BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

THE ANALYSIS OF CRYPTOGRAPHIC TECHNIQUES FOR OFFLOADING COMPUTATIONS AND STORAGE IN BLOCKCHAINS

ANALÝZA KRYPTOGRAFICKÝCH TECHNÍK NA ODĽAHČENIE VÝPOČTOV A UKLADANIA V BLOCKCHAINOCH

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. SAMUEL OLEKŠÁK**
AUTOR PRÁCE

SUPERVISOR **Ing. MARTIN PEREŠÍNI**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

155940

Institut:	Department of Intelligent Systems (DITS)
Student:	**Olekšák Samuel, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Cybersecurity
Title:	**The analysis of cryptographic techniques for offloading computations and storage in blockchains**
Category:	Security
Academic year:	2023/24

Assignment:

1. Study principles of blockchains and smart contracts.
2. Get familiar with cryptographic techniques utilized in the context of blockchains (and smart contracts) to offload computations and storage (such as ZKP: zkSNARKs, zkSTARKs, and accumulators, etc.).
3. Look at the existing use cases of techniques and their details in different blockchain ecosystems/platforms.
4. Analyze the privacy properties of studied techniques and the performance in terms of savings w.r.t. computations/storage savings.
5. Select one technique from point 2 and propose a new use case of utilization.
6. Make proof-of-concept implementation and evaluation of the proposed solution.
7. Make a security and privacy analysis of your solution.
8. Discuss the achieved results, possible extensions, and future work.

Literature:

- Ozcelik, Ilker, et al. "An overview of cryptographic accumulators." *arXiv preprint arXiv:2103.04330* (2021).
- Petkus, Maksym. "Why and how zk-snark works." *arXiv preprint arXiv:1906.07221* (2019).
- Fhenix: End-to-End Encrypted Web3.
- NEXUS: Enabling General-Purpose Verifiable Computing powered by zero-knowledge proofs.
- Vitalik Buterin. "STARKs, Part I: Proofs with Polynomials." online.
- Vitalik Buterin. "An approximate introduction to how zk-SNARKs are possible." online.
- Schumm, Daria, Rahma Mukta, and Hye-young Paik. "Efficient Credential Revocation Using Cryptographic Accumulators." *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2023.
- DarkFi, an anonymous L1 based on zero-knowledge, multi-party computation, and homomorphic encryption.

Requirements for the semestral defence:
1-4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Perešíni Martin, Ing.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	6.11.2023

Abstract

The emergence of blockchain technologies has enabled a new perspective on distributed computing and decentralised data management. However, with increasing popularity, platforms face challenges in the form of scalability, since their operation requires cryptographic principles which are computationally difficult. This thesis explores techniques that address this problem by offloading computations and storage from blockchains using zero-knowledge proofs, cryptographic accumulators and other cryptographic techniques. The second part of the thesis proposes a novel approach to implementing a blockchain-based zero-knowledge proof marketplace with proof of useful work (PoUW) consensus protocol.

Abstrakt

Vznik blockchainových technológií umožnil nový pohľad na distribuované výpočty a decentralizovanú správu dát. Avšak so vzrastajúcou popularitou platforiem vznikajú problémy so škálovateľnosťou, keďže ich prevádzka vyžaduje kryptografické princípy náročné na zdroje. Táto práca skúma techniky, ktoré tento problém riešia odľahčovaním výpočtov a ukladania v blockchainoch pomocou zero-knowledge dôkazov, kryptografických akumulátorov a iných kryptografických techník. Druhá časť práce navrhuje nový spôsob implementácie trhoviska zero-knowledge dôkazov založeného na blockchaine s využitím proof of useful work (PoUW) konsenzuálneho protokolu.

Keywords

cryptography, blockchain, zero-knowledge proof, SNARK

Kľúčové slová

kryptografia, blockchain, zero-knowledge dôkaz, SNARK

Reference

OLEKŠÁK, Samuel. *The analysis of cryptographic techniques for offloading computations and storage in blockchains.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Martin Perešíni

Rozšírený abstrakt

Kryptografické techniky na zabezpečenie kľúčových vlastností blockchainov, ako napríklad decentralizácia, integrita dát a distribuovaný konsenzus, sú obecne veľmi náročné na výpočtové zdroje a úložisko. Preto existuje motivácia identifikovať a izolovať komponenty, ktorých výpočet alebo uloženie by sa dali presunúť mimo blockchainu (tzv. *off-chain*). Techniky, ktoré toto umožňujú síce existujú, avšak ich implementácia je pomalá z dôvodu bezpečnostných nedostatkov alebo kvôli zvýšenej réžií.

Táto práca popisuje základné kryptografické princípy potrebné na pochopenie toho, ako fungujú blockchainy. Nasleduje popis toho, ako fungujú dve hlavné techniky využívané na odľahčenie výpočtov a ukladania z blockchainu, ktorými sú zero-knowledge dôkazy a kryptografické akumulátory.

Dvomi z najčastejšie používaných schém zero-knowledge dôkazov sú zk-SNARKy a zk-STARKy, ktoré dokazovateľovi umožňujú presvedčiť overovateľa o platnosti výpočtu definovaného pomocou aritmetického obvodu bez toho, aby overovateľ zistil hodnoty samotných vstupných dát výpočtu. Kryptografické akumulátory zase poskytujú možnosť komprimovaného uloženia zoznamu dátových položiek s rizikom kolízií a taktiež umožňujú opýtanie sa na prítomnosť konkrétnych položiek v zozname bez potreby zverejniť obsah celého zoznamu, čo má široké uplatnenie v oblasti aplikácií na ochranu súkromia.

Keďže vykonávanie výpočtov na blockchaine typicky znamená, že každý uzol podieľajúci sa na konsenze vykonáva daný výpočet, existuje veľká motivácia premiestňovať tento výpočet mimo blockchain. V priebehu rokov vzniklo mnoho techník, ktoré sa na tomto podieľajú, pričom za spomenutie stoja:

- **Bitcoin Lightning sieť** – Lightning sieť umožňuje dvojici používateľov Bitcoinu, aby si bezpečne vytvorili kanál na opakovanú výmenu meny bez toho, aby sa každá transakcia musela zapisovať do blockchainu. Jediné dve operácie, ktoré sa musia doň zapísať sú vytvorenie a uzatvorenie Lightning kanálu. Široké využitie by umožnilo výrazne zvýšiť priepustnosť Bitcoinu, ktorá je na úrovni približne siedmych transakcií za sekundu.

- **Zk-bridge** – Výmena informácií medzi dvomi blockchainami (zdrojovým a cieľovým) vyžaduje, aby cieľový blockchain verifikoval platnosť stavu zdrojového blockchainu. Táto operácia je výpočtovo náročná a je takmer nemožné takýto výpočet vykonávať na blockchaine. Preto zk-bridge navrhol riešenie, kde cieľový blockchain neoveruje platnosť stavu celého blockchainu, ale overuje iba zk-SNARK dôkaz, ktorý dokazuje, že sa konkrétna udalosť na zdrojovom blockchaine stala. Verifikácia zk-SNARKu je operácia, ktorá je uskutočniteľná na blockchaine, avšak vygenerovanie dôkazu prebieha v tzv. *relay* sieti, ktorá beží mimo blockchainu.

- **Mina protocol** – Mina využíva rekurzívne generovanie zk-SNARKov na dokázanie platnosti stavu blockchainu. Vďaka tomu je potrebných na overenie platnosti stavu blockchainu iba 22 kB dát na rozdiel od klasických blockchainov, medzi ktoré patrí napríklad Bitcoin, ktorý vyžaduje stiahnutie stoviek gigabajtov dát na dokázanie, že stav v blockchaine je platný.

V druhej časti práce je navrhnutá blockchainová sieť, ktorá funguje ako trhovisko na SNARKy, kde uzly s nevyužitým výpočtovým výkonom môžu získať odmenu v podobe natívnej meny za generovanie zero-knowledge dôkazov. Sieť podporuje dva druhy transakcií

a to transakcie mincové, ktoré umožňujú účtom v sieti vymieňať si natívnu menu a transakcie dôkazové, ktoré za poplatok vytvárajú účastníci, ktorí majú záujem nechať si vygenerovať dôkaz iným uzlom v sieti. Čerstvo vytvorené transakcie sa umiestnia do zoznamu nedokončených transakcií, z ktorého sa vyberajú účastníkmi na konsenzuálnom protokole (tzv. baníkmi, ang. *miners*), potvrdzujú a vkladajú do novovytvoreného bloku. Potvrdenie transakcie pozostáva zo skontrolovania platnosti transakcie a v prípade dôkazovej transakcie aj z vygenerovania dôkazu, čo je výpočtovo náročné. Táto výpočtová náročnosť je využívaná na implementáciu konsenzuálneho protokolu typu proof of work (dôkaz vykonanej práce), ktorý sa používa aj napr. v sieti Bitcoin. Avšak v typickom proof of work dochádza k výpočtom, ktoré sa neskôr zahodia, pretože neslúžia na nič iné okrem realizácie konsenzuálneho protokolu – v našom prípade sa výpočty nezahadzujú, pretože nimi vznikajú zero-knowledge dôkazy, o ktorých vygenerovanie má záujem pôvodný žiadateľ, ktorý si ich môže prečítať z blockchainu po zverejnení bloku s jeho transakciou.

Navrhnutá blockchainová sieť (resp. jej overenie konceptu) je implementovaná v jazyku Python s využitím knižnice ZoKrates, ktorá poskytuje podporu pre generovanie a overovanie zk-SNARKov. Funkcionalita, bezpečnostné aspekty, možné vylepšenia a možné použitia v existujúcich platformách sú následne evaluované.

The analysis of cryptographic techniques for offloading computations and storage in blockchains

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Martin Perešíni. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Samuel Olekšák
May 23, 2024

Acknowledgements

I would like to thank my supervisor Ing. Martin Perešíni for his guidance and patience during the development of this thesis. I would also like to thank Ing. Ivan Homoliak, Ph.D. for his insights and the expertise he shared with me.

Contents

Chapter 1

Introduction

Cryptographic techniques to ensure key properties of blockchains, such as decentralisation, data integrity, and distributed consensus, are generally very demanding in terms of computational resources and storage. Blockchains are typically distributed networks with data replicated on multiple nodes in the network. Due to the immutability of blockchain data, the storage demands of the network grow over time as new data are added.

Since performing computations and storing data on-chain is expensive, there is motivation to optimise the processes by identifying components that could be addressed off-chain. A plethora of techniques have been proposed and some have already been implemented, but their overall adoption is slow due to security and overhead concerns.

This thesis outlines the cryptographic concepts necessary to understand how blockchains operate and secure their data. Furthermore, techniques used for optimising on-chain storage and computation requirements of blockchains are listed and explained with a focus on zero-knowledge proofs and cryptographic accumulators. The description of specific existing technologies follows along with their security analysis.

In the second part of this thesis, a new use case of utilisation used for optimising computation in blockchain is proposed, designed and implemented – SNARK marketplace. SNARK marketplace is a blockchain designed to facilitate the delegation of computationally intensive proof generation tasks. Proof generation is an important part of the Proof of Useful Work (PoUW) consensus mechanism used in the blockchain, where computational resources are directed toward solving meaningful tasks as opposed to Proof of Work (PoW) used in many blockchain networks including Bitcoin, where miners compete to solve arbitrary cryptographic puzzles, which have use outside of the consensus protocol. This aspect of PoW is often criticised as it consumes significant amounts of energy without producing any tangible real-world value beyond securing the blockchain [26].

Chapter 2 will explain the necessary cryptographic concepts and primitives while focussing on aspects of blockchain, cryptographic accumulators, and zero-knowledge proofs. Chapter 3 will explore existing methods for offloading storage and computations from the blockchain, while Chapter 4 will describe specific technologies and networks utilising them. The SNARK marketplace design, implementation and evaluation will be described in Chapter 5, Chapter 6 and Chapter 7, respectively.

Chapter 2

Cryptographic Concepts

Cryptography [36] is the study of mathematical techniques related to aspects of information security. It is closely related to the field of computer science. Over the centuries, an elaborate set of protocols and mechanisms has been created to address information security issues.

2.1 Cryptographic Goals

Each objective of information security can be derived from one of the following four cryptographic goals. The fundamental purpose of cryptography is to adequately address these goals:

- **Confidentiality**, also called secrecy or privacy, is a service that protects information from unauthorised access or disclosure.
- **Data integrity** is a service that protects against unauthorised data modification.
- **Authentication** is a service that provides the identification of communicating entities.
- **Non-repudiation** is a service that prevents an entity from denying past actions or commitments.

This thesis will refer back to these goals when introducing various cryptographic primitives and algorithms.

2.2 Symmetric Cryptography

Symmetric cryptography [36, 50] has existed for thousands of years, way before the invention of computers as we know them today. It is defined by both parties sharing the same secret key K. Symmetric cryptography works by employing two functions – one for encryption E and one for decryption D. The encryption function is parameterised by the key and plaintext P (message). The output of the function is called ciphertext C and should only allow the recovery of the original plaintext with the decryption function and with the knowledge of K.

Symmetric encryption ciphers can be divided into two categories:

- **Block ciphers** – Blocks of fixed size are encrypted one by one. Examples of this algorithm include DES *(Data Encryption Standard)* and AES *(Advanced Encryption Standard)*.
- **Stream ciphers** – Data is encrypted one digit at a time. Instances of this category are RC4 and GSM A5/1 algorithms.

Symmetric encryption can provide confidentiality, data integrity, and authentication, but its problem lies in the distribution of keys. Keys generally have to be distributed using physical media (USB drives, snail mail, etc.), or asymmetric cryptography could be used. To provide the mentioned authentication, each pair of communicating parties has to set up their own key. This proves challenging in the context of a large pool of N users who want to communicate with everyone else in the pool, since the number of user pairs (keys) U increases quadratically:

$$U = \frac{N * (N - 1)}{2} \tag{2.1}$$

This means that for a company with 100 employees, there need to be 4950 keys so that each employee is able to communicate with any other employee, which is a very large amount also considering that keys should be regularly refreshed.

One of the inherent disadvantages of symmetric cryptography is that it cannot provide non-repudiation, since the same key is always shared between multiple parties.

2.3 Asymmetric Cryptography

As opposed to symmetric encryption, which is based on both parties sharing the same secret key, asymmetric encryption [36] depends on a pair of mathematically linked keys – a private key and a public key. There are a limited number of mathematical problems that can be leveraged to create asymmetric encryption schemes, namely:

- **Knapsack problem** – Optimisation problem which involves finding the heaviest subset of a set of items weighing in sum not more than some predefined maximum value.
- **Integer factorisation** – Task of finding the prime factors of a natural number.
- **Discrete logarithm** – Based on working out the discrete logarithm of an element in a finite cyclic group.

Due to the high computational complexity of the problems, they could not have been reasonably used before the invention of the computer and the attainment of reasonable computational speed. Therefore, asymmetric cryptography is with us for a much shorter time than symmetric cryptography.

Symmetric encryption is generally regarded as much faster per byte in contrast to asymmetric cryptography. Often a hybrid approach is used, where asymmetric cryptography is used to securely distribute a symmetric key before using asymmetric encryption to transfer the bulk of the data.

As mentioned, this type of cryptography leverages a key pair consisting of a private key $K_{private}$ (one that is only kept by a single party) and a public key K_{public} (which can be widely distributed). Based on which side's key pair is utilised, we can distinguish two schemes:

- **Using recipient's key pair** – The message sender knowing the $K_{public}^{recipient}$ can use it to encrypt a message, which can only be decrypted by $K_{private}^{recipient}$. This scheme provides confidentiality but does not address integrity, authentication nor non-repudiation.
- **Using sender's key pair** – The message sender uses their $K_{private}^{sender}$ to produce a proof of message integrity and further provide the authentication and non-repudiation property. This is typically implemented by encrypting a message hash – this process is also known as signing. Confidentiality is not achieved since the decryption key K_{public}^{sender} is public.

By combining these two schemes into one, we can achieve all four cryptographic goals mentioned in Section 2.1. The stacking can be performed in two ways that are equal in their satisfaction of the cryptographic goals (Figure 2.1).

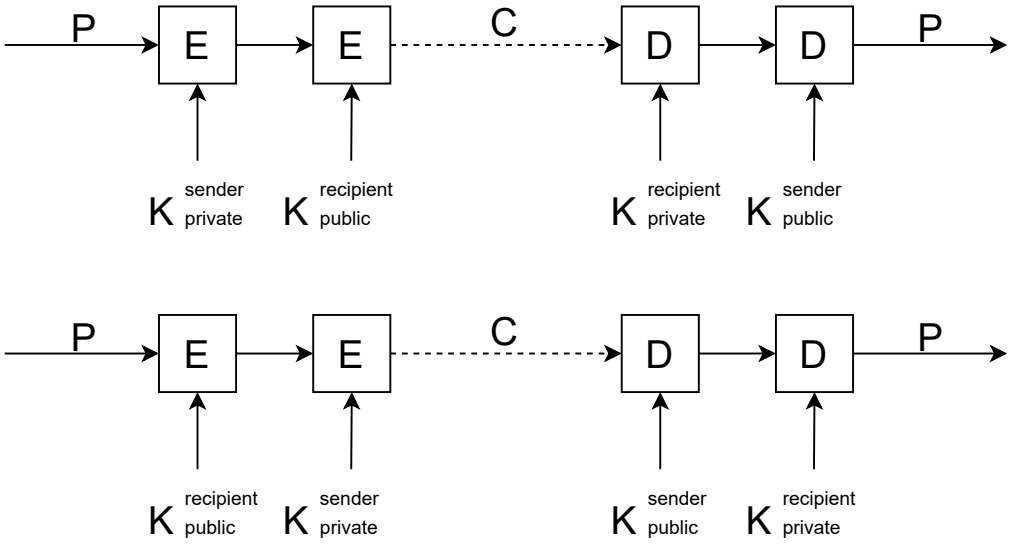

Figure 2.1: Two variants of an asymmetric encryption scheme providing confidentiality, data integrity, authentication and non-repudiation. A dashed line represents an unsecured channel, for example, the Internet.

2.4 Cryptographic Hash Function

A cryptographic hash function (CHF) [49] is an algorithm with the following properties appealing to cryptographic applications:

$$H : \{0,1\}^* \to \{0,1\}^n$$

- **Unconstrained input length** – Thanks to their iterative fashion, CHFs can process input of arbitrary length.
- **Finite output length** – Output (so-called *digest* or *hash*) size is finite.

- **First preimage resistance** – It is computationally infeasible to find x for a given $H(x)$.
- **Second preimage resistance** – It is computationally infeasible to find x' for a given x such that $x \neq x' \wedge H(x) = H(x')$.
- **Collision resistance** – It is computationally infeasible to find inputs x and x' such that $x \neq x' \wedge H(x) = H(x')$.

CHFs can be classified into two main groups – *message digest codes* (MDCs) and *message authentication codes* (MACs). When using algorithms in the MDCs group, the hash can be calculated just from the knowledge of the message. On the other hand, MACs also make use of keys. With MACs not only is the knowledge of plaintext required, but also a knowledge of the key (Figure 2.2).

The importance of CHFs in the rest of modern cryptography, especially blockchains, cannot be overstated. Their applications within the field of blockchains will be discussed in the following sections. They are used in a wide range of applications outside of blockchains, ranging from password storing and verification, file integrity checking, to malware detection, and digital signatures.

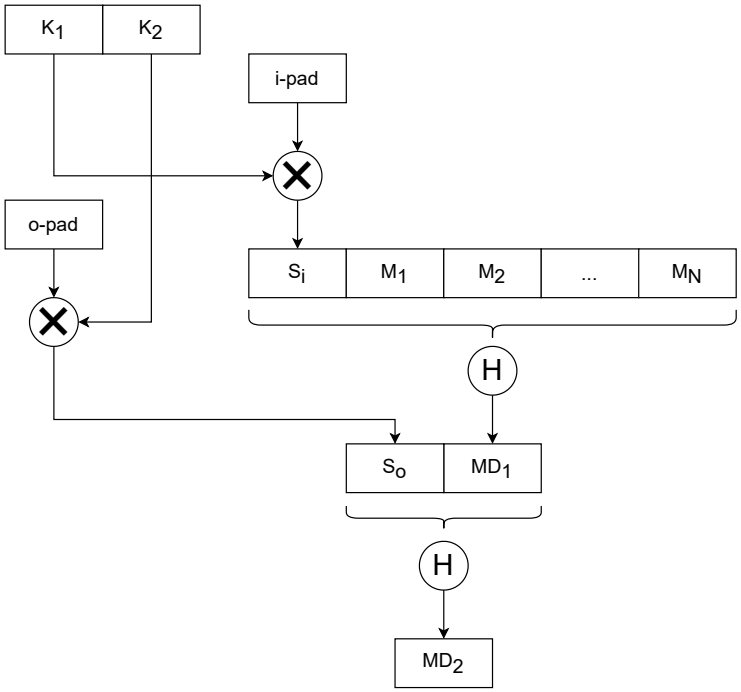

Figure 2.2: Schematic of the HMAC protocol – one of the most popular MAC schemes. The message is represented as blocks M_1 up to M_N, key is split in half into K_1 and K_2. The output of the algorithm is represented by MD_2. Constants *i-pad* and *o-pad* which mix with key halves using XOR operation are inherent to the algorithm and don't change with the key nor message. The nodes marked with H represent a hashing function.

2.4.1 Lamport's Hash Chains

Lamport's hash chain [33] is a mechanism that can be used to implement one-time passwords (OTPs). They are based on repeated application of CHF on a randomly selected seed s. This algorithm is very simple to implement, although it suffers from the fact that OTPs deplete after n generations:

1. The client generates and saves a random seed s.
2. The client applies the hashing function to the seed n times and registers $H^n(s)$ on a server. The number of iterations represents the number of one-time passwords that can be generated from this single registration before they are depleted.
3. The client uses $H^{n-1}(s)$ to authenticate at a server – thanks to the first preimage resistance property of CHFs, the value $H^{n-1}(s)$ cannot be deduced just from $H^n(s)$ without the knowledge of s or any $H^m(s)$, $0 < m < n$. The server can very easily verify the correctness of OTP by hashing it once and comparing it to the previous correct submitted OTP.
4. Upon each attempt to authenticate, the value of n will decrement by one. Thus, the server will reject subsequent attempts to authenticate with the same OTP.

2.4.2 Linked Timestamping

One of the applications of CHFs fundamental to the creation of blockchains is linked timestamping (Figure 2.3). It is used to cryptographically link multiple pieces of data (blocks) into a chain. This is used to immutably record events in the order of occurrence – no block inside the chain can be modified without changing the hash of the entire chain, thus achieving non-repudiation and data integrity.

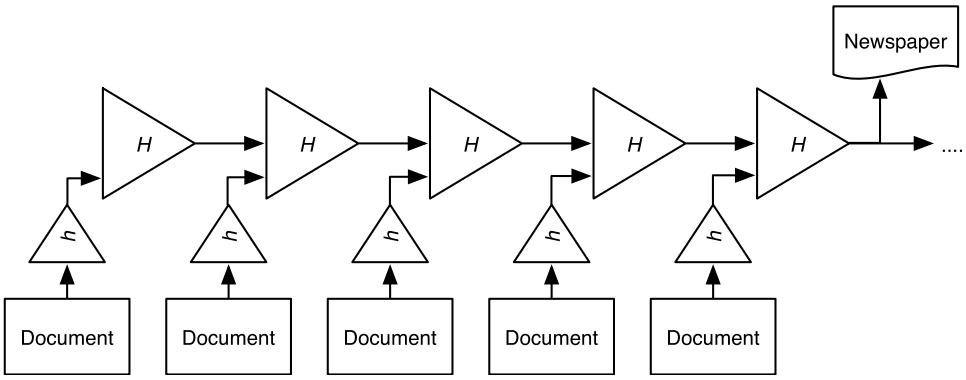

Figure 2.3: Hash chain of newspaper prints used to validate integrity. This chain can be used to prove existence, content, and time of addition of a document in relation to other documents in the chain. Retrieved from [32].

2.4.3 Merkle Tree

A Merkle tree (also known as a binary hash tree), originally described in [37], is a data structure used to ensure data integrity of a large list of items with an efficient membership test by utilising a CHF. The construction of a Merkle tree is described in Figure 2.4 and verification of element presence is visualised in Figure 2.5.

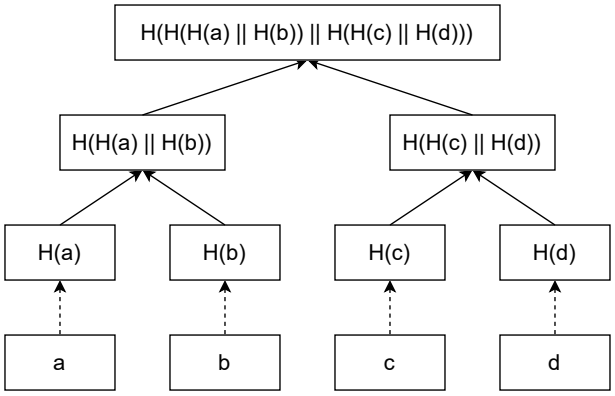

Figure 2.4: The items contained within the Merkle tree are listed as a, b, c and d. The items are first hashed, which creates the leaf nodes of a Merkle tree. Pairs of hashes on the leaf layer are concatenated (represented with || operator in the image) and again hashed to obtain an output of the same size for each Merkle tree node. This is performed repeatedly until only one hash on a layer is left – this hash is called the root hash. If an adversary modified even a single element in the original list, it would completely change the root hash. Thus, the root hash provides integrity of the original list data.

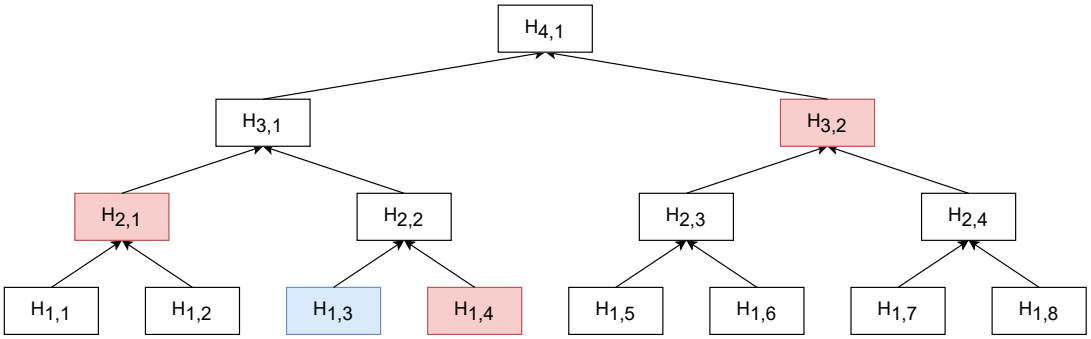

Figure 2.5: To verify the presence of hash $H_{1,3}$ (and its corresponding element) in the Merkle tree, only a small subset of other pair hashes in the tree (so-called Merkle path) needs to be fetched. The number of pair hashes necessary scales logarithmically with the number of original list elements.

In addition to the data integrity use mentioned, Merkle trees have a multitude of other uses across cryptographic and blockchain applications like simplified payment verification (SPV) and zero-knowledge proofs (see Section 3.1).

2.4.4 Merkle-Patricia Trie

Merkle-Patricia trie (MPT) [22] combines Merkle trees and Patricia tries. MPT is used for efficient (in theory with $O(log(n))$ complexity) insertion, deletion, and lookups of items. It allows deterministic and cryptographically verifiable storage of key-value pairs.

If the hash function used satisfies the cryptographic properties highlighted in Section 2.4 then it is computationally infeasible to create two distinct states with the same root hash.

MPTs generally consist of three main types of nodes (demonstrated on an example in Figure 2.6):

- **Branching nodes** – These nodes have up to n child nodes. The value of n depends on the branching factor of the trie, but Ethereum's MPT uses a branching factor of 16, one per each value of the current nibble (resp. hexadecimal character) of the key.
- **Extension nodes** – Used to aggregate nodes that share a common prefix.
- **Leaf nodes** – Leaf nodes store the actual value associated with the complete key and are at the end of the specific key path.

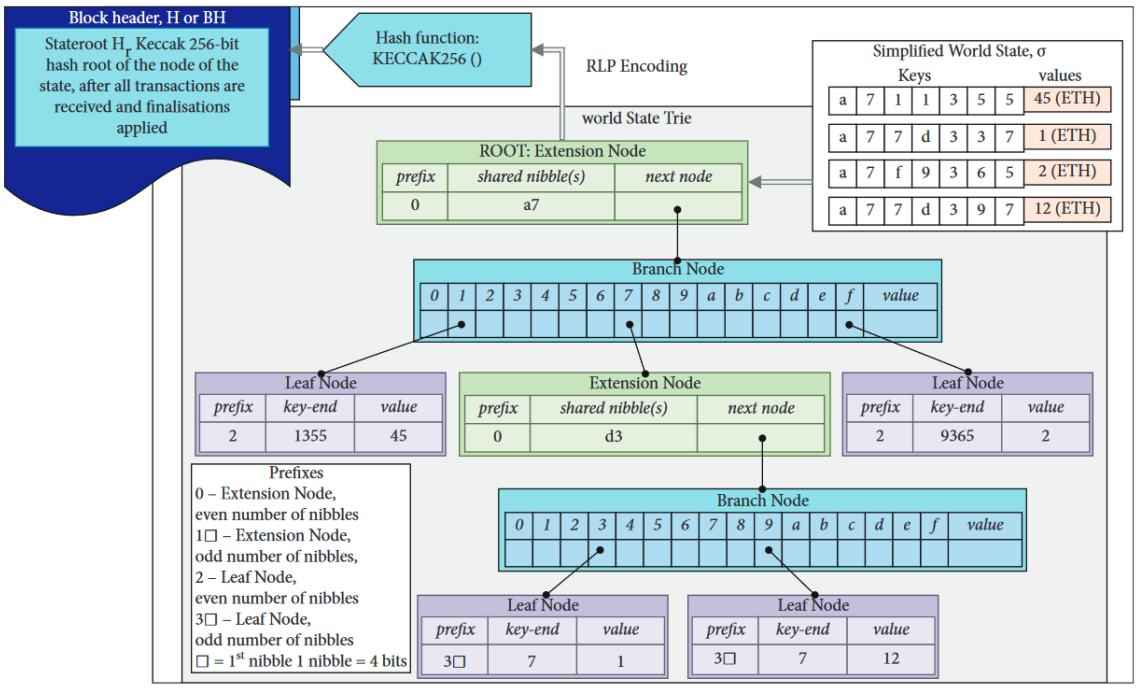

Figure 2.6: Modified version of Merkle-Patricia trie that is used in the Ethereum blockchain. It consists of three kinds of nodes – extension node, branch node and leaf node. Retrieved from [1].

2.5 Blockchain

Blockchain is a technology built on distributed ledgers. Although ledgers have historically been used as means to record financial transactions, blockchains can record any kind of digital event in general.

Bitcoin [39] is undoubtedly the most popular example of blockchain technology, which has found uses in both the financial and non-financial areas. Blockchain principles will be explained using the Bitcoin network, since it was the first blockchain platform and is in essence relatively simple. The description will be kept as general as possible, but there is a huge variety in the blockchain platforms, meaning that these principles might not apply to every blockchain technology.

The general features of a blockchain network are as follows:

- **Immutability** – Thanks to the cryptographic principles described in Section 2.4.2 and Section 2.4.3, the events recorded on blockchain cannot be tampered with – altered nor deleted. This is why blockchains are often referred to as write-only databases.
- **Transparency** – Each transaction on the blockchain is publicly visible and auditable. This did apply for the early blockchain networks like Bitcoin, but this trait is not always desirable, thus many blockchains use zero-knowledge (Section 3.1) and other techniques to implement more confidentiality into the blockchain.
- **Decentralisation** – Blockchains are implemented as peer-to-peer (P2P) networks where decisions are made with distributed consensus. This means that there is no central authority that controls the blockchain, unlike financial institutions, which can be controlled by a small group of people or a government.

The term blockchain refers to the essential data structure used to implement the technology, which consists of blocks that are cryptographically linked together (Figure 2.7). Each block consists of transactions, which represent payments in the network. Linking typically occurs by including the hash of the previous block in the newly created block, protecting the previous block's integrity. For efficient querying of the transaction, two cryptographic structures are used – Merkle trees (Section 2.4.3) and Bloom filters (Section 3.2.2). Merkle trees aggregate transactions of a block and protect their integrity while making querying for transactions within a block efficient. To find a particular transaction inside the whole blockchain without the knowledge in which block the transaction is located, bloom filters are used to rule off the majority of the blocks and greatly shrink the search space.

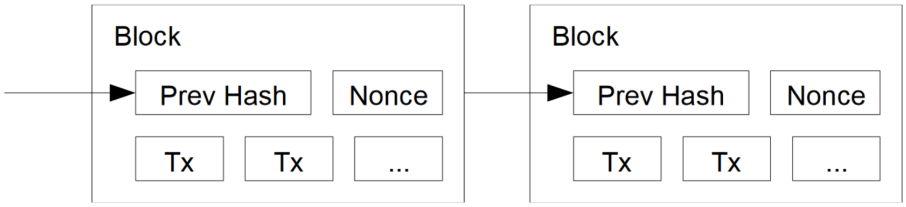

Figure 2.7: Essential components of a typical hash chain. Retrieved from [39].

Each block (Figure 2.8) consists of a header containing metadata about the block including the previous block hash, Merkle root, timestamp, number of transactions, and a nonce – random number used to adjust the block hash until a partial collision is found according to current difficulty. Difficulty is a regulating mechanism used to keep the block creation time constant (in Bitcoin around 10 minutes), regardless of the current computing power of the network.

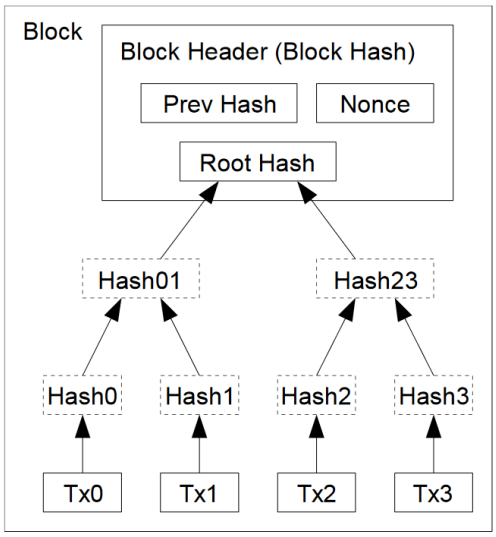

Figure 2.8: Simplified look on a single block of a blockchain along with corresponding Merkle tree containing transactions. Retrieved from [39].

A consensus protocol is used to govern the creation of blocks and the addition of transactions to blocks. These protocols create incentives to participate in the network and penalise malicious nodes that violate the rules of the protocol. Some of the most common consensus mechanisms are described in the following sections.

To understand the problems consensus protocols try to solve, we will look at an overview of common attacks in blockchain networks and how they are remediated:

- **Double spending** – The point of this attack is to spend the same cryptocurrency twice by creating conflicting transactions. This problem is typically remediated by waiting for multiple so-called confirmations (blocks on top of the block that contained the transaction). The deeper the transaction in the blockchains is, the lower is the chance that it will be overwritten.
- **51 % attack** – This attack involves a single entity or a group of colluding entities controlling more than half of the network consensus power. This means that they can produce blocks regardless of the will of the rest of the network, including reversing transactions and stealing funds.
- **Selfish mining** – If an attacker controls a considerable proportion of the network power, they can mine blocks without publishing them. Depending on the power of an attacker, they could accumulate a lead of multiple blocks which, published at the right time, could overwrite blocks on the main chain.

A divergence of a blockchain is called *forking*. Forking can occur for multiple reasons, namely implementation of new features, security updates, or disagreements[1]. Forks are mainly classified into two categories – *soft fork*, which is backward compatible with the previous version and *hard fork*, that is not backward compatible and produces two separate

[1]For an example, see the Ethereum and Ethereum Classic dispute that stemmed from the DAO hack – https://www.bitstamp.net/learn/crypto-101/ethereum-dao-hack/.

blockchains. If forks happen accidentally (like, for example, if two verifiers mine a block at the same time), the dispute is typically resolved using the longest chain rule or the strongest chain rule, since more computational work has been put into them.

Blockchain has a wide range of applications across various areas, including, but not limited to:

- **Financial sector** – Monetary transactions and trading without relying on centralised entities.
- **Digital identity management** – Decentralised identity management with transparent auditing with the possibility of utilising zero-knowledge proofs for improved privacy.
- **Voting** – A secure, transparent, and fraud-resistant voting system could be implemented using blockchain technologies.
- **Asset ownership** – Assets such as real estate and art pieces could be represented by tokens and listed publicly and securely on a blockchain.
- **Proof of authorship** – Blockchains can provide a write-only record of authorship and help with intellectual property rights.

Proof of Work

Proof of work (PoW) [25] was the first consensus protocol to be used in a blockchain network. Its use in blockchains was inspired by the Hashcash algorithm [4], which was used as a countermeasure to denial of service attacks and spam in email and other systems. The idea behind it was to create a cost function that was expensive to compute but easy to verify. The solution was to use a non-interactive, publicly auditable, trapdoor-free cost function with unbounded probabilistic cost. In essence, the algorithm consists of finding a partial hash collision of data and some nonce, typically involving a certain number of leading zeros in the hash output. Since finding partial hash collisions is computationally intensive, the found hash serves as a proof that the sender invested some resources into their action and makes performing a DoS attack much more computationally expensive.

A very similar algorithm is used in Bitcoin today – so-called verifiers (miners) compete to find a first partial collision of a block hash. This is done by repeatedly chaining the nonce value in the block header until some number of leading zeros (depending on the current network parameter called difficulty) in the output hash are found.

Proof of work has been criticised for the amount of energy that is necessary to make the protocol work. Later, alternative protocols have been proposed like proof of stake, proof of storage, proof of authority, and many more.

Proof of Stake

Proof of stake (PoS) was first implemented in the Peercoin protocol [31]. This protocol foregoes the energy-intensive computations of PoW and selects the next block validator pseudorandomly, where the probability of being chosen is proportional to the validator's stake in the network. The largest stakeholders have the highest motivation to prevent the 51 % attack, as they have the most to lose if the trust in the network diminishes. If a malicious validator validates an invalid transaction, then they will lose a part of their

stake (so-called *slashing*), creating an incentive for honesty and maintaining the integrity of a network.

Probably the most notable example of a network using PoS is Ethereum, which switched from PoW to PoS in 2022.

Proof of Authority

Proof of authority (PoA) is a consensus mechanism based on a list of trusted validators that are allowed to validate transactions and are often selected based on their reputation in the network. It is used mostly in private blockchains since if the number of trusted validators is low, it allows a certain degree of centralisation.

2.5.1 Smart Contracts

The first blockchain platforms were generally based on the exchange of currency. With the increase in popularity, came the need for more complex logic to be executed on the blockchain. This need led to the development of smart contracts, which are, in essence, programs stored and executed on the blockchain.

Predating the rise of smart contracts as we know them today, Bitcoin laid the groundwork with its Bitcoin Script language. This stack-based language provided a simple set of instructions dictating conditions how a transaction output can be spent. This simplicity came at the cost of limited functionality for complex applications.

The first smart contract platform is generally considered to be Ethereum, launched in 2015. It introduced the concept of smart contracts as terms of agreement written directly into code using a Turing-complete programming language. The smart contracts were to be executed inside an environment called Ethereum virtual machine (EVM), which defined a set of possible operations and ensured reproducible execution. This allowed the birth of so-called *dApps (decentralised applications)* and *DAOs (decentralised autonomous organisations)*. Ethereum has paved the way for a plethora of decentralised innovation and applications within the blockchain and cryptocurrency space.

2.6 Secure Multi-party Computation

Secure Multi-party Computation (SMPC) [20] is an area of cryptography focusing on cooperative computation between two or more untrusted parties, or even between competitors, while keeping their inputs of computation private.

As an example, we can consider multiple banks that want to collaborate to identify fraudulent transactions. One solution would be to use a trusted third party to gather all customer data across all banks and produce a model based on the data. Alternatively, banks could use SMPC to create a shared model identifying fraudulent transactions without compromising individual customer's data and without having to rely on a third party which has to be trusted by all of the involved banks.

There are multiple cryptographic principles that we can use in SMPC, but one of the most common is *homomorphic encryption*. Homomorphic encryption allows some operations to be performed on the encrypted data without having to decrypt it first. If we consider, for example, additive homomorphism, it applies:

$$E(P_1) + E(P_2) = E(P_1 + P_2) \tag{2.2}$$

If there exists a suitable scheme that supports all the necessary operations, the computation can be performed on the encrypted data, providing security guarantees and facilitating SMPC.

2.7 Cryptographic Commitments

Cryptographic commitment is a concept that involves locking into a value without having to reveal the value itself yet. One of the ways to implement cryptographic commitments is to use a cryptographic hash function to bind a value to a specific representation without disclosing the value.

The process involves two phases – the commit phase and the reveal phase. During the commit phase, the committing party generates a random nonce r, appends it to the commitment value v, and sends its digest $H(v||r)$ to the other party. $H(v||r)$ is referred to as a commitment. The first preimage resistance property of the hashing function prevents the other party from learning the value v or r (hiding property), and at the same time the second preimage resistance property of the same function provides the binding property. In the second phase – the so-called reveal phase – the committing party discloses the value of r and v, allowing the other party to verify whether the commitment received earlier was valid by calculating $H(v||r)$.

This process can be demonstrated on a simple example – a game of Rock paper scissors over a network. Two players are trying to play without a third party. Since perfect time synchronisation over this network is impossible, one of the players might receive the decision of the other player in time to change their own decision so that they win. To prevent this, we can force both players to commit to rock, paper, or scissors before any of the players reveal their choice (Figure 2.9).

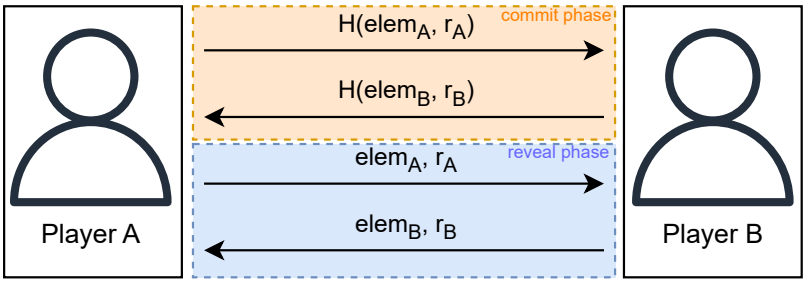

Figure 2.9: A fair game of Rock paper scissors over a network implemented using cryptographic commitments. Values r_A and r_B represent the nonces and $elem_A$ and $elem_B$ represent the rock/paper/scissors choice.

Among the many applications of cryptographic commitments, there are zero-knowledge proofs, electronic voting, time-locking of digital funds, and smart contracts.

2.8 Polynomials

A polynomial (Figure 2.10) is a mathematical expression consisting of variables with non-negative integer exponents and coefficients. Cryptographic proofs often use polynomials as a medium of the proof, since they can efficiently represent large amounts of data.

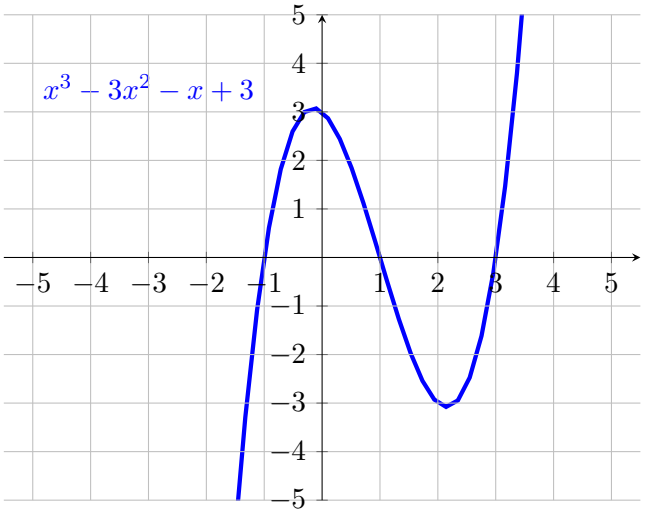

Figure 2.10: A polynomial is a mathematical expression that can be represented as a curve on a graph.

2.8.1 Degree of a Polynomial

Degree of a multivariate polynomial is defined as the largest sum of the variable exponents of all terms[2]. From now on we will only consider single variable polynomials, where the degree will be defined as the largest variable exponent in the polynomial.

The fundamental theory of algebra states that a single-variable polynomial f of degree d can have at most d intersections with the x-axis, thus d real roots. If we consider complex domain, each solvable polynomial has exactly d roots[3].

Two polynomials f and g both of degree d have at most d intersections. This can be proved by defining a polynomial $h = f - g$. When two polynomials f and g intersect for some x their evaluation is equal: $f(x) = g(x)$, rearranged $f(x) - g(x) = 0$. Thus, the newly created polynomial h roots correspond to the intersection points. The polynomial h can have a degree (therefore roots count) of at most d since no multiplication operations have been carried out to increase the degree. This shows that two polynomials of degree d cannot have more than d intersection points.

2.8.2 Constructing Polynomials

Polynomials are very useful for representing information. An example of this are Reed-Solomon codes [47] where a message is represented by points through which a polynomial is threaded. The polynomial is then sampled at additional points to create redundancy and achieve error detection and error correction capabilities when sending a message over an unreliable channel.

There are multiple ways to convert points into polynomials – namely Fast Fourier transform (FFT) and Lagrange interpolation. Let us shortly describe the latter method, since it is simpler.

[2]The degree of $f(x, y) = x^2y^2 + x^3 + y^2 + 3xy^2$ will be 4 since it's the largest sum from the list $(4, 3, 2, 3)$ representing the sum of all variable exponents per each term in the polynomial left to right.

[3]The polynomial $f(x) = x^2 + 1$ has zero real roots, but two complex roots: $x_1 = i$ and $x_2 = -i$.

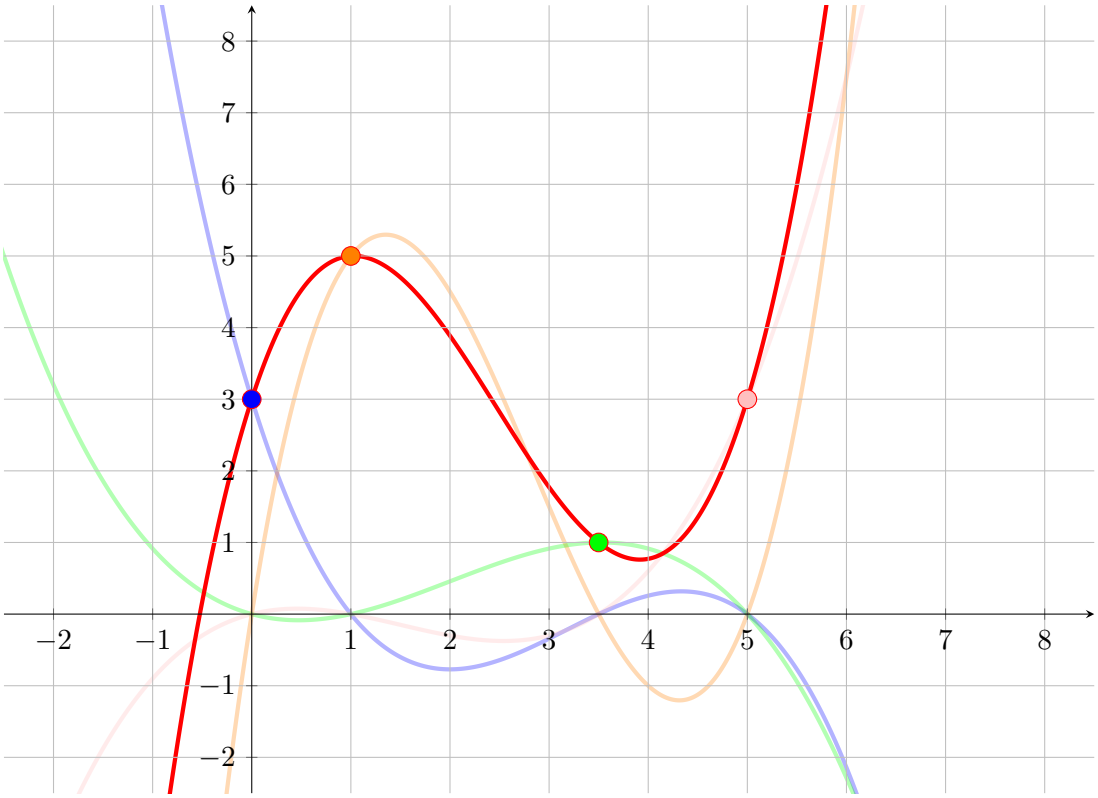

Figure 2.11: Lagrange interpolation with points {(0, 3), (1, 5), (3.5, 1), (5, 3)}. The red curve passes through all four points and is created by summing a partial polynomial (drawn with a transparent colour) for each point.

Lagrange Interpolation

Lagrange interpolation for n points (x_i, y_i) for $i \in \{1, ..., n\}$ representing some message consists of finding n polynomials p_i, one for each point, and adding them to receive the final polynomial representing the entire message. The trick is to choose p_i in such a way that its evaluation is zero for each point in the message except the i-th point. This can be achieved simply by setting x_j as roots of p_i, $j \in \{1, ..., n\}$, $i \neq j$. A graphical example of this can be seen in Figure 2.11.

Unisolvence Theorem

Given a message of n points (x_i, y_i), $i, j \in \{1, ..., n\}$ and for $i \neq j$ it holds that $x_i \neq x_j$ then $\exists! f : deg(f) < n$ where $f(x_i) = y_i$. The polynomial f is called *Lagrange (low-degree) polynomial* and can be found using Lagrange interpolation.

Low-degree polynomials and their detection is an important part of zero-knowledge protocols (described in Section 3.1), since they rely on knowledge of a message represented by a polynomial and exchange of polynomial evaluations between the parties. If we forego low-degree polynomial testing in those protocols, a malicious party could abuse the protocol by using high-degree polynomials, which can be shaped to fit constraints put up to prevent false proofs.

Chapter 3

Offloading Techniques

Two of the most common techniques for secure delegation of work and storage to entities outside of the blockchain are zero-knowledge proofs and cryptographic accumulators.

3.1 Zero-knowledge Proofs

A zero-knowledge (ZK) protocol [11] is performed between two parties – the prover and the verifier. The prover is trying to convince the verifier that some statement is true without revealing any additional information about the statement. A zero-knowledge proof must satisfy the following three properties:

- **Completeness** – If the statement is true, an honest verifier will be convinced by an honest prover.
- **Soundness** – If a malicious prover tries to prove a false statement, an honest verifier will not be convinced with sufficiently high probability.
- **Zero knowledge** – Proving the statement will not reveal any other information about the statement (except truthiness) to the verifier.

Let us explore the soundness property more and talk about what it means to be convinced with a sufficiently high probability. We will consider an abstract example of a zero-knowledge protocol about two balls and a colour-blind friend. We will be taking a role of prover who has two balls – one blue, one yellow – who is trying to convince our colour-blind friend (the verifier) that those two balls are distinguishable. We will want to prove that we can distinguish them, but we do not want to reveal which is blue and which is yellow to the verifier.

The protocol will consist of the verifier taking both balls and holding one in each hand, hiding them behind their back, and possibly swapping them. The verifier will remember whether he swapped them or not. The job of the prover is to say whether they were swapped or not based on the colour vision he is trying to prove. If a prover correctly identified whether the swapping occurred or not, the verifier is now convinced with a probability of 50 % that the prover can distinguish colours.

Now, they can repeat the protocol for as many rounds r as it takes to convince the verifier with sufficiently high probability p, where $p = 1 - 0.5^r$. Unfortunately, we will never be able to prove the statement with 100 % certainty, but this is generally acceptable with zero-knowledge proofs in practice.

In general, the exact properties of zero-knowledge proofs depend on the particular protocol used, but they generally provide some of the following benefits:

- **Succinctness** – Proof size is much smaller than the actual statement it is proving, therefore, both network bandwidth and storage can be saved.
- **Enhanced privacy** – No additional information about the statement must be shared except the truthfulness of the statement. As an example, we can consider the Bitcoin protocol. In it, each input and output of a transaction amount is publicly known. Bulletproofs [14], a cryptographic technique for range proofs, could be used by a transactor to prove that they have sufficient funds to perform a transaction without having to disclose the exact amount of Bitcoin (BTC) they own nor the amount that is transacted.
- **Performance** – A party can prove to other parties that a computation was performed correctly using a zero-knowledge proof, thus the other parties do not need to rerun the computation, they just need to verify the proof, which is often computationally much cheaper. The efficiency gains of the ZK proofs increase linearly with the number of validators [11].

3.1.1 Zero-knowledge Arguments

A zero-knowledge argument is very similar to a zero-knowledge proof with a single important distinction – ZK proofs are resistant against an adversary with computationally unbounded resources, as opposed to a computationally bounded adversary in ZK arguments. Most conventional ZK arguments are based on cryptographic hash functions, which have the property of finite co-domain. This means that a computationally unbounded adversary is guaranteed to find a hash collision after hashing at most $n + 1$ different inputs, where n is the cardinality of the hash function co-domain (Pigeonhole principle).

3.1.2 Zk-SNARKs

Zk-SNARK [43] is a type of zero-knowledge proof developed in the early 2010s. SNARK stands for Succinct Non-interactive Argument of Knowledge and, as the name implies, it allows production of proofs which are very short in comparison to the original statement being proven and are also fast to verify. Non-interactivity allows prover to send a proof to the verifier in a single message without needing to communicate back-and-forth, as is the case with interactive proofs, such as zk-STARKS. On the other hand, one of the drawbacks of SNARKs is the necessity to perform the so-called trusted setup. The security aspects of trusted setup are discussed in Section 3.1.4.

The process of obtaining SNARKs typically starts with a program in a high-level language, such as Rust or ZoKrates. The ability to use a particular language and the restrictions applied depend on the compiler used. Examples of popular SNARK compilers include Zokrates (Section 4.3), zkLLVM (Section 4.6) and Circom. The compiler converts the source program into an arithmetic circuit by mapping each operation to its arithmetic equivalent.

Arithmetic circuits are used to represent computations using mathematical operations, specifically addition and multiplication. They consist of nodes (gates) and edges between them. Each gate performs an arithmetic operation, while each wire transports a value between nodes. The gates perform operations on the inputs they receive, and the results are

passed through the circuit until the final output is obtained. An example of an arithmetic circuit is shown in Figure 3.1 along with the next step of SNARK computation, which is the conversion into Rank 1 Constraint System (R1CS).

R1CS represents the program in a system of linear equations which capture the relationships between the inputs, intermediate values, and outputs. R1CS provides a standardised way to represent computations used in SNARK proofs. This allows different programming languages and compilers to target R1CS, making it easier to develop interoperable solutions and perform audit.

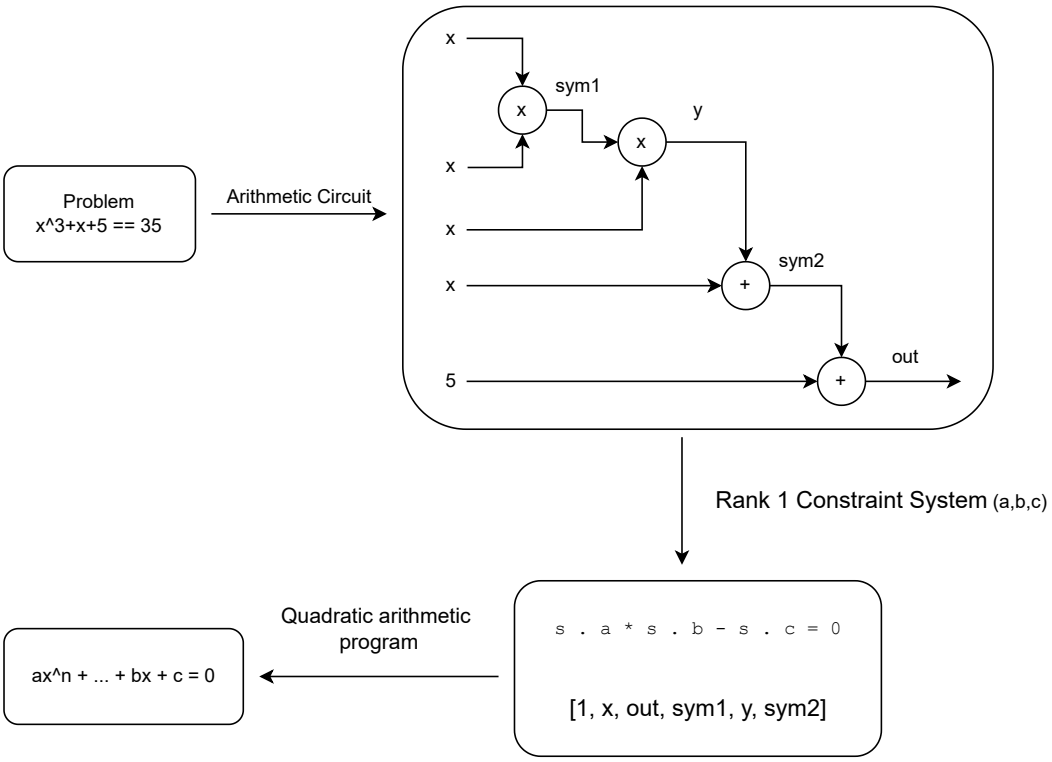

Figure 3.1: Steps of transforming a mathematical problem into a zk-SNARK. Retrieved from [48].

3.1.3 Zk-STARKs

Zk-STARK (Zero-knowledge Succinct Transparent Argument of Knowledge) [8, 10] is a zero-knowledge proof system proposed in March 2018. They are similar to zk-SNARKs but offer several advantages, including transparency and post-quantum security. STARKs eliminate the need for trusted setups that introduce a potential vector for an attack. STARKs do not rely on the hardness of factoring large prime numbers or the discrete logarithm problem, which are vulnerable to Shor's algorithm, which can be run efficiently on quantum computers.

On the other hand, STARKs typically offer a larger proof size and require interactive communication during the proving. The latter can be remediated by using a Fiat-Shamir transform discussed in the following sections along with a walk-through of a simple STARK

proof. To understand a zk-STARK proof, we need to describe three major parts of it, and those are problem definition and computational integrity conditions definition, problem arithmetisation and low-degree testing.

Problem Definition and Computational Integrity Statement

Let us consider a simple example of a sequence X of N non-negative integer elements $x_i, i \in \{0, 1, ..., N-1\}$. The prover wants to prove to the verifier that he knows a sequence containing only zeroes and ones. If we were to describe this condition with a polynomial, it would look like this:

$$x_i \cdot (x_i - 1) = x_i^2 - x_i = 0 \tag{3.1}$$

This is our computational integrity (CI) statement that we as a prover want to prove to the verifier without having to reveal any additional information about our sequence.

Problem Arithmetisation

The second step consists of transforming the problem into an algebraic realisation so that we can reason in the natural numbers. As with many cryptography problems, we will not work in the Euclidean space, but rather over a finite (Galois) field.

The trace is an evaluation of some polynomial f for some $E \subset F$ where $|E| = N$. We define a generator g of the cyclic subgroup G of multiplicative group $F_{/\{0\}}$, where F has a size of a prime number. The size of G is N. The prover then maps the elements of sequence X to the subgroup G, which are defined by the generator g:

$$f(g^i) = x_i, i \in \{0, 1, ..., N-1\} \tag{3.2}$$

The next step involves finding the polynomial f. In Section 2.8.2 we defined two methods for polynomial construction which were Lagrange interpolation and the Fast Fourier transform. In general, there exists an infinite number of polynomials that can satisfy given constraints, but only one of them is low-degree as discussed in Section 2.8.2.

So far we have defined mathematical constraints that hold if the statement is true, but for now the verifier could just evaluate the trace at each point and found out the original data, which would not help with succinctness nor zero-knowledge properties. That is why the next step is to evaluate the polynomial on a larger domain, which creates a Reed-Solomon correction code. This involves selecting L where $G \subset L \subseteq F_{/\{0\}}$, such that $f : L \to F$, and evaluating f on L.

By combining Equation 3.1 and Equation 3.2 we receive:

$$x_i^2 - x_i = 0 \implies f(g^i)^2 - f(g^i) = 0, i \in \{0, 1, ..., N-1\} \tag{3.3}$$

From this, the prover then derives a composition polynomial p, which is the final step of the arithmetisation process. The interactive protocol between the prover and the verifier consists of the prover first committing to p and f and then querying for values of p and f and checking if they fit the commitment and the required conditions. If a prover used a polynomial f' which is not low-degree, they would be able to cheat and prove false proofs. That is why low-degree testing is a part of the proving protocol.

Low-degree Testing

Low-degree testing uses *Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI)* protocol [7] to reduce the dimensionality of the problem by halving the degree of a polynomial that represents it in each iteration. After a certain number of iterations λ a constant is reached. There exists an upper bound on λ which decides whether the original polynomial was of low degree with a certain probability. This algorithm has a time complexity of $O(\log n)$ where n is the number of data points that our polynomial is describing.

Fiat-Shamir Transform

The interactive property of STARKs is not always beneficial and could lead to efficiency and scalability problems. Fortunately, there is a way to transform an interactive proof of knowledge to a non-interactive one using a technique known as the Fiat-Shamir transform, which utilises a cryptographic hash function to produce pseudo-random queries from the transcript of a protocol up to that point [10].

To visualise the Fiat-Shamir transform [11], let us consider a game of Sudoku played by the prover. Prover wants to convince the verifier that they have solved the game without revealing the solution. Both the prover and the verifier know the initial given numbers. A Sudoku solution can be validated by checking the following three types of constraints:

- Each row contains distinct digits.
- Each column contains distinct digits.
- Each 3×3 block contains distinct digits.

Since the Sudoku grid contains 9 rows, 9 columns, and 9 blocks, the total number of constraints to check is 27. For the sake of example, let us say that instead of writing the digits on paper, we solved the game by placing tokens numbered on one side with 1 to 9 on a grid, and we turned them face-down so that the verifier cannot see the numbers inside the grid (intial given numbers are visible). The verifier could only ask for the contents of a row, column, or block, to verify only one of the 27 constraints at a time (Figure 3.2). The prover would pick up all tokens from the selected row, column, or block, scramble them, and give them to the verifier to check if they are distinct. Afterwards, the tokens would be returned to the grid face-down. If the solution is valid, the condition will hold. If the solution is not valid, there is a chance that this particular constraint would still hold true. This could be repeated for multiple rounds and constraints to increase the certainty of catching a malicious prover.

This protocol requires interactivity since the prover is responding to verifier's queries for multiple rounds. We could turn this protocol into a non-interactive one using the Fiat-Shamir transformation (heuristic). This requires the prover to commit to the solution, so that they cannot change the solution in-between the querying rounds (Figure 3.3). This can be performed, for example, using a Merkle tree (Section 2.4.3). The prover creates a hash of each row, column, and block and joins the hashes into a Merkle tree and finally computes the Merkle root. Sharing the Merkle root with the verifier will prevent prover from changing the Sudoku solution during the querying, since before that, prover could easily create a solution that is invalid but satisfies the single-queried condition.

Fiat-Shamir transformation would consist of the following steps:

4	1	2	9	6	3	8	5	7
5	8	3	4	7	2	1	6	9
7	9	6	1	5	8	2	4	3
2	5	9	6	3	7	4	1	8
6	7	1	5	8	4	3	9	2
8	3	4	2	1	9	6	7	5
1	2	7	3	4	5	9	8	6
3	6	8	7	9	1	5	2	4
9	4	5	8	2	6	7	3	1

Figure 3.2: An example of a solved Sudoku, where the digits in bold correspond to the initial given numbers. In our example the tokens which are part of the solution are placed face down so that the verifier does not see them. The verifier can request to see the tokens of a row, a column or a 3×3 block. Prover will pick the nine tokens from the grid, scramble them and give them to the verifier, who will check if the tokens have distinct digits.

1. Prover would convert the Merkle root into a number between 0 and 26 using modular arithmetic (remember, this conversion is part of the protocol that is agreed upon before).
2. The number between 0 and 26 represents the condition to be queried.
3. We have now pseudo-randomly (but deterministically) produced a query without having to interact with the verifier.
4. We repeat this algorithm from step one until we have performed enough rounds. We will append the result of the query to the Merkle root and hash it before receiving a number between 0 and 26 in step one, otherwise we would receive the same query.

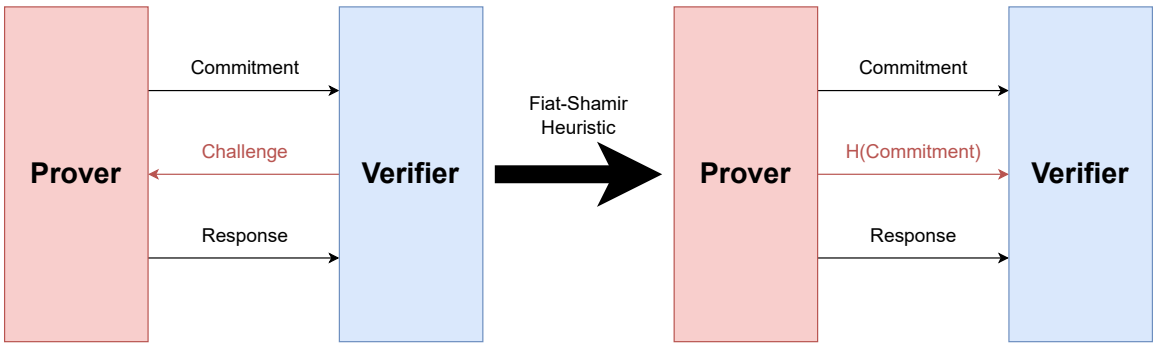

Figure 3.3: Fiat-Shamir heuristic (transformation) converts interactive proof to a non-interactive one. Function H represents a hashing function. Adapted from [29].

We were able to substitute the role of an interactive verifier using Merkle trees for commitments and modular arithmetic to select queries, thus converting an interactive proof into a non-interactive one.

3.1.4 Comparison of SNARKs and STARKs

Both STARKs and SNARKs are zero-knowledge protocols which allow one party to prove a statement to another party succinctly and without having to reveal any further information about the statement aside from the statement's truthfulness. One of the inherent disadvantages of SNARKs is the necessity of initial setups (trusted ceremonies), which pose a security risk – the initial key generation must take place within a secure environment, since the generation of keys produces data often referred to as *toxic waste* which can be used to generate false proofs [57]. The impact of this disadvantage is greatly reduced by the ability of keys to be generated using SMPC (Section 2.6) where the responsibility is shared among all participants in the setup. If at least one of the participants is honest and deletes their part of the toxic waste, then no false proof can be generated anymore. SNARKs are considered to be more efficient and faster compared to STARKs, while also benefiting from a smaller proof size and a shorter verification time [17]. Their differences are also summarised in Table 3.1.

	SNARKs	**STARKs**
Primary cryptographic mechanism	Elliptic curves	CHFs
Interactive	No	Yes
Require initial setup (ceremony)	Yes	No
Believed to be post-quantum secure	No	Yes

Table 3.1: Comparison of properties of two popular zero-knowledge protocols.

3.2 Cryptographic Accumulators

A cryptographic accumulator [42] is a data structure used for set membership tests that can be used as an alternative to search-based approaches – linear search in an unsorted list of elements (time complexity of $O(n)$), binary search in sorted list or search tree-based approaches (time complexity of $O(\log n)$) or hash tables (time complexity of $O(1)$ under certain conditions), where n is the size of the list.

Cryptographic accumulators use cryptographic primitives in order to achieve sublinear time complexity for set-membership operations. They also provide the benefit of allowing proof memberships without having to reveal the actual members of a set, which is a property used in privacy preserving applications.

3.2.1 Classification

Depending on whether the accumulators use symmetric or asymmetric cryptography, they are classified as symmetric or asymmetric. *Symmetric accumulators* are able to verify membership of an element without having to generate a witness after every inserted element. An example of a symmetric accumulator would be the Bloom filter (Section 3.2.2), which uses k hash functions to select which of the m bits should be set to one. Its false negative rate is zero, but the false positive rate grows with the number of accumulated elements.

Asymmetric accumulators are typically (but not always) built on asymmetric cryptography and require witness creation and update for dynamic verification of set membership. An example of an asymmetric accumulator would be the RSA accumulator (Section 3.2.4) that uses exponentiation and modular arithmetic.

Certain applications do not require the accumulator to perform both membership and non-membership tests. Accumulators that can only provide membership-proof for the elements they have accumulated are called *positive accumulators*. On the other hand, an accumulator able to perform only non-membership proof for non-accumulated elements is called *negative accumulator*. If an accumulator is both positive and negative, it is referred to as a *universal accumulator*.

If an input of an accumulator can change in time, it is called *dynamic accumulator*, otherwise it is called *static accumulator*. *Additive accumulators* can only add new elements, and *subtractive accumulators* can only remove input elements.

3.2.2 Bloom Filters

Bloom filter (Figure 3.4) [51] is a probabilistic data structure that allows a set to be saved in a space-efficient way. This structure consists of an m bit array and k hash functions with output in the range $[0, m-1]$. Initially (with the filter containing zero items), all bits are set to zero. The structure allows for set membership queries which can result in false positives but cannot result in false negatives. In other words, the two possible results of the query are "definitely not in set" or "possibly in set". The false positive rate increases with the number of accumulated elements.

Insertion of an element involves applying all hash functions to the input and receiving k indices into the bit array. The bits in the places of the k indices are set to one[1]. It can be easily seen that the insertion operation is idempotent, as would be expected when working with sets.

Querying for an element is performed by applying all hash functions to the input element, receiving k indices, and checking if all bits in the indices are set to one. If at least one of the indexed bits is zero, then the element is definitely not in the set. All bits containing ones mean a high probability that the element is in the set. We cannot say with complete certainty that an element is in a set, since two situations could have occurred:

- There is an element already in the set with a hash collision for all k hash functions occurring at once (if we consider the output distribution of all hash functions to be uniform and independent of other hash functions, the probability of two elements colliding in all hash functions is $1/m^k$).
- Multiple previously added elements caused all bits of the queried element to be set to one. We can consider this extreme case – if the distribution of all hash functions is uniform along the whole bit field, and we keep inserting distinct elements, eventually all the bits in the filter will be set to one, thus querying for any element will yield a positive result.

Statistical dependence of the false positive rate FPR on k and m and the number of accumulated elements n is [42]:

[1] Note that sometimes two or more hash functions may yield the same result, and therefore set the same bit to one twice.

$$FPR = (1 - [1 - \frac{1}{m}]^{kn})^k \approx (1 - e^{\frac{-kn}{m}})^k \tag{3.4}$$

Bloom filters are a very important tool for storage optimisation, used (not only) in blockchains. For example, in the Bitcoin protocol, Bloom filters are part of the block header and are used to encode transactions within a block[2] – a user who searches for a transaction on a Bitcoin blockchain first queries the filter to see if the transaction is possibly within the block checked currently. Although not perfect, thanks to Bloom filters the user has to inspect only a small fraction of blocks.

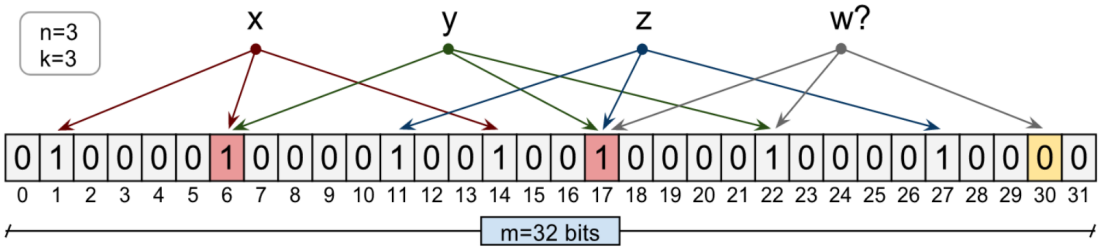

Figure 3.4: Overview of a Bloom filter with three hash functions and their indices for various elements. If we were to insert x and w into the Bloom filter and then query for y, we would get a positive result, even though y is not in the set. Retrieved from [51].

3.2.3 Cuckoo Filters

Cuckoo filters are made up of m buckets and use two hash functions to map and item to two indices. The buckets store a fingerprint of an item, typically a hash. The items fingerprint can be located at either of the two indices given by the lookup function. If the buckets in both indices are occupied, the fingerprints are rearranged (Figure 3.5) so that each fingerprint is located in one of their assigned indices. If they cannot be rearranged, the two lookup functions are altered and the structure is regenerated. In contrast to Bloom filters, cuckoo filters also allow the deletion of set items.

3.2.4 RSA Accumulator

An RSA accumulator [9] is an asymmetric accumulator that uses modular exponentiation similar to the RSA cryptosystem. In a simple RSA accumulator, the modulo value N is calculated as the product of two large prime numbers p and q. To accumulate (insert) a value x, the new value acc_n of the accumulator is calculated as follows:

$$acc_n = acc_{n-1}^x \mod N \tag{3.5}$$

where acc_{n-1} is the previous value of the accumulator. Unfortunately, RSA accumulators are collision-free only when accumulating prime numbers[3] unless we map input elements onto primes.

[2]For details, see BIP-37 (https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki).

[3]When accumulating the numbers 2 and 6 we would receive the same accumulator value as if we accumulated 3 and 4.

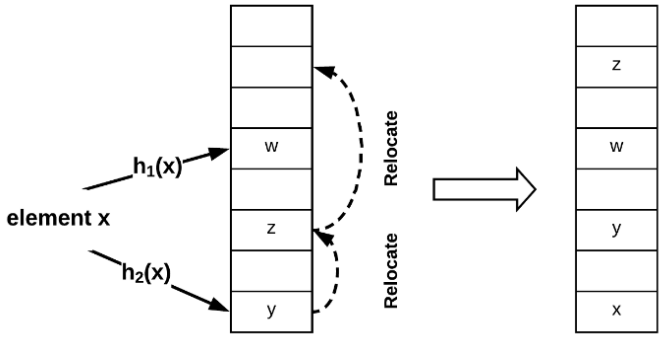

Figure 3.5: Insertion of an element x into a cuckoo filter. Since none of the buckets were empty, the existing items had to be rearranged. Retrieved from [42].

3.2.5 Merkle Tree Accumulator

A Merkle tree (Section 2.4.3) structure can be used as an accumulator [28] by treating the leaves as the elements that are being accumulated. The root of the tree serves as the accumulator value, which means that the value is constant in size regardless of the number of accumulated elements. This would be classified as an asymmetric accumulator because a set member needs a witness for membership proof, although does not use any asymmetric cryptographic primitives. The witness corresponds to the Merkle path of the queried element, and its size is logarithmic to the number of accumulated elements.

3.2.6 Evaluation and Applications

Cryptographic accumulators are data structures that provide efficient aggregation and verification of a large set of elements. Their uses can be divided into two categories – storage efficiency applications and privacy preserving applications.

The former category includes applications such as the use of Bloom filters by Bitcoin to simplify the lookup of a transaction inside the blockchain. These applications utilise the data compression ability of the accumulator without having additional security requirements. Since the accumulator uses fewer bits to represent the accumulated element than bits used to represent the original data, it creates the possibility of collisions, leading to being able to query for elements that were not accumulated (shown in Figure 3.4). The false positive probability is proportional to the number of accumulated elements and inversely proportional to the size of the accumulator (Figure 3.6).

Their usage in the privacy-preserving category requires balancing trade-offs between security and performance savings as required by their intended application. To prevent manipulation of the accumulated data, the data structure must ensure that it is infeasible to find two distinct sets of elements that produce the same accumulator value. The membership or non-membership proofs have to be unlinkable, so that an adversary cannot deduce any information about the contained elements even if multiple proofs are observed. Thanks to these properties and especially the inability of extracting the accumulated elements, cryptographic accumulators are an integral part of many privacy-preserving schemes including, but not limited to anonymous credential systems [3, 15, 34], group signatures [15], ring signatures [41] and fail-stop signatures [6].

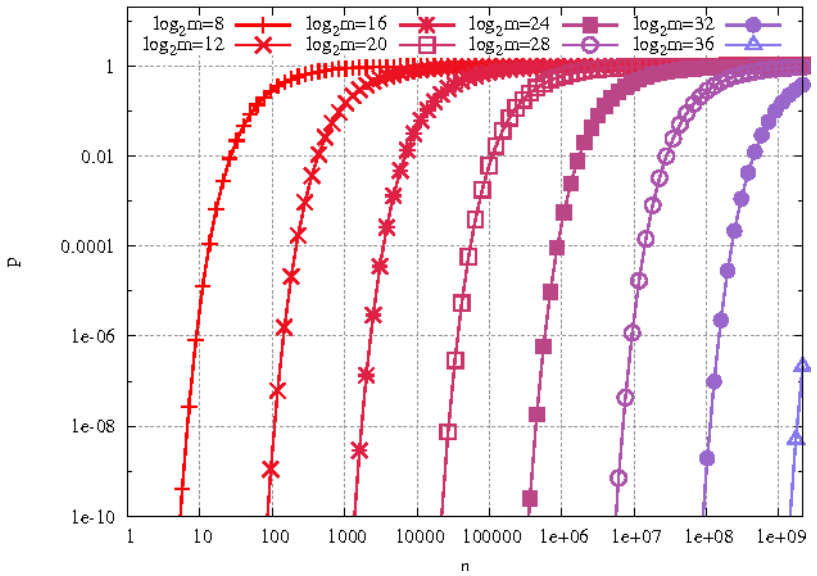

Figure 3.6: The false positive probability p as a function of number of elements n and the size m of a Bloom filter with optimally selected number of hashing functions k. Retrieved from [27].

Chapter 4

Existing Technologies and Applications

This chapter discusses existing technologies utilising storage or computation offloading using cryptographic techniques mentioned in the previous chapters, with heavy emphasis on zero-knowledge protocols.

4.1 Trusted Execution Environments

A Trusted Execution Environment (TEE) [38] is a secure area of CPU and memory protected from the rest of the system by a layer of encryption, which protects the area (the so-called *secure enclave*) from unauthorised access and manipulation. This technology prevents the reading of data outside the TEE by unauthorised parties, achieving confidentiality. Code integrity provides guarantees that code inside TEE has not been tampered with by unauthorised parties. Some TEEs provide a capability called *attestation*, which can be used to remotely prove that a particular program is running inside a secure enclave.

The use of TEEs in blockchain can provide an extra layer of security for critical tasks such as signing transactions or can improve the scaling of blockchain by offloading some computations to the secure enclave. Thanks to their properties, TEEs can functionally replace zero-knowledge proofs, fully homomorphic encryption, or secure multiparty computation.

The most notable examples of TEE platforms are the Intel SGX (Software Guard Extensions) technology, which is implemented on the hardware level, and AWS Nitro, which is a software TEE.

4.2 Bitcoin Lightning Network

The Bitcoin network can only handle about seven transactions per second, which is nowhere comparable to online merchants such as Visa, which could handle 65 000 transactions as of August 2017[1]. This leads to high transaction fees and slowness of the network with regard to transaction confirmation.

The obvious solution would be to either increase the block size to accommodate more transactions inside each block. This is a controversial topic because increasing the block

[1] According to https://www.visa.co.uk/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf.

size would mean higher storage requirements (even without this, the Bitcoin blockchain is now more than half a terabyte large), which means that fewer users could afford setting up a node, leading to higher centralisation. Although this did not stop the inception of Bitcoin Cash – a hard fork of Bitcoin that changed the block size from Bitcoin's 1 MB to 32 MB.

Another obvious solution would be to decrease the block time – this would lead to blocks being created faster, increased throughput, and decreased transaction latency of the network. The problem with this solution is that the new blocks might not be propagated quickly enough throughout the network and might lead to unwanted forking. Thus, multiple more involved solutions were proposed to speed up the network, one of which is the Lightning network.

Lightning network [46] is a layer-2 protocol built on top of Bitcoin. It provides a way between two parties to transact securely repeatedly without having to write each transaction to the blockchain. The only interactions necessary with the Bitcoin blockchain are the opening and closing of channels.

Let us say that two parties Alice and Bob want to exchange Bitcoins regularly. To save transaction fees and speed up exchange, Alice creates a Lightning network channel and deposits 1 BTC into a multi-signature wallet that represents the channel. A channel balance sheet is generated that tracks the amount of currency within the channel, with Alice having 1 BTC that she deposited. Changes in this balance sheet are not tracked on the blockchain, but each transaction within the channel must be signed by both parties. Alice and Bob each have a signed copy of the most recent balance sheet and can each whenever decide to close the channel and withdraw the Bitcoin amount written in the most recent balance sheet from the multi-signature wallet. The whole interaction is summarised in Figure 4.1.

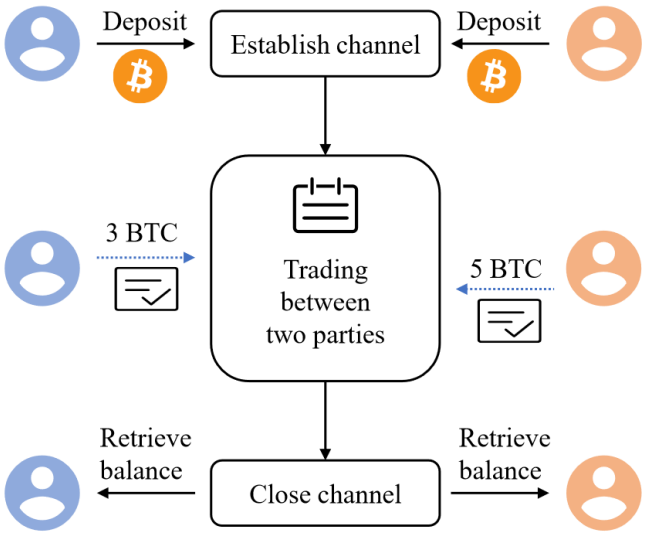

Figure 4.1: Summary of how Lighting network is used. The only necessary interaction with the blockchain is in steps "Establish channel" and "Close channel". Retrieved from [56].

Due to the transitive nature of channels, two exchanging parties do not always have to create a channel directly, but could utilise a chain of users within the Lightning network mesh (Figure 4.2).

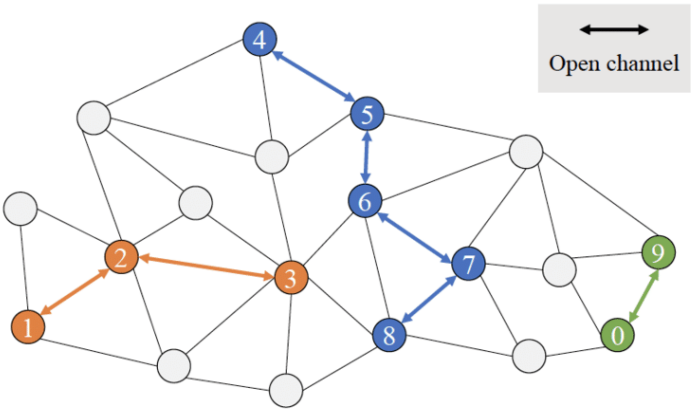

Figure 4.2: Example topology of Lightning network. The image shows one direct (in green) and two indirect connections (in yellow and blue) of users using channels. Retrieved from [56].

The only two operations that write to the blockchain and can be examined publicly are the opening of a channel together with the initial charge amount, multi-signature wallet address, and depositor address and closing of the channel with settlement amounts.

In a 2021 Bitcoin update nicknamed *Taproot*, the network switched to Schnorr signatures, which make multi-signature transactions indistinguishable from simple transactions in terms of on-chain footprint [16].

Lightning network uses watchtower, which is a service that detects and prevents fraud in the network. If a node breached a channel (using an out-of-date commitment transaction), they would receive a penalty.

Thanks to the Lightning network, two parties can exchange Bitcoins faster and with much smaller fees. With it, the calculations within the channel need to be performed only on two nodes instead of the whole Bitcoin network, which reduces the Bitcoin network load. The decrease in number of transactions is hard to empirically express, since the channel transactions are not public, but even if we consider that each Lightning channel sees ten transactions before closing, the decrease of on-chain transactions is fivefold.

4.3 ZoKrates

ZoKrates [21, 57] is an open-source library written in Rust, providing tools for working with zk-SNARKS. This library includes support for arithmetic circuit compilation and the generation of zk-SNARKs. Generated proofs can be verified directly using the library or by exporting a verification smart contract that can be deployed on the Ethereum network.

ZoKrates provides a domain-specific high-level language for describing arithmetic circuits. Its syntax can be observed by inspecting Figure 4.3 and Figure 4.4.

Here are listed individual ZoKrates commands in order that they are typically used to generate a proof:

- A ZoKrates source file (typically denoted with `.zok` extension) is compiled into an arithmetic circuit using `zokrates compile` command.
- The command `zokrates compute-witness` along with the proof parameters is executed by the prover to create a witness file.

- The verifier creates a pair of keys using `zokrates setup` and sends the proving key to the prover. It is important that this step is run by the verifier as the prover could abuse this procedure and generate false proofs.
- The prover generates the proof using `zokrates generate-proof` and sends the proof to the verifier for verification.
- The correctness of the proof can be checked using `zokrates verify`.

```
def main(
    private field factor1,
    private field factor2,
    public field product
) -> bool {
    assert (factor1 * factor2 == product);

    return true;
}
```

Figure 4.3: ZoKrates source code for a program which can be used by a prover to prove a knowledge of two secret factors which make up public product.

```
def main(private u32 a) -> bool {
    u32 mut b = a;

    for u32 i in 0..5 {
        b = if b % 2 == 0 { b / 2 } else { b * 3 + 1 };

        assert (b != 1);
    }

    return true;
}
```

Figure 4.4: Source code for a ZoKrates program to check if a prover knows a number that would take at least 5 steps to converge to the number 1 given the rules of Collatz conjecture without revealing that number to the verifier.

4.3.1 Trusted Setup Using SMPC

As mentioned in Section 3.1.4, the trusted setup procedure produces data that can be used to generate false proofs by the party performing the trusted setup. This is problematic in the trustless multi-party environment, since trust has to be put into the hands of a single party. Thankfully, ZoKrates provides commands for performing the trusted setup using a secure multiparty computation protocol (SMPC, Section 2.6).

The process consists of two phases. The first phase of the setup is called *Perpetual Powers of Tau Ceremony* and includes using `phase2-bn254`[2] on the latest response from the *Perpetual Powers of Tau* repository[3] to produce initial SMPC parameters. In the second phase, each of the participants can contribute their own randomness to the key setup with the `zokrates mpc contribute` command. There only needs to be a single honest node, which deletes its randomness afterwards, to make the setup secure, so that false proofs cannot be generated. The ceremony is finalised by applying a random beacon with `zokrates mpc beacon` to get the final SMPC parameters, from which a key pair can be exported using `zokrates mpc export`. At any point in the second phase, any participant can verify individual contributions by running the command `zokrates mpc verify`.

4.4 Zk-Bridge

With the proliferation of blockchain technologies, the heterogeneity of various applications grows. To build a functioning solution allowing multi-chain communication between applications, there needs to be a bridge that facilitates cross-blockchain transfer of messages or funds.

Zk-bridges [54] offer a decentralised solution which achieves practical performance with reasonable fees using zk-SNARKS. Other existing solutions [55, 44] suffer either poor performance or rely on central parties.

Zk-SNARKs are leveraged to prove that a certain event (message, monetary transaction, condition fulfilment) took place on the sender chain. The updater contract on the receiver chain (Figure 4.5) verifies the SNARKs and after successful verification performs the state change in the receiver network. This technology can be used, for example, to implement decentralised exchanges that enable trustless swaps of tokens on different blockchains.

In between these two chains, there is a relay network that generates zk-SNARKs based on the sender chain state. The relay network needs only a single honest node to function. Bad actors inside the relay network are disincentivized since invoking the updater contract costs fees on the receiver chain. The usage of an off-chain relay network is necessary since the zk-SNARK proof generation is too computationally expensive to be performed on chain as opposed to the proof verification, which is verified in the receiver chain smart contract when using zk-bridges. The setup assumes that both linked blockchains are live and consistent, that there is at least one honest node in the relay network, and that the zk-SNARK used is sound.

This solution utilises recursive proof generation technique to achieve a short proof generation time, but requires the receiver chain to support smart contracts. However, a solution has been proposed to use Bitcoin (which does not support smart contracts and has a very limited scripting system) as a receiver chain as well [45].

Another solution for cross-chain message relaying using zk-SNARKs is zkRelay [53] which shared many commonalities with zk-Bridge.

4.5 Mina Protocol

Mina Protocol [13, 19, 24] is declaring itself as the "world's lightest blockchain" thanks to its ability to capture the blockchain state in a verifiable snapshot of a constant size.

[2] Available from https://github.com/kobigurk/phase2-bn254.
[3] Available from https://github.com/weijiekoh/perpetualpowersoftau.

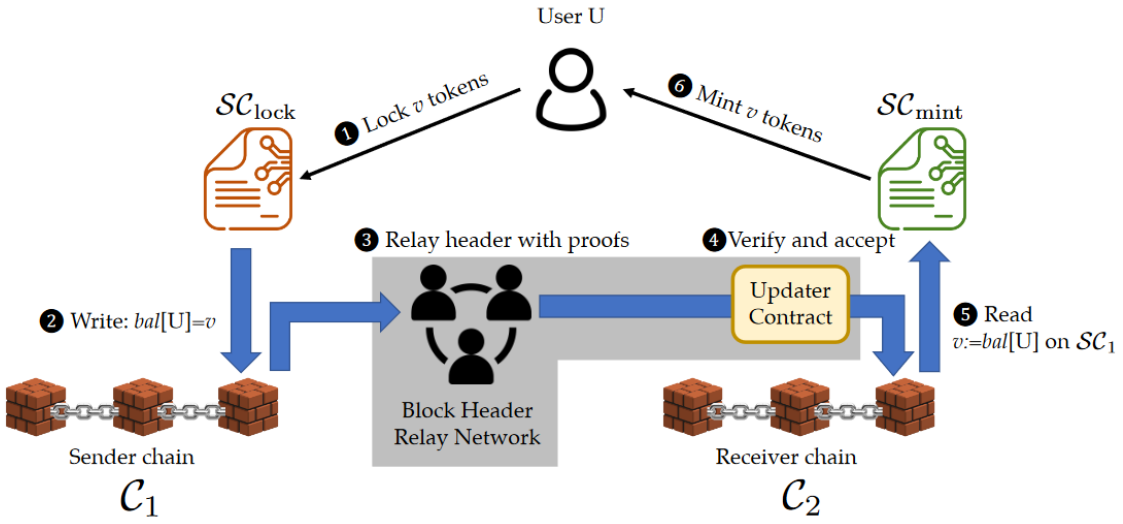

Figure 4.5: Zk-bridge example facilitating token transfer across two chains. The notation of sender and receiver chain is used for this particular example – the bridges can be used symmetrically in both directions. Retrieved from [54].

Unlike Bitcoin's blockchain, which is over 570 GB as of May 2024[4] and evergrowing with every block produced, the Mina Protocol blockchain size is tiny 22 kB and does not grow over time. Obviously, 22 kB is not even close to being able to capture all transactions and events on the blockchain. However, the Mina protocol classifies the blockchain as a structure that captures the data required to verify if the current state is accurate and valid in a trustless manner. Unlike Bitcoin or Ethereum, which require a user to download the whole blockchain to verify the state in a trustless manner[5], Mina uses a SNARK proof to capture the correctness of a state without having to store the entire data source.

Mina leverages recursive SNARKs to ensure succinctness at any point in time as the number of blocks increases. This means that as the blockchain grows, a new SNARK proof must be generated that validates both the new blocks and the existing SNARK proof (Figure 4.6). This type of SNARK where a SNARK proof attests to the verifiability of another SNARK proof is called *incrementally-computable SNARK* [52].

The consensus mechanism Mina uses is called *Ouroboros Samasika* and has the characteristics of proof of stake protocol where the probability of being selected as the block producer is proportional to the amount staked relative to other nodes. However, unlike the proof of stake protocol described in Section 2.5, Mina does not lock funds, nor does it slash funds; it only stops distributing rewards to nodes that are offline or behave maliciously.

4.6 ZkLLVM

ZkLLVM [40] is a compiler designed to convert programs in high-level programming languages into inputs for provable computation protocols, particularly zero-knowledge proof

[4] As reported in https://ycharts.com/indicators/bitcoin_blockchain_size.

[5] Bitcoin has a concept of Simplified Payment Verification (SPV) clients which only download the block headers to drastically decrease the amount of data downloaded. But this means that SPV nodes need to trust the source of the data, since they do not verify block transactions themselves.

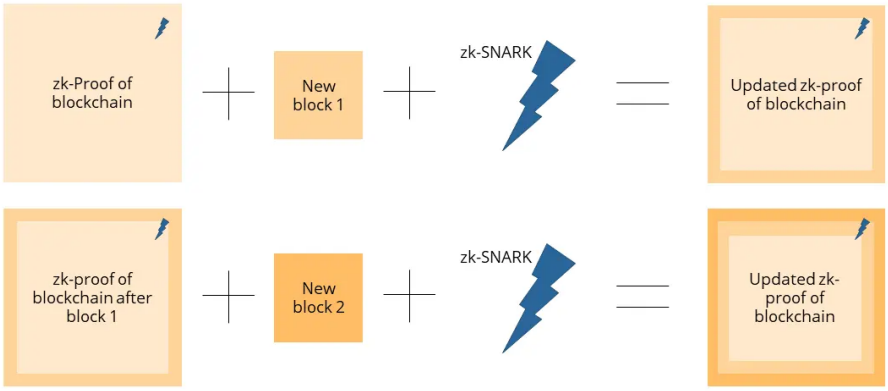

Figure 4.6: Recursive nature of SNARKs used in Mina protocol to produce constant size proofs. Adapted from [24].

systems, by leveraging the LLVM infrastructure. It allows developers to create applications in a compatible language (C, C++, Rust, and more). These are compiled using a modified version of the Clang compiler, which outputs an intermediate representation of the circuit. The compiled circuits are then used to generate in-EVM applications for on-chain proof verification.

In 2023 the foundation responsible for zkLLVM also introduced Proof Market [30] whose goal is to match the developers aiming to implement proof-based applications without access to expensive infrastructure and suppliers of computational resources. This market implements a matching engine that uses parameters of cost, order timeout, and generation time to find the best proof generation supplier for a given order. The proof producer who does not submit a proof in a promised time interval could be subject to penalties, restrictions, and a decrease in their ranking which determines the likelihood of a future request match.

4.7 Marlin

Marlin [35] is a solution for delegating complex computations to the decentralised cloud utilising trusted execution environments (TEEs) and zero-knowledge based co-processors. It enables smart contract-based protocols, web, mobile clients, and enterprises to securely rent compute instances via a decentralised network of globally distributed nodes, which allows for auto-scaling and fault tolerance.

Oyster is the Marlin subnetwork that specialises in TEE-based computation that ensures the integrity and confidentiality of the computation carried out on the host machine. The other subnetwork called *Kalypso* is focused on the usage of zero-knowledge proofs for efficient verification of the computation correctness.

Each of the subnetworks is a permissionless network, which can be joined by any node provided the subnetwork requirements are fulfilled. Thanks to the open-source nature of the project, the public can develop new specialised subnetworks.

Chapter 5

Design

Based on the techniques described in the previous chapters, we propose a novel approach to offloading computations and storage off of blockchain – SNARK marketplace with proof of useful work (PoUW) [5] incentive mechanism where clients can request a SNARK proof generation by providing either ZoKrates source code or by referencing an existing arithmetic circuit known to a network along with proof parameters. Since the proof requester shares both their public and private parameters with the rest of the network, this approach foregoes the zero-knowledge aspect of zk-SNARKs and ZoKrates and focuses only on plain SNARKs. Proof generation is encapsulated within proof transactions. The network also uses coin transactions of the native currency, which facilitates block rewards and fees for transaction confirmation and proof generation.

5.1 Incentive Scheme

Full nodes can receive a native currency reward by producing a block which contains generated SNARK proofs and coin transactions. The reward consists of three components:

- **Proof generation fees** – Sum of the fees paid by the clients requesting proofs. The fee paid for a particular proof generation is proportional to the complexity of the proof.
- **Coin transaction fees** – Sum of the fees paid by the clients who request coin transactions.
- **Block reward** – Newly minted amount of native currency. This reward increases the total supply of the currency, but will decay until the maximum supply is reached, after which the reward will become zero, similarly to Bitcoin.

5.2 Transactions

Coin transactions facilitate the transfer of native currency between accounts. They consist of the following fields:

- **Transaction ID** – Unique identifier of a transaction.
- **Sender address**.

- **Recipient address**.
- **Amount**.
- **Signature** – Signature of the transaction metadata by the sender to authenticate and protect the integrity of the metadata.

Proof transactions wrap the proof generation tasks including the information about the corresponding arithmetic circuit and parameters. A proof transaction has the following fields:

- **Transaction ID** – Unique identifier of a transaction.
- **Requester address**.
- **Requester signature** – Since requesting a signature deducts a fee from the requester account, the proof transaction has to be authenticated.
- **Circuit hash** – Uniquely identifying a circuit within the network.
- **Public parameters** – Parameters of the circuits necessary to generate and verify the proof.

The list of transactions contained within a block is included inside the block body with additional hash inside a block header protecting the integrity of the transactions.

5.3 Block Construction

The block construction consists of selecting the proof transactions according to the current network difficulty parameter. The difficulty threshold is a network-controlled parameter that is calculated based on the rolling average of block creation time across fixed amount of latest blocks. Recalculation of difficulty does not happen after each and every block, but rather after a retargeting period to smooth out short-term fluctuations in the mining power.

The contribution of the proof to reaching the difficulty threshold depends on the complexity of the proof, therefore, one proof with a complex arithmetic circuit can contribute more than multiple simpler proofs, thus fairly compensating miners based on the work (computation) performed. The complexity of a circuit is defined by the number of constraints within it.

After the proof and coin transactions have been selected and verified, the work phase begins where each of the selected proofs has to be generated (Figure 5.1). Verification of coin transactions is simple enough – it consists of signature verification, checking whether the sender has enough funds, and updating the state tree. SNARK generation is computationally difficult, which is why it is used as the work in the proof of work consensus protocol. The work performed here is useful even outside of the consensus protocol, unlike Bitcoin, which is why this consensus protocol is referred to as proof of useful work.

To protect the integrity of a block and to prevent a malicious node from modifying the finished block, each of the generated SNARKs has a special last parameter where the digest of the currently generated block is placed. With this, the block integrity is saved inside each and every piece of work performed, making the proofs tied to that block and unable to be removed and included inside a different block.

