EASTERN SWITZERLAND
UNIVERSITY OF APPLIED SCIENCES

FOCUS PROJECT 2

Zero-Knowledge Inclusion Proofs

Author:
Roman BÖGLI

Supervisors:
Dr. Kaoutar EL KHIYAOUI
Dr. Angelo DE CARO
Prof. Dr. Mitra PURANDARE

A work submitted in fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Science (MSE CS)

at

Institute for Network and Security (INS),
Departement of Computer Science

in collaboration with

Security Research Department,
IBM Research Zurich Lab

21. August 2023

Abstract

This work discusses a solution for implementing a zero-knowledge inclusion proof using existing software components, referred to as automation frameworks. In the beginning, the use case that necessitates such proofs is stated together with establishing the contextual backdrop. This includes elaborating essential prerequisites such as cryptographic hash functions, commitment schemes, and Merkle trees.

Furthermore, it presents an overview of zero-knowledge proof systems, detailing their core characteristics, analogies, and real-world applications. The discussion delves into the two prominent implementation families, namely *zkSNARK* and *zkSTARK*, emphasizing their distinguishing features. To ensure resistance against potential threats originating from quantum computers, the proposed approach centers on utilizing non-interactively employed zkSTARK proofs enabled through the Fiat-Shamir transformation.

Finally, this work formally states the objective of the zero-knowledge inclusion proof for the specific use case and proposes an algorithmic specification. A curated selection of promising automation frameworks with the potential to facilitate the objective's implementation is presented, with in-depth scrutiny applied to two specific frameworks: *RISC Zero* and the *Winterfell*. The work concludes by discussing initial experiences with these frameworks and outlining future endeavors to chart the path towards implementing a minimal viable product.

The appendix complements the discussion by providing an overview of current post-quantum cryptography developments.

Keywords: zero-knowledge proof, proof of inclusion, Merkle tree, zkSNARK, zkSTARK, RISC Zero, Winterfell

Declaration of Authorship

I, Roman BÖGLI, declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge.

Contents

1 Introduction

Advances in digitalization and connectivity through the internet have also affected a millennial-old means of transaction, namely money. What used to be coins made out of precious metals in ancient times has been replaced with physical and later digital debt notes.

The fields of computer science and cryptography opened the door to the most recent type of medium of exchange, namely *cryptocurrencies* which elevates the quality of money traits even more. These traits include unforgeability, verifiability, portability, divisibility, fungibility, durability, and — exclusively in the case of digital money — programmability. Prominent cryptocurrencies, such as Bitcoin for example, additionally introduce the trait of being censorship-resistant through the use of *Distributed Ledger Technology (DLT)* powered by proof-of-work consensus mechanