Back. Hashcash. http://www.cypherspace.org/hashcash/ [Accessed: (15.11.2023)], 1997.

[17] Adam Back. *Hashcash - A Denial of Service Counter-Measure*, 2002.

[18] AARON VAN WIRDUM. The genesis files: Hashcash or how adam back designed bitcoin's motor block. *Bitcoin magazine*, Jun 2018.

[19] AARON VAN WIRDUM. The genesis files: How hal finney's quest for digital cash led to rpow (and more). *Bitcoin magazine*, Aug 2018.

[20] Wei Dai. b-money, an anonymous, distributed electronic cash system. http://www.weidai.com/bmoney.txt [Accessed: (15.11.2023)], 1998.

[21] AARON VAN WIRDUM. The genesis files: If bitcoin had a first draft, wei dai's b-money was it. *Bitcoin magazine*, Jun 2018.

[22] Nick Szabo. Trusted third parties are security holes. https://nakamotoinstitute.org/trusted-third-parties/ [Accessed: (15.11.2023)], 2001.

[23] Nick Szabo. Bit gold proposal. https://unenumerated.blogspot.com/2005/12/bit-gold.html [Accessed: (15.11.2023)], 2005.

[24] AARON VAN WIRDUM. The genesis files: With bit gold, szabo was inches away from inventing bitcoin. *Bitcoin magazine*, Jul 2018.

[25] Trusted Computing Group. https://trustedcomputinggroup.org/work-groups/trusted-platform-module/ [Accessed: (15.01.2024)], 2019.

[26] Hal Finney. Rpow. https://nakamotoinstitute.org/finney/rpow/theory.html [Accessed: (25.11.2023)], 2004.

[27] Martin Neil Baily, Robert E Litan, and Matthew S Johnson. The origins of the financial crisis. 2009.

[28] Satoshi Nakamoto. https://bitcointalk.org/index.php?topic=342.msg4508 [Accessed: (05.01.2024)], 2010.

[29] Francis Elliott, Deputy Political Editor, and Gary Duncan. Chancellor alistair darling on brink of second bailout for banks. *www.thetimes.co.uk*, 01 2009.

[30] Nick Szabo. Smart contracts. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html [Accessed: (17.11.2023)], 1994.

[31] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper/ [Accessed: (30.12.2023)], 2014.

[32] What was the ethereum presale? https://cryptoassetrecovery.com/posts/what-was-the-ethereum-presale#:~:text=The%20Ethereum%20presale%20was%20an,blockchain%2C%20at%20a%20discounted%20rate. [Accessed: (08.12.2023)], Mar 2023.

[33] Robin Linus. Bitvm: Compute anything on bitcoin. https://bitvm.org/bitvm.pdf [Accessed: (25.12.2023)], 2023.

[34] Ehab Zaghloul, Tongtong Li, Matt W. Mutka, and Jian Ren. Bitcoin and blockchain: Security and privacy. *IEEE Internet of Things Journal*, 7(10):10288–10313, 2020.

[35] SafePal Knowledge Base. Utxo - what is it and how to use it. https://1859378688-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-Li7KELJ8dC4KTfiiQTZ%2F-MS7JqkZ4c9m8viLVJLo%2F-MS7Jsnbm62zXvb0Sfhs%2Fimage.png?alt=media&token=08f24073-78a5-4e0b-ac8f-675fd3359d24 [Accessed: (20.01.2024)], 2022.

[36] Block chain - block headers. https://developer.bitcoin.org/reference/block_chain.html [Accessed: (03.12.2023)].

[37] J. Göbel and A.E. Krzesinski. Increased block size and bitcoin blockchain dynamics. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6, 2017.

[38] Rostislav Skudnov. Bitcoin clients. https://www.theseus.fi/bitstream/handle/10024/47166/Skudnov_Rostislav.pdf?sequence=1&isAllowed=y [Accessed: (19.11.2023)], 2012.

[39] Bitmain website. https://m.bitmain.com/ [Accessed: (27.11.2023)].

[40] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pages 919–927, 2015.

[41] Bitcoin mining pools. https://hashrateindex.com/hashrate/pools [Accessed: (08.12.2023)], 2024.

[42] Yoichi Hirai. Defining the ethereum virtual machine for interactive theorem provers. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 520–535, Cham, 2017. Springer International Publishing.

[43] Thirdweb sdk. https://portal.thirdweb.com/unity [Accessed: (30.01.2024)].

[44] Henri Arslanian. *Ethereum*, pages 91–98. Springer International Publishing, Cham, 2022.

[45] Elie Kapengut and Bruce Mizrach. An event study of the ethereum transition to proof-of-stake. *Commodities*, 2(2):96–110, 2023.

[46] Fahad Saleh. Blockchain without waste: Proof-of-stake. *The Review of financial studies*, 34(3):1156–1190, 2021.

[47] Cryptopedia Staff. The dao: What was the dao hack? https://www.gemini.com/cryptopedia/the-dao-hack-makerdao [Accessed: (18.02.2024)], Oct 2023.

[48] Kamil Polak. Hack solidity: Reentrancy attack. https://hackernoon.com/hack-solidity-reentrancy-attack [Accessed: (03.01.2024)], 2022. [Online; accessed February 15, 2024].

[49] Yogita Khatri. Jpmorgan says ether will likely outperform bitcoin next year. *The Block*, Dec 2023.

[50] Alan Mathison Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society, Series 2*, 45:161–228, 1939.

[51] W. E. Howden. Theoretical and empirical studies of program testing. *IEEE Trans. Softw. Eng.*, 4(4):293–298, jul 1978.

[52] Arim Park and Huan Li. The effect of blockchain technology on supply chain sustainability performances. *Sustainability*, 13(4):1726, 2021.

[53] Mohsen Attaran. Blockchain technology in healthcare: Challenges and opportunities. *International Journal of Healthcare Management*, 15(1):70–83, 2022.

[54] Giulio Caldarelli. Understanding the blockchain oracle problem: A call for action. *Information*, 11(11), 2020.

[55] Alesja Serada, Tanja Sihvonen, and J. Tuomas Harviainen. Cryptokitties and the new ludic economy: How blockchain introduces value, ownership, and scarcity in digital gaming. *Games and Culture*, 16(4):457–480, 2021.

[56] The CryptoKitties Team. Cryptokitties: Collectible and breedable cats empowered by blockchain technology. `https://www.cryptokitties.co/about` [Accessed: (30.12.2023)], 2017.

[57] Dapper Labs. Kittycore. `https://etherscan.io/address/0x06012c8cf97bead5deae237070f9587f8e7a266d` [Accessed: (30.11.2023)], 2017.

[58] Liu XF Jiang X-J. Cryptokitties transaction network analysis: The rise and fall of the first blockchain game mania. *Front. Phys.*, 9:631665, 2021.

[59] Yiming Lai, Sizheng Fan, and Wei Cai. Quantitative analysis of play-to-earn blockchain games: A case study of axie infinity. In *2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom)*, pages 250–257, 2023.

[60] Matthew S. Smith. The spectacular collapse of cryptokitties. *IEEE Spectrum*, 59(9):42–47, 2022.

[61] Andrew R. Chow and Chad De Guzman. A crypto game promised to lift filipinos out of poverty. here's what happened instead. *Time*, Jul 2022.

[62] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. Probabilistic smart contracts: Secure randomness on the blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 403–412, 2019.

[63] Proxy upgrade pattern - openzeppelin docs. `https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies` [Accessed: (15.01.2024)].

[64] Di Yang, Chengnian Long, Han Xu, and Shaoliang Peng. A review on scalability of blockchain. In *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, ICBCT'20, page 1–6, New York, NY, USA, 2020. Association for Computing Machinery.

[65] What is sharding and how does it work? `https://academy.binance.com/en/articles/what-is-sharding-and-how-does-it-work` [Accessed: (17.01.2024)].

[66] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf` [Accessed: (12.12.2023)], 2016.

[67] Blockchain trilemma: The ultimate guide. `https://www.nervos.org/`

knowledge-base/blockchain_trilemma [Accessed: (15.01.2024)].

[68] Tian Min, Hanyi Wang, Yaoze Guo, and Wei Cai. Blockchain games: A survey. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[69] Bela Gipp, Norman Meuschke, and André Gernandt. Trusted timestamping using the crypto currency bitcoin. In *iConference 2015 Proceedings*. iSchools, 2015.