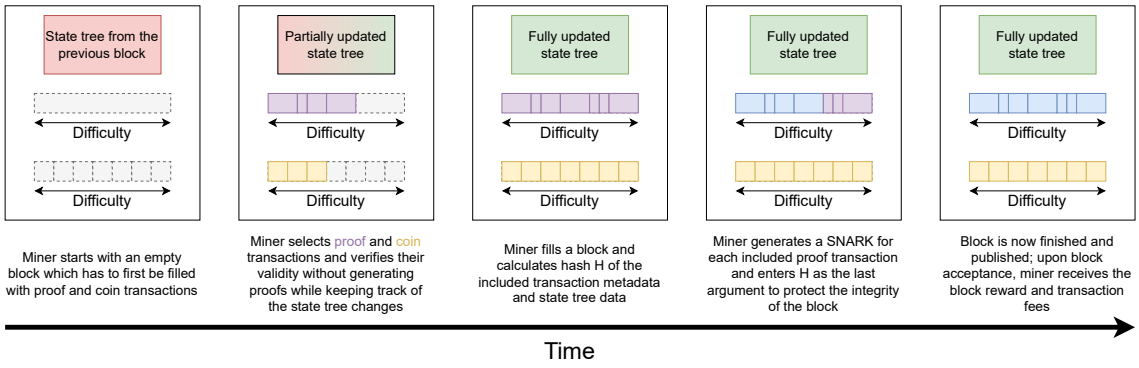

Figure 5.1: Lifecycle of a block being mined. Miner has to first perform useful work by generating enough proofs to cross the difficulty threshold.

If we were not to protect the integrity of a block inside each generated proof, then a malicious node could extract the generated proofs from freshly broadcasted block and use them to produce their own block while preventing acceptance of the original block, by either delaying the broadcast or even not broadcasting the original block further.

Another solution that seems possible at first is the inclusion of only miner identification within a proof. This would prevent any other miner from stealing a published proof, but would not prevent a malicious node from producing multiple different versions of a block with the same proofs. Thus, this solution would be incomplete.

5.4 Block Verification

All nodes participating in consensus have to confirm all received blocks. Let us assume that the preceding block B_{old} has already been verified and B_{new} is currently being verified. Verification of B_{new} consists of the following steps:

1. **Header Verification**
   - $B_{new}.serial\_id == B_{old}.serial\_id + 1$
   - $current\_time > B_{new}.timestamp > B_{old}.timestamp$
   - Difficulty rolling average calculation and verification.
   - $B_{new}.previous\_block\_hash == B_{old}.current\_block\_hash$
2. **Individual Proof Transaction Verification**
   - Check each proof validity using ZoKrates. SNARK verification step is much cheaper in terms of computational resources in contrast to proof generation.
   - Check each proof complexity. Verify that the sum of the proofs complexities inside the block reaches the current difficulty threshold.
   - Check if the proofs last parameter corresponds to the block metadata digest.
   - Apply fee transfer to the state tree.
3. **Individual Coin Transaction Verification**
   - Check if sender funds are sufficient.

- Apply transferred funds to the state tree.
- Apply transfer fee to the state tree.

4. **State Tree and Block Integrity Verification**
   - Check if calculated state tree hash and transaction lists hashes matches the header parameter.
   - Check if current block hash is correct.

5.5 Circuit Storage

Our solution uses a subnetwork of nodes that manage the registered arithmetic circuits. The subnetwork provides functions for decentralised registration and retrieval of arithmetic circuits along with the corresponding proving and verification keys, information on the circuit complexity, and the content of the original ZoKrates source code file. The circuit subnetwork consists of nodes that stake a sufficient amount of native currency. In the event of node misbehaviour or inactivity, the stake will be slashed.

The registration is initiated by a node that was unable to find a sufficient arithmetic circuit among the already registered circuits. This node sends a request to the circuit subnetwork and pays a fee. The fee is used as a mechanism to incentivise the reusing of existing circuits when possible. Part of the fee is divided among the subnetwork nodes, and part of it is burnt to prevent subnetwork nodes from registering circuits themselves for free. The registration request contains a ZoKrates source code file. Each node in the subnetwork will compile the ZoKrates source file and will contribute a piece of randomness to the trusted ceremony using ZoKrates SMPC (Section 4.3.1).

The use of SMPC allows the nodes to generate the proof key pair in such a way that only one of the nodes needs to be honest and delete their randomness after the ceremony is over to retain the network security. If all nodes in the subnetwork were malicious, they could share their randomness and use it to generate false proofs. After the registration has been completed, the circuit hash can now be used inside a proof request along with custom parameters by any party.

The registered circuits are uniquely identified within the network by their hash. There are multiple ways to generate the hash – we could generate the hash from raw ZoKrates source file, optimised source file (with removed comments, whitespace characters, etc.), binary circuit file or R1CS file. Generating a hash from the ZoKrates file has the benefit of the proof requester not having to compile the circuit locally to verify the correspondence of the fetched ZoKrates file and the hash. On the other hand, using circuit or R1CS hash would enable reusing circuits without having to depend on the program details that were compiled away, such as variable names.

The circuit subnetwork allows retrieval of data which is necessary for secure generation and verification of SNARKs within the network. The circuit database is write-only, meaning that registered circuits are immutable, allowing clients to cache the circuit data without ever having to invalidate the cached data.

5.6 Internode Communication

Blockchains typically rely on peer-to-peer (P2P) networks to exchange information while avoiding centralisation. Our network also uses the P2P paradigm, which means that each

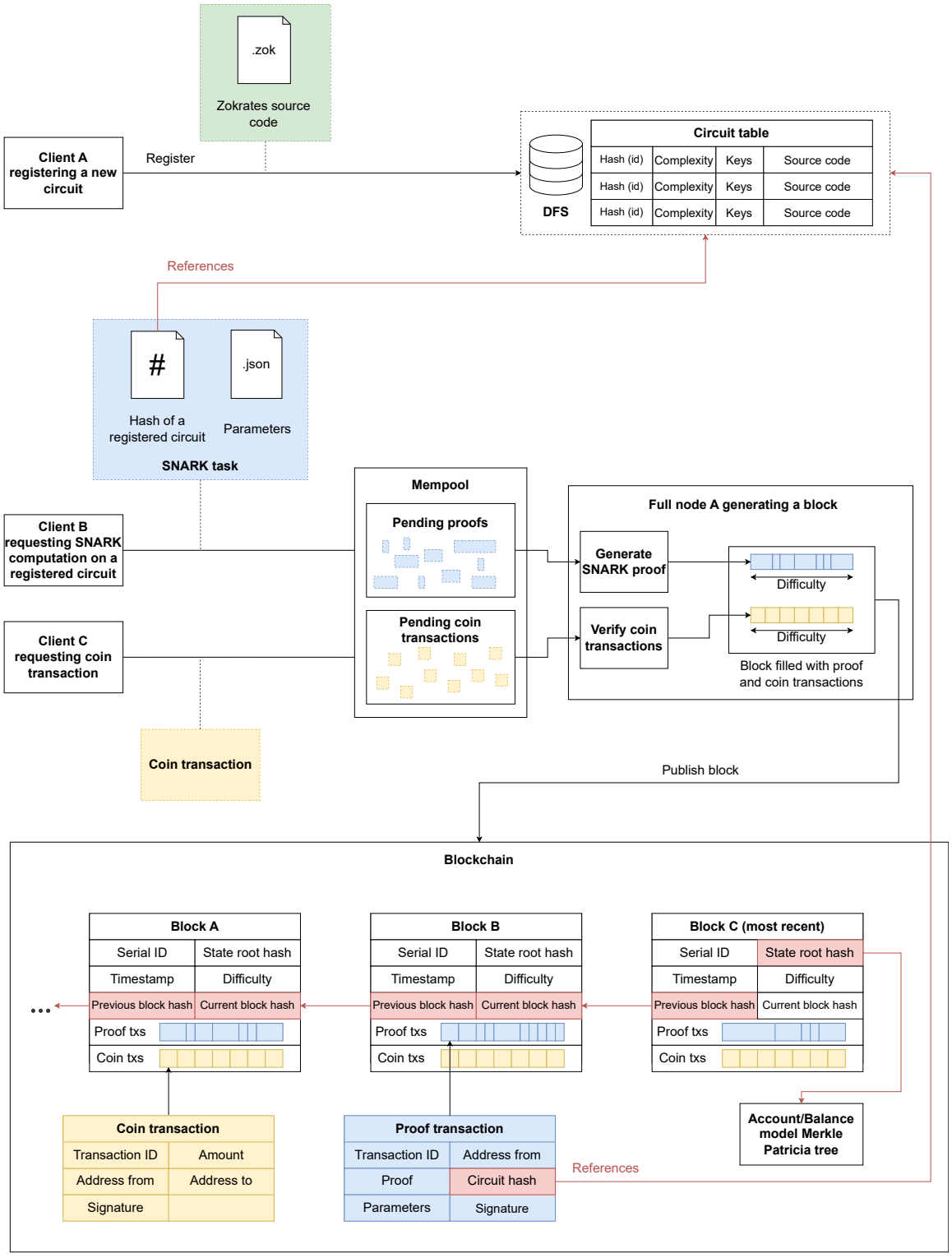

Figure 5.2: Diagram of a complete journey of transactions from creation to inclusion inside blocks. Clients can request either a coin transaction or a proof transaction on registered circuit. These requests will be placed inside two pools of pending transactions. Full nodes will pick transactions from the pools, verify them, and include them inside blocks. Finished blocks will be broadcast into the network and included inside the blockchain upon verification by the peer nodes.

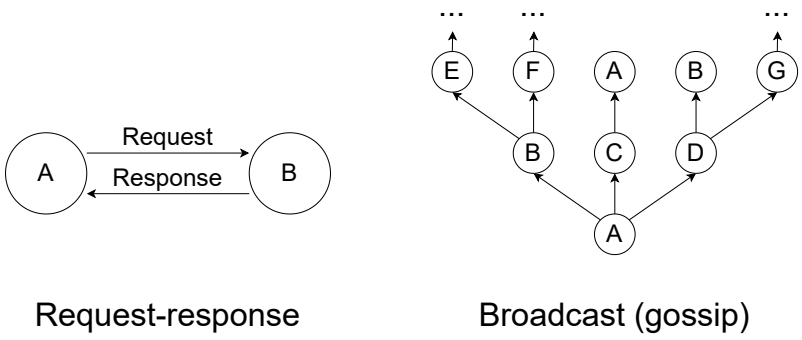

Figure 5.3: Two types of communications within the network. The circles represent the nodes within a network. In the broadcast type we can see what happens when there is a loop in the transitive closure of the peer relation – the nodes A and B occur in the network twice, but as we can see they do not transmit the message on the second receipt of the message.

node has to keep track of its peers so that it can be synchronised with the rest of the network. The number of peers has to be large enough to avoid a single or colluding group of nodes that have become malicious to influence the network in a negative way, for example by broadcasting invalid transactions or withholding new broadcasted blocks.

Each node curates its own statically defined list of seed nodes which are contacted upon node initiation and which recursively provide lists of their own peers, a list of pending transactions, and are willing to send information about blocks that have been mined since the initiated node was last online.

Communication between two nodes can have two types (Figure 5.3) – request-response and broadcast. Request-response are self-explanatory – a node is missing information and requests it from one of its peers, ideally from the one with the highest reputation (Section 5.10), who will answer with a response. Publishing a new pending transaction or a new block uses the broadcast communication type in which a starting node broadcasts a message to all of its peers and all of the peers will broadcast the message further recursively[1]. This is why the second communication type is sometimes referred to as a *gossip protocol* in the context of P2P networks.

5.7 Storing User Account Data

User account data for the current block is stored using the Account/Balance model (inspired by the Ethereum network [22]) where a Merkle-Patricia trie (Section 2.4.4) is used to store the mapping of addresses to coin balances. The root hash of the current version of the Merkle-Patricia trie for the corresponding block is saved in the block header. Current network implementation only needs to store the mapping of account address to balance, but Merkle-Patricia tries are extensible and can support storage of application data as well, including smart contract data.

[1]Unless the sending node has a low reputation or the message has already been broadcast by the current node. The latter condition is in place to prevent infinite message transmissions with loops in the network topology.

5.8 Proofs

Since writing custom SNARK generation and verification code would be very difficult and prone to security problems, ZoKrates (Section 4.3) was selected as the library that would provide means of manipulation with arithmetic circuits and proofs. ZoKrates is a tool that started its development in 2019 and has been incorporated into many existing blockchain ecosystems, such as zkRelay [53], and has proven to be robust and reliable. ZoKrates is easy to integrate into the rest of the SNARK marketplace as it is packaged into a CLI application and is well documented.

Another benefit of using ZoKrates is the ability to export a verifier smart contract that is compatible with the Ethereum network, which would allow the integration of the SNARK marketplace with Ethereum.

Considered were also alternatives, namely Circom, SnarkyJS and libsnark, but they were inferior in the criteria mentioned above.

5.9 Keys and Identifiers

Intergrity and authentication on the network is facilitated by asymmetric cryptography, in particular the ECDSA algorithm on the SECP256k1 curve (Figure 5.4). User accounts are identified by the public key that is stored in compressed SEC1 format, which has 33 bytes or 66 hexadecimal characters. Private keys, which are used to identify the user within the client application and to sign transactions, are saved in PEM format.

The SECP256k1 curve was selected due to its widespread adoption among cryptocurrencies, especially two of the most popular ones – Bitcoin and Ethereum. This curve is also classified as a *Koblitz curve*. This category of elliptic curves has performance and security benefits compared to regular elliptic curves [12].

For hashing, the SHA-256 function was selected, which produces 32 bytes long hashes. Hashing is used to generate the following fields – transaction identifier, block hash, state root hash, and hash of transactions within a block.

5.10 Reputation System

The proposed network bears the properties of anonymity and decentralisation which present challenges in establishing trust and ensuring reliable functionality of the network. This necessitates the creation of mechanisms that incentivise cooperation according to the rules to mitigate free-riding and other forms of undesirable behaviour. Such networks often use reputation systems where the activity of individual nodes within the network is tracked and a score is assigned depending on their behaviour and contribution to the network.

In this particular proof of concept, a very simple reputation system is introduced – each node assigns integer score between -10 and 10 to each of their peers. If their peer sends them a message which is correct and desirable (gossiped message or a response to a request), their reputation is incremented by 1. On the other hand, if an invalid, undesired, or malformed message is received from a peer, their reputation is decremented by 1. Failure to respond to a request (timing out) will also yield a decrease in reputation. If peer's reputation reaches -10 then they are considered removed from the list of peers, and their messages are rejected. If a node needs to make a request, it will select the peer of theirs with the highest reputation.

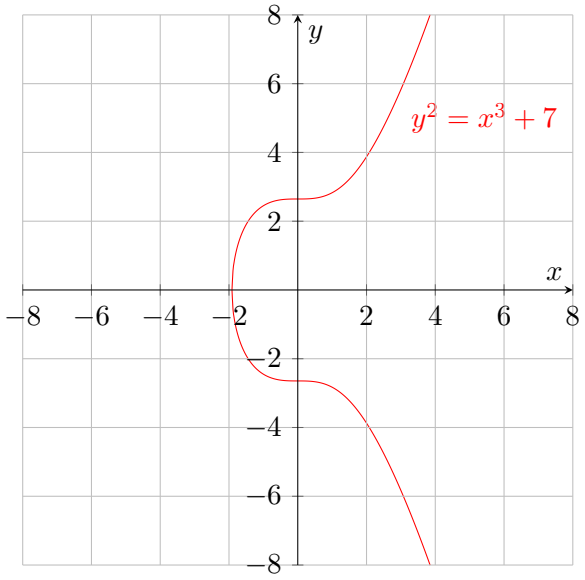

Figure 5.4: SECP256k1 elliptic curve.

Only a very rudimentary reputation system was proposed, which could easily be abused by carefully manipulating messages to perform a DoS attack on a node. This is intentional since the reputation system was not the focus of this proof of concept implementation, but the modular nature of the SNARK marketplace codebase allows easy swapping of the reputation system for a more suitable one.

Chapter 6

Implementation

The marketplace is implemented in Python version 3.11 and uses the following dependencies from the PyPI registry:

- `ecdsa` – Support for elliptic curve cryptography.
- `pytest` – Testing framework with support for unit tests, integration tests and E2E tests.

As mentioned previously, the ZoKrates library provides features for working with SNARKs.

6.1 Client Application

A node connects to the network using the client application, which consists of a client script and multiple support scripts. It has a command-line interface with multiple parameters.

```
Usage: python client.py [-k|--key <private key file>] [-v|--verbose] [-h|--help]
         [-p|--port <port number>] [-c|--command <command>] [-f|--config <config file>]

    -k, --key <private key file>   Authenticate using an existing private key file
    -v, --verbose                  Show more detailed log messages
    -h, --help                     Print this message
    -p, --port <port number>       Open the listening socket on a specific port number
    -c, --command <command>        Run semicolon separated list of commands just after
                                   client initialisation
    -f, --config <config file>     Provide a non-default configuration file
    -n, --no-color                 Don't print colored text into the terminal
```

6.1.1 Modes

The client application keeps track of two boolean variables – one defines whether verbose mode is enabled, and the second tracks whether the user is authenticated.

Verbose Mode

The verbose mode can be enabled by either using the `-v` switch or `verbose <on|off>` client command and based on its state, verbose priority messages will or will not be printed.

Authenticated and Non-authenticated Mode

If a user does not provide a private key for authentication using the `-k` switch, the client application will be started in anonymous (non-authenticated) mode where they can track of the current state of the blockchain but cannot contribute to it – they cannot create/confirm pending coin transactions, create/generate pending proofs, or create blocks.

The client script provides the `generate-key <output file>` command to generate a brand new SECP256k1 private key and save it inside `<output file>` in PEM format. The private key along with the address that will be printed will represent a new account in the blockchain.

6.1.2 Client Commands

The client application can be controlled using client commands. The semicolon-separated list of client commands can be optionally inputted using the `-c` switch when starting the application. After starting, the client commands can be inputted in the interactive mode. To list all available client commands, the `help` command can be inputted.

Complete list of available client commands, where commands marked with an asterisk are only available in authenticated mode:

- `verbose <on|off>` – Toggle verbose logging, defaults to `on` if `-v` switch was present, otherwise defaults to `off`.
- `exit` – Terminate the client along with the listening socket.
- `send <receiver address> <amount>`* – Create a coin transaction and submit it to the network.
- `request-proof <circuit hash> <parameters>`* – Request a proof to be generated and submit the request to the network.
- `select-proof-tx <proof index>`* – Check if a pending proof transaction is valid and include it in the partial block.
- `select-coin-tx <coin tx index>`* – Check if a pending coin transaction is valid and include it in the partial block.
- `partial`* – Print information about currently produced partial (unfinished) block.
- `produce-block`* – Generate a proof for each proof transaction inside the partial block, finish the block, and broadcast it to the network.
- `generate-key <output file>` – Generate SECP256k1 private key and save it in `<output file>` in PEM format.
- `inspect <block id>` – Print information about block with `<block id>`.
- `status` – Print current status of the network.
- `auth <private key file>` – Switches from anonymous mode to authenticated mode.
- `balance [<address>]` – Print current (latest known block) balance of `<address>` or self if authenticated and `<address>` is not provided.
- `logout`* – Switch to non-authenticated mode.

Command id	Command type	Command argument	Command description
GET_PEERS	Request	–	Requests a list of all peers known to the message receiver
PEERS	Response	peers : Peer[]	Sends the list of all known peers
GET_LATEST_BLOCK_ID	Request	–	Requests latest known block id known to the message receiver
LATEST_BLOCK_ID	Response	latest_block_id : int	Sends the latest known block id
GET_BLOCK	Request	block_id : int	Requests the header and body of a block with provided block id
BLOCK	Response	block : Block	Sends a block header along with block body with specified id
GET_PENDING_COIN_TXS	Request	–	Requests the list of known pending coin transactions
PENDING_COIN_TXS	Response	txs : CoinTransaction[]	Sends the list of known pending coin transactions
GET_PENDING_PROOF_TXS	Request	–	Requests the list of known pending proof transactions
PENDING_PROOF_TXS	Response	txs : ProofTransaction[]	Sends the list of known pending proof transactions
BROADCAST_BLOCK	Broadcast	block : Block	Broadcasts a newly mined block
BROADCAST_PENDING_COIN_TX	Broadcast	tx : CoinTransaction	Broadcasts a new coin transaction to be confirmed
BROADCAST_PENDING_PROOF_TX	Broadcast	tx : ProofTransaction	Broadcasts a new request for a proof to be generated and included in a block

Table 6.1: Complete list of protocol commands.

6.1.3 Communication Protocol

Clients open a socket on a port specified by the `-p` switch to listen to incoming TCP connections (Figure 6.1). Messages between clients are serialised into JSON format, which would not be optimal in practice but makes debugging in our proof of concept easy. Individual blockchain objects (blocks, transactions, and state trees) contain the verification of the incoming data inside their deserialisation method. Each message sent contains a command and a body. The list of commands and their parameters is listed in Table 6.1. Commands are split into two categories – request/response and broadcast.

6.1.4 Joining the Network

The client will request a list of peers from all of its seed nodes upon starting and will recursively seek a sufficient number of live peers. The next step consists of synchronising the state of the network which consists of fetching all missing blocks and all pending transactions. Figure 6.2 describes the sequence of messages exchanged with the new client's peers.

6.1.5 Logging Priorities

The client application logs messages with 4 different priorities:

- **VERBOSE** – Messages used for debugging, can be enabled or disabled with the `-v` switch or with `verbose <on|off>` client command.

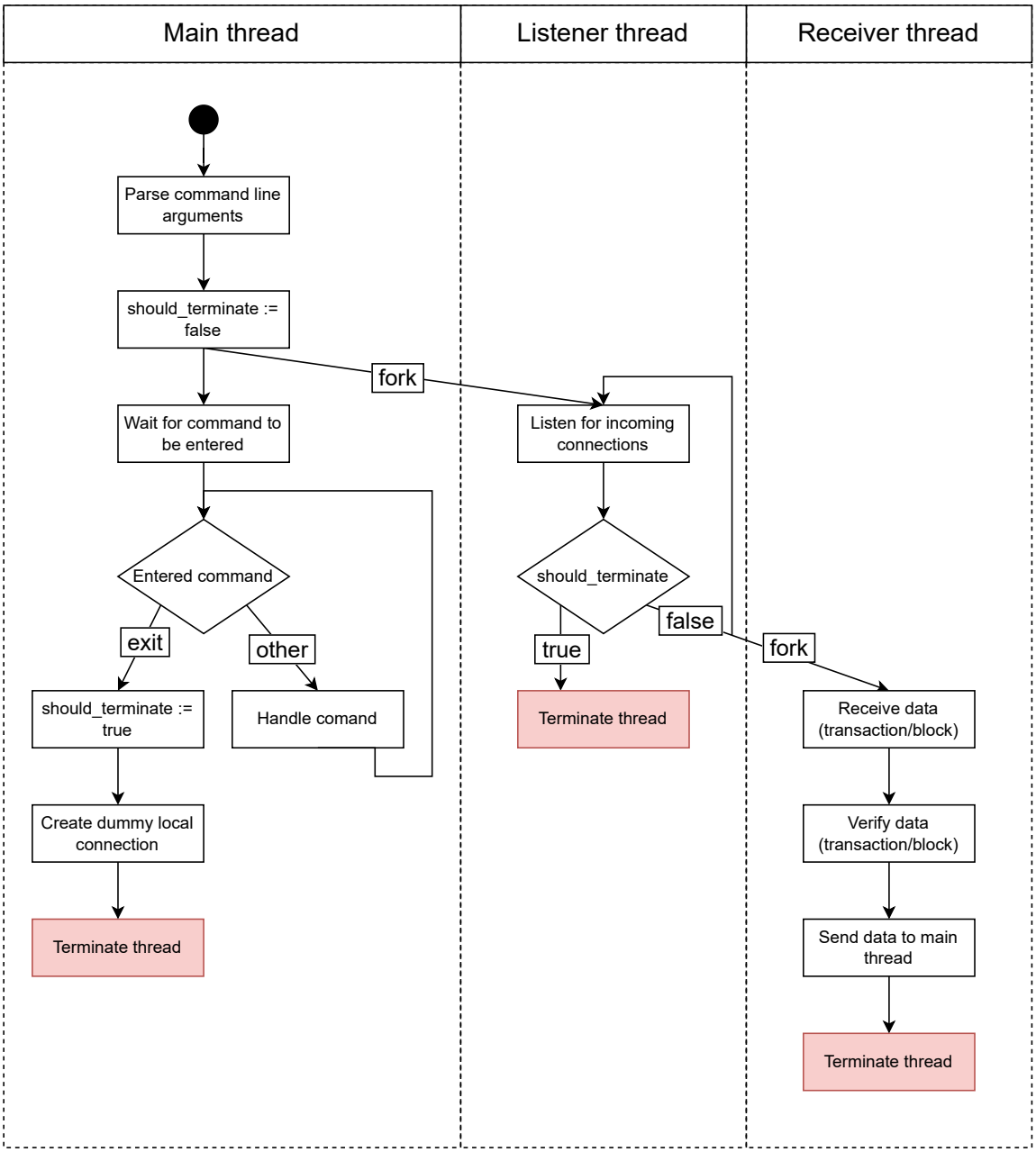

Figure 6.1: Diagram showing the actions of individual threads of the client application. The main thread is responsible for prompting the user for client commands and executing them. The single spawned listener thread opens a TCP socket which listens for incoming connections and spawns a new receiver thread for each incoming connection. The receiver thread's job is to receive incoming data (which is either a newly propagated block, a new pending transaction, or a response to client's request), verify the data and send it to the main thread.

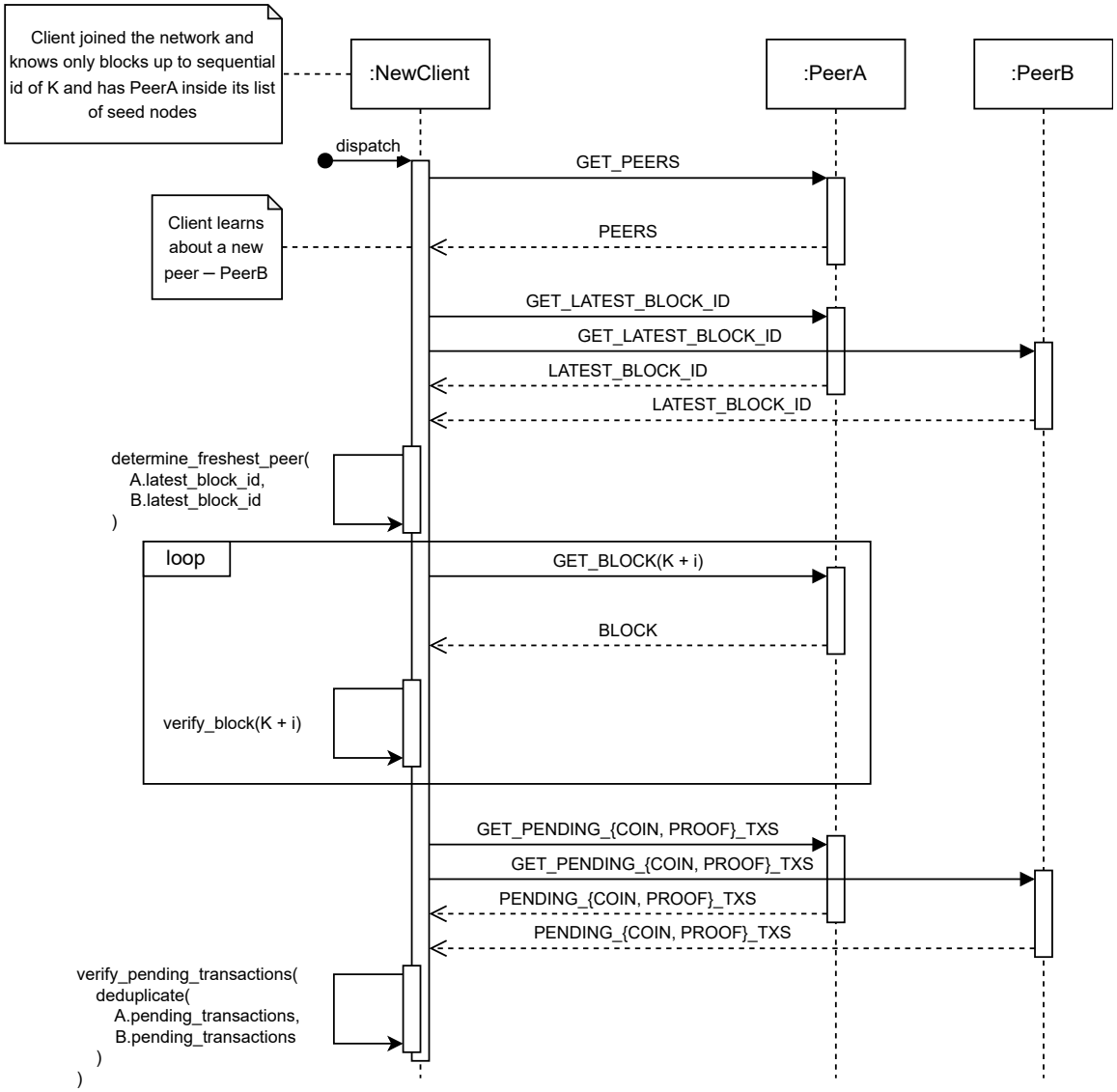

Figure 6.2: A new node described by the `NewClient` process is joining the network. The node first requests a list of peers from the statically defined seeder `PeerA`. Seeder returns a list with a single peer – represented by the `PeerB` process. The joining node requests the latest serial block id from both of the peers and determines which one of them is fresher – reports knowing larger block id. The joining node then requests and verifies its missing blocks one by one until it is synced up with the rest of the network. If it is the first time the joining nodes join the network, it will have to sync all blocks from the genesis block. After the blockchain is fetched and verified, the client will request the list of pending transactions, verify them, and starts listening to new incoming blocks and pending transactions. In a more realistic scenario, the number of seeding and peer nodes would, of course, be much higher. The recursive request for peers to `PeerB` has been removed from the diagram for clarity.

- **INFO** – General messages about current state of the application and success notifications.
- **WARNING** – Message about an event which may or may not require attention.
- **ERROR** – Messages about failures of a command or the whole application.

The priority name is always prepended to the message along with colour coding within the terminal environment unless disabled with the `--no-color` switch.

6.1.6 Network Configuration and Genesis Block

The client and genesis block parameters are isolated inside a separate JSON file for ease of modification and testing located at `src/config.json`. The most important configuration parameters are the list of seed nodes and the maximum number of peers tracked, which were discussed in this chapter. The file also contains a configuration of the genesis block. For nodes to be compatible within a network, they need to have the same genesis block statically defined inside their configuration file. The provided configuration file sets up a genesis block that assigns 1000 coins to a node whose private key file is located at `src/test/misc/private_key`.

A custom configuration file can be passed to a client using `-f` switch. This switch is used extensively in tests that verify the behaviour of various configurations.

6.1.7 ZoKrates Library Integration

The ZoKrates library functions are invoked from the client application using a static class `bind_zokrates.py` which wraps ZoKrates subprocess command execution inside Python calls for increased modularity and ease of testing.

The client application will make sure to check the installed ZoKrates version at initialisation using the `zokrates --version` command. If the absence of the ZoKrates library or incompatible version is detected, a warning message is printed to the console.

6.1.8 Example Uses

To illustrate the functionality of the client application, here is a list of some example commands for common use cases. Recall that commands after the `-c` switch could also be entered one by one in interactive mode.

- `python src/client.py -c "help"` – Prints a list of all available client commands (also listed in Section 6.1.2).
- `python src/client.py -c "generate-key key.pem"` – Generates a new private key into the `key.pem` file and prints its corresponding address to `stdout`. This private key can be used with the `-k` parameter to use client in authenticated mode.
- `python src/client.py -k key.pem -c "send 0008b58b73bbfd6ec26f599649ecc624863c775e034c2afea0c94a1c0641d8f000 50"` – Enters authenticated mode, creates a pending coin transaction (if sender funds are sufficient) and broadcasts it to the network.

- `python src/client.py -k key.pem -c "request-proof f6d00f1b20054ec6660af23c8b5953ae8799ddbb8c9bd9e1808376fef65d970e 2 3 6"` – Enters authenticated mode, creates a pending proof transaction (if sender funds are sufficient to pay the fee) and broadcasts it to the network.

Some commands require the client to synchronise to the network first. This is accomplished by launching the client in an interactive mode with `python src/client.py -k key.pem` without using the `-c` switch. The client will initiate network synchronisation, which consists of pending transaction and block fetch. After the client has synchronised, the following commands are ready to use – `balance` command to retrieve information from the state tree of the latest block about balances of accounts, `status` and `inspect` commands to find information about the blockchain.

6.2 Testing

The client application and the classes representing blockchain objects are tested using the `pytest` library to check the validity of the program and avoid regressions during development. A large portion of the codebase was developed using test-driven development.

6.2.1 Client Application Testing

The client application is being tested in an end-to-end manner where a single or multiple instances are run. These tests could be divided into two categories:

- **Non-interactive tests**, where all of the tested commands are inputted using the command line parameters when starting the client with the `-c` switch. At the end of the command list there is an `exit` command to exit the client application, meaning there are no commands input inside of the interactive client environment. Tests either verify the output of the client application or correctness of created files (for example when generating a private key with the `generate-key` command).
- **Interactive tests**, which launch one or more client applications and communicate with them using pipelines which are used to send commands and receive inputs. This allows for testing of client synchronisation, inter-client communication and gradual reaction of clients on a series of commands in time.

Chapter 7

Discussion

The application was designed and implemented as a proof of concept to show that a blockchain can be implemented, where an individual needing to generate a SNARK proof defined by a ZoKrates program can exchange a fee in native currency to have that proof generated for them. Proof generation is a computationally difficult task and could even be impossible to perform on some low-power devices that need SNARK generation for their applications, such as mobile wallets. That is why outsourcing SNARK generation could improve the range of possible use cases of zero-knowledge technologies.

Considering that SNARKs are often used for outsourcing verifiable computation [53, 54, 48] and that our market facilitates outsourcing of SNARK generation, we deal with double outsourcing of verifiable computation. The other major use case of SNARKs in their zero-knowledge form is private computation. Due to the design of our marketplace, we are not able to facilitate this use case.

7.1 Solution Analysis and Limitations

The designed architecture leverages the benefits of blockchains to implement a SNARK marketplace. Native currency was conceived as a means to reward a miner for the work performed. The blockchain tracks both generated SNARK proofs and coin transactions to create an immutable ledger. The proof of work consensus protocol is used to secure the blockchain as described in Section 2.5. Typically, the work in PoW has no other uses outside the consensus protocol, leading to a waste of computational resources. We remedied this problem by using the actual work needed to generate the proof as a means of demonstrating consensus power.

The probability of being able to generate a block in a classical PoW protocol, such as Bitcoin, is proportional to the computational power of a node, but due to the high randomness in the search for a partial collision, it is very hard for a single node to mine multiple consecutive blocks in row even for the strongest node in the network, leading to increased decentralisation. One of the flaws of our consensus protocol in comparison to classical PoW as in Bitcoin is that the generation of a block does not contain that much randomness to prevent the strongest node from generating multiple consecutive blocks. This could be mitigated by decreasing the freedom of a node in regards to selecting arbitrary proof transactions leading to decreased ability of a node to nitpick most computationally-cost-effective proofs. The solution implementing this also has the benefit of reducing duplicated work and is discussed in Section 7.2.1.

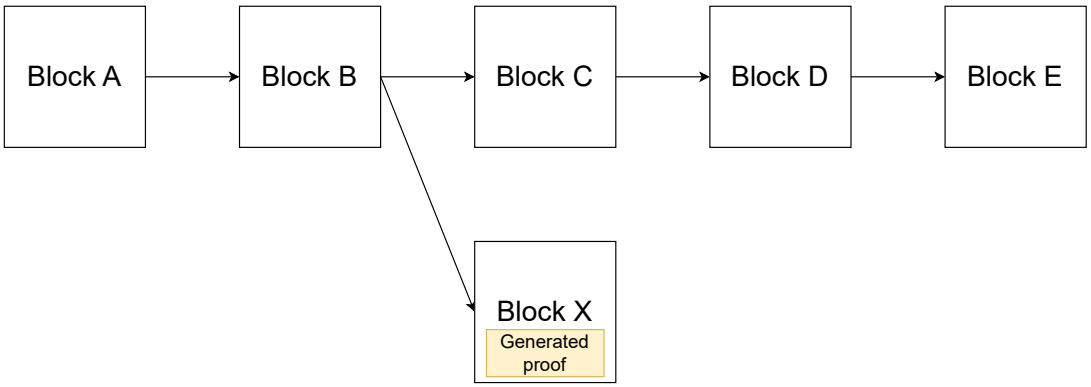

Figure 7.1: Main blockchain containing blocks A through E and an orphaned block X on a rejected fork. Proofs on block X are valid, thus they can be read by the proof requesters if the block was transmitted throughout the network.

As with every blockchain, the security of the solution depends on the security of the underlying cryptographic schemes. We have chosen the SHA-256 hashing function and ECDSA over the SECP256k1 curve as the underlying cryptographic algorithms, since they are widely used in many blockchain applications and are considered secure as of the writing of this thesis.

7.1.1 Forks

As with other PoW resp. PoUW consensus protocols, our proposed solution suffers from forks – divergences of the blockchain. Unintentional forks can happen due to inability of a node to see the whole picture of the network at some exact time. Malicious nodes can also induce forks intentionally, but the length of the induced fork is proportional to the computing power of the malicious nodes in relation to the rest of the network, since the fork conflicts are resolved by the *longest chain rule* – accepting the longer chain, since more work has been put into it.

The ability to perform *selfish mining*, a malicious behaviour in which a node or group of colluding nodes withhold from publishing blocks immediately after mining them and reintroducing them later after some time to overwrite publicly known blocks, requires controlling at least a third of the network. The ability of malicious nodes to selfishly mine in PoW networks has negative consequences – wasted resources which were spent on orphaned blocks and increased centralisation, the latter of which is inherent to a third of the network being controlled by a single entity.

Our network has a single negative consequence of forks beyond those which are applicable to Bitcoin and other classical PoW networks, and it is inherent to how proof requesters read their proof from the blockchain. If an orphaned block on a rejected fork has been transmitted far enough to reach the proof requester, the requester can read the proof without it having to be included in the main blockchain (Figure 7.1). This is not a problem on the requester side, since the fee they have paid for the proof generation is locked and cannot be withdrawn. But for a prover this means that they will not receive the reward because their block has been orphaned. This highlights the importance of long block time and low network latency to decrease the frequency of updates and the time it takes for the network to synchronise.

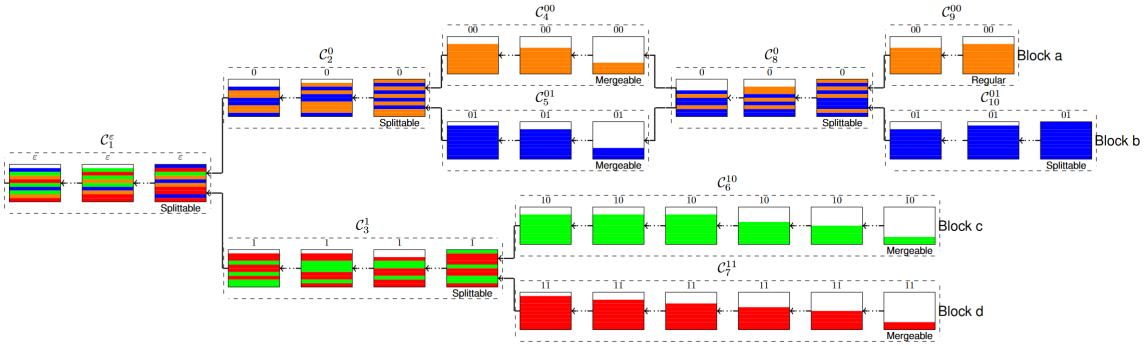

Figure 7.2: Sycomore ledger creates a split in the block DAG in times of high traffic. Transaction here are colour-coded based on their prefix, which determined their position after a split. Retrieved from [2].

7.2 Possible Enhancements

The design and implementation are just a proof of concept and is in no way ready for production use. This section suggests some enhancements that would improve performance, improve security, or widen the area of possible use cases.

7.2.1 Reducing Wasted Work

As mentioned in Section 7.1.1, the current design suffers from wasted work – when a node publishes a block and that block is accepted by the network, all of the transactions contained within the block are confirmed and have to be removed from all partial blocks being produced by other nodes. According to Section 5.3, all selected proofs have to be selected and committed prior to the generation of the very first proof of the block. That means that if only a single one of those coin or proof transactions was confirmed inside block generated by another node, the entire pending block would need to be started over, since a transaction cannot be confirmed inside multiple blocks.

To remedy this problem, a solution inspired by Sycomore [2] is proposed. Sycomore is a network that uses a directed acyclic graph (DAG) of blocks instead of a linear blockchain to reduce the number of forks and conflicts while increasing the network throughput (Figure 7.2). One of its aspects that we could adapt is the use of transaction identifier prefixes to determine the position of a transaction within a blockchain. We will apply the aspect of sorting transactions by prefix to our blockchain without the introduction of DAGs.

Consider our network with multiple miners. If two miners have (even a slight) overlap in the set of transactions in their partial blocks, the miner that does not publish the block first will have to throw away a large portion of their work. If two miners were currently producing a block of t transactions (for simplicity, we consider each transaction to be equally difficult to confirm) with p transactions pending throughout the network, then the probability r of two miners selecting distinct subsets of transactions is:

$$r(p,t) = 1 - \frac{\binom{p-t}{t}}{\binom{p}{t}} \tag{7.1}$$

This shows that the number of conflicts is proportional to the number of transactions within the block and inversely proportional to the total number of pending transactions. If

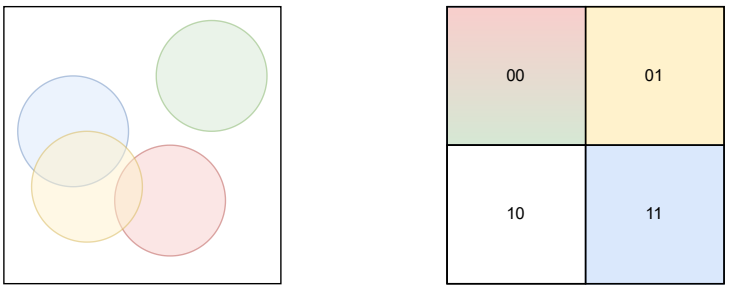

Figure 7.3: Visualisation of randomly selected transactions (left) and transactions with specific prefixes (right), where the colour distinguishes each of the consensus participants. Using prefixes defines strict boundaries decreasing the probability of conflicts.

we consider a network having more miners, then the probability of conflict is almost 100 %. If we introduced transaction splitting, where miners could only confirm transactions with prefix matching the prefix of the miner's address, then we could achieve the separation of transactions into distinct groups (Figure 7.3). The size of the prefix being matched could be based on the current difficulty parameter of the network, which is already tracking the number of miners inside the network. This would not eliminate conflicts altogether, but would decrease their number drastically, since the conflicts will be larger but less common. The number of conflicting transactions does not matter since even a single transaction overlap would require a redo of the entire block along with its proofs.

7.3 Practical Use Cases

We present a couple of practical use cases that could leverage the SNARK marketplace to improve the efficiency of existing technologies.

7.3.1 Zk-Bridge

Zk-Bridge (Section 4.4) [54] uses zero-knowledge proofs for cross-chain transfer of tokens. SNARKs are used for proving the source blockchain status to the verifier contract on the receiver chain; therefore, our marketplace not supporting zero-knowledge aspect of SNARKs does not pose any problems. Our SNARK marketplace could be used to outsource SNARK generation within the relay network of the zk-Bridge architecture.

The zk-Bridge whitepaper does not discuss incentive schemes of the relay network nodes and leaves incentive design for future work, but one of the possible reward schemes for the nodes could be to reward the proof submitter with tokens on the receiver chain. Since there is already a verifier contract in place, it would be very simple to add reward functionality to it.

From the security standpoint of the relay network, there is only a liveness requirement – there needs to be at least one honest node in the relay network regardless of the number of malicious nodes in the network. Denial of service (DoS) attacks are disincentivized by the contract invocation requiring a fee in the receiver chain currency (often called *gas*).

7.3.2 Zk-Rollups

Zk-Rollups [18, 23] were designed to help increase blockchain scalability by bundling transaction data into batches and moving computation outside the blockchain, while using SNARKs to verify the correctness of the computation. These solutions are often referred to as *layer-2* because they provide a layer of abstraction above the original blockchain network, which is called *layer-1* or the *base-layer*.

Outsourcing of the rollup SNARK generation to the SNARK marketplace would allow nodes with lower computational power to participate in the rollups as well leading to a decreased fee on the layer-1 blockchain. The economical feasibility of paying fee for a SNARK generation on the marketplace chain to decrease the fee on the rollup blockchain would depend on the relative price of the respective cryptocurrencies, but since the marketplace uses proof of useful work where the SNARK generation is also used as means to perform the consensus protocol, it could be expected that it would be cheaper than having to generate a SNARK outside of the marketplace.

Chapter 8

Conclusion

The first part of this thesis listed and elaborated on common techniques for computational and storage offloading off of blockchains along with explaining the necessary cryptographic primitives and concepts. Next, solutions for offloading were discussed with an emphasis on zero-knowledge proofs, namely STARKs, SNARKs and their differences. Cryptographic accumulators were mentioned as a means of optimising the storage requirements of blockchain networks.

Zero-knowledge proofs are an integral part of many technologies currently used in production environments for two main reasons – they allow outsourcing of verifiable computation and proving a statement without having to reveal its details. Blockchain networks use these properties to enable cross-chain token transfer [53, 54], network throughput scaling by using layer-2 solutions [23], transaction confidentiality [14] and many more.

Then in the second part followed a design and proof of concept implementation of a SNARK marketplace where computational power could be exchanged for native blockchain currency in an environment with proof of useful work (PoUW) consensus protocol. PoUW mitigates one of the most criticised problems of plain PoW which is the amount of wasted computational work and its environmental impact by utilising the actual proof generation to demonstrate the consensus power of a node. The implementation was developed using Python along with the ZoKrates library, which is used to compile arithmetic circuits, perform a trusted setup using SMPC, generate proofs, and verify proofs.

Having SNARKs generated by a third party inside a marketplace enables faster and more cost-efficient development of blockchain platforms that focus on verifiable computation and privacy preservation. Some of the existing technologies that could benefit from this are zk-bridge [54], which enables cross-chain token and data transfer, and zk-rollups [18] designed to bundle transactions and reduce fees paid on the level-1 blockchain.

Bibliography

[1] Ahanger, T., Aldaej, A., Atiquzzaman, M., Ullah, I. and Yousufudin, M. Distributed Blockchain-Based Platform for Unmanned Aerial Vehicles. *Computational Intelligence and Neuroscience*. August 2022, vol. 2022, p. 1–16. DOI: 10.1155/2022/4723124.

[2] Anceaume, E., Guellier, A., Ludinard, R. and Sericola, B. Sycomore: A Permissionless Distributed Ledger that Self-Adapts to Transactions Demand. In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. 2018, p. 1–8. DOI: 10.1109/NCA.2018.8548053.

[3] Au, M. H., Tsang, P. P., Susilo, W. and Mu, Y. Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems. In: Fischlin, M., ed. *Topics in Cryptology – CT-RSA 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, p. 295–308. ISBN 978-3-642-00862-7.

[4] Back, A. Hashcash - A Denial of Service Counter-Measure. September 2002. Available at: http://www.hashcash.org/papers/hashcash.pdf.

[5] Ball, M., Rosen, A., Sabin, M. and Vasudevan, P. N. *Proofs of Useful Work* [Cryptology ePrint Archive, Paper 2017/203]. March 2017. Available at: https://eprint.iacr.org/2017/203.

[6] Barić, N. and Pfitzmann, B. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In: Fumy, W., ed. *Advances in Cryptology — EUROCRYPT '97*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, p. 480–494. ISBN 978-3-540-69053-5.

[7] Ben Sasson, E., Bentov, I., Horesh, Y. and Riabzev, M. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In: *Electron. Colloquium Comput. Complex.* 2017. Available at: https://api.semanticscholar.org/CorpusID:5637668.

[8] Ben Sasson, E., Bentov, I., Horesh, Y. and Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.* 2018, vol. 2018, p. 46. Available at: https://api.semanticscholar.org/CorpusID:44557939.

[9] Benaloh, J. and Mare, M. de. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In: Helleseth, T., ed. *Advances in Cryptology — EUROCRYPT '93*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, p. 274–285. ISBN 978-3-540-48285-7.

[10] Berentsen, A., Lenzi, J. and Nyffenegger, R. A Walk-through of a Simple Zk-STARK Proof. *SSRN*. 2022. DOI: 10.2139/ssrn.4308637. Available at: https://ssrn.com/abstract=4308637.

[11] Berentsen, A., Lenzi, J. and Nyffenegger, R. An Introduction to Zero-Knowledge Proofs in Blockchains and Economics. *Federal Reserve Bank of St. Louis Review*. Fourth Quarter 2023, vol. 105, no. 4, p. 280–294. DOI: 10.20955/r.105.280-94. Available at: https://doi.org/10.20955/r.105.280-94.

[12] Bjoernsen, K. Koblitz Curves and its practical uses in Bitcoin security. In:. 2015. Available at: https://api.semanticscholar.org/CorpusID:16632570.

[13] Bonneau, J., Meckler, I., Rao, V. and Shapiro, E. *Mina: Decentralized Cryptocurrency at Scale*. March 2020. Available at: https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf.

[14] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P. et al. Bulletproofs: Short Proofs for Confidential Transactions and More. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, p. 315–334. DOI: 10.1109/SP.2018.00020.

[15] Camenisch, J. and Lysyanskaya, A. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: Yung, M., ed. *Advances in Cryptology — CRYPTO 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, p. 61–76. ISBN 978-3-540-45708-4.

[16] Casas, P., Romiti, M., Holzer, P., Mariem, S. B., Donnet, B. et al. Where is the Light(ning) in the Taproot Dawn? Unveiling the Bitcoin Lightning (IP) Network. In: *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. 2021, p. 87–90. DOI: 10.1109/CloudNet53349.2021.9657121.

[17] Chainlink. *Understanding the Difference Between zk-SNARKs and zk-STARKS*. 2023. Accessed: 2024-05-17. Available at: https://chain.link/education-hub/zk-snarks-vs-zk-starks.

[18] Chainlink. *What Are Zk-Rollups (Zero-Knowledge Rollups)?* 2023. Accessed: 2024-05-15. Available at: https://chain.link/education-hub/zero-knowledge-rollup.

[19] Cohn, B., Shapiro, E. and Tekişalp, E. *Mina: Economics and Monetary Policy*. October 2020. Available at: https://minaprotocol.com/wp-content/uploads/economicsWhitepaper.pdf.

[20] Du, W. and Atallah, M. J. Secure multi-party computation problems and their applications: a review and open problems. In: *Proceedings of the 2001 Workshop on New Security Paradigms*. New York, NY, USA: Association for Computing Machinery, 2001, p. 13–22. NSPW '01. DOI: 10.1145/508171.508174. ISBN 1581134576. Available at: https://doi.org/10.1145/508171.508174.

[21] Eberhardt, J. and Tai, S. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, p. 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199.

[22] Ethereum Foundation. *Merkle Patricia Trie*. 2024. Accessed: 2024-05-09. Available at: https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/.

[23] Ethereum Foundation. *Zero-knowledge rollups*. 2024. Accessed: 2024-05-15. Available at: https://ethereum.org/en/developers/docs/scaling/zk-rollups/.

[24] Goel, K. *Mina Protocol - Small but Mighty*. 2022. Accessed: 2024-05-22. Available at: https://messari.io/report/mina-protocol-small-but-mighty.

[25] Hijfte, S. V. *Blockchain Platforms: A Look at the Underbelly of Distributed Platforms*. Morgan & Claypool Publishers, 2020.

[26] Jana, R. K., Ghosh, I. and Wallin, M. W. Taming energy and electronic waste generation in bitcoin mining: Insights from Facebook prophet and deep neural network. *Technological Forecasting and Social Change*. 2022, vol. 178, p. 121584. DOI: https://doi.org/10.1016/j.techfore.2022.121584. ISSN 0040-1625. Available at: https://www.sciencedirect.com/science/article/pii/S0040162522001160.

[27] Jerzak, Z. and Fetzer, C. Bloom filter based routing for content-based publish/subscribe. In: *Distributed Event-Based Systems*. 2008. Available at: https://api.semanticscholar.org/CorpusID:13891409.

[28] Jhanwar, M. P. and Tiwari, P. R. *Trading Accumulation Size for Witness Size: A Merkle Tree Based Universal Accumulator Via Subset Differences* [Cryptology ePrint Archive, Paper 2019/1186]. 2019. Accessed: 2024-04-25. Available at: https://eprint.iacr.org/2019/1186.

[29] Kang, F. *How To Transform Interactive Proofs Into Non-Interactive Proofs? Fiat-Shamir Heuristic!* May 2023. Accessed: 2024-05-23. Available at: https://en.foresightnews.pro/how-to-transform-interactive-proofs-into-non-interactive-proofs-fiat-shamir-heuristic/.

[30] Karpeev, I. *=nil; zkLLVM + Proof Market: Enabling accessible and effective zkProofs for all*. 2023. Accessed: 2024-05-21. Available at: https://nil.foundation/blog/post/proof-market-and-zkllvm-pipeline.

[31] King, S. and Nadal, S. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. In:. 2012. Available at: https://api.semanticscholar.org/CorpusID:42319203.

[32] Laanoja, R. *Simple hash-linking based time-stamping scheme*. December 2009. Accessed: 2024-01-18. Available at: https://commons.wikimedia.org/wiki/File:Hashlink_timestamping.svg.

[33] Lamport, L. Password authentication with insecure communication. *Communications of The ACM*. 1981, vol. 24, p. 770–772. Available at: https://api.semanticscholar.org/CorpusID:12399441.

[34] Li, J., Li, N. and Xue, R. Universal Accumulators with Efficient Nonmembership Proofs. In: Katz, J. and Yung, M., ed. *Applied Cryptography and Network Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 253–269. ISBN 978-3-540-72738-5.

[35] Marlin. *What is Marlin?* 2024. Accessed: 2024-05-21. Available at: https://docs.marlin.org/learn/what-is-marlin/.

[36] Menezes, A. J., Oorschot, P. C. van and Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 2001.

[37] Merkle, R. C. A Digital Signature Based on a Conventional Encryption Function. In: Pomerance, C., ed. *Advances in Cryptology — CRYPTO '87*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, p. 369–378. ISBN 978-3-540-48184-3.

[38] Microsoft. *Trusted Execution Environment | Microsoft Learn*. Microsoft, June 2023. Accessed: 2024-01-15. Available at: https://learn.microsoft.com/en-us/azure/confidential-computing/trusted-execution-environment.

[39] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available at: http://www.bitcoin.org/bitcoin.pdf.

[40] Nil Foundation. *What is zkLLVM?* 2024. Accessed: 2024-05-21. Available at: https://docs.nil.foundation/zkllvm/overview/what-is-zkllvm.

[41] Odoom, J., Huang, X., Zhou, Z., Danso, S., Benedicta, N. E. N. et al. Blockchain-assisted sharing of electronic health records: a feasible privacy-centric constant-size ring signature framework. *International Journal of Computers and Applications*. Taylor & Francis. 2023, vol. 45, no. 9, p. 564–578. DOI: 10.1080/1206212X.2023.2252238. Available at: https://doi.org/10.1080/1206212X.2023.2252238.

[42] Özçelik, I., Medury, S., Broaddus, J. T. and Skjellum, A. An Overview of Cryptographic Accumulators. *CoRR*. 2021, abs/2103.04330. Available at: https://arxiv.org/abs/2103.04330.

[43] Petkus, M. Why and How zk-SNARK Works. *CoRR*. 2019, abs/1906.07221. Available at: http://arxiv.org/abs/1906.07221.

[44] Poly Team. *PolyNetwork: An Interoperability Protocol for Heterogeneous Blockchains*. Poly Network, May 2020. Available at: https://www.poly.network/PolyNetwork-whitepaper.pdf.

[45] Polyhedra Network. *Introducing the Bitcoin Messaging Protocol with zkBridge*. Polyhedra Network Blog, December 2023. Accessed: 2024-05-10. Available at: https://blog.polyhedra.network/introducing-the-bitcoin-messaging-protocol-with-zkbridge.

[46] Poon, J. and Dryja, T. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016. Available at: https://lightning.network/lightning-network-paper.pdf.

[47] Reed, I. S. and Solomon, G. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*. Society for Industrial and Applied Mathematics. 1960, vol. 8, no. 2, p. 300–304. ISSN 03684245. Available at: http://www.jstor.org/stable/2098968.

[48] Slávka, S. *Mobile Cryptocurrency Wallet Based on zk-SNARKs and Smart Contracts*. Brno, CZ, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/23223/.

[49] Smart, N. *Cryptography: An Introduction*. 3rd ed. McGraw-Hill College, 2004. ISBN 978-0077099879.

[50] Stallings, W. *Cryptography and Network Security: Principles and Practice*. 6th ed. USA: Prentice Hall Press, 2013. ISBN 0133354695.

[51] Tarkoma, S., Rothenberg, C. E. and Lagerspetz, E. Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Communications Surveys & Tutorials*. 2012, vol. 14, no. 1, p. 131–155. DOI: 10.1109/SURV.2011.031611.00024.

[52] Valiant, P. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. In: Canetti, R., ed. *Theory of Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 1–18. ISBN 978-3-540-78524-8.

[53] Westerkamp, M. and Eberhardt, J. *ZkRelay: Facilitating Sidechains using zkSNARK-based Chain-Relays* [Cryptology ePrint Archive, Paper 2020/433]. 2020. Accessed: 2024-05-18. Available at: https://eprint.iacr.org/2020/433.

[54] Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y. et al. *ZkBridge: Trustless Cross-chain Bridges Made Practical*. October 2022. Available at: https://arxiv.org/abs/2210.00264.

[55] Zarick, R., Pellegrino, B., Zhang, I., Kim, T. and Banister, C. *LayerZero Whitepaper v2.1.0*. LayerZero Labs. Available at: https://layerzero.network/publications/LayerZero_Whitepaper_V2.1.0.pdf.

[56] Zhou, Q., Huang, H. and Zheng, Z. Solutions to Scalability of Blockchain: A Survey. *IEEE Access*. January 2020, PP. DOI: 10.1109/ACCESS.2020.2967218.

[57] ZoKrates Team. *ZoKrates: A toolbox for zkSNARKs on Ethereum*. November 2023. Accessed: 2024-05-08, version 0.8.8. Available at: https://zokrates.github.io/.

Appendix A

Contents of the Attached Medium

The `README.md` file contains a brief introduction and the steps necessary to run the application. Other notable folders are:

- `src/` – contains the Python source files implementing the application.
- `src/test/` – contains the definition of all the tests and support files.
- `src/circuit/` – contains files necessary for the generation and verification of individual proofs.

```
CD-ROM
├── src/
│   ├── circuit/
│   ├── test/
│   │   ├── misc/
│   │   ├── test_bind_zokrates.py
│   │   ├── test_block.py
│   │   ├── test_client.py
│   │   ├── test_coin_tx.py
│   │   ├── test_proof_tx.py
│   │   └── test_state_tree.py
│   ├── bind_zokrates.py
│   ├── block_body.py
│   ├── block_header.py
│   ├── block.py
│   ├── client.py
│   ├── coin_tx.py
│   ├── config.json
│   ├── encodeable.py
│   ├── network.py
│   ├── peer.py
│   ├── proof_tx.py
│   ├── state_tree.py
│   └── util.py
├── README.md
└── requirements.txt
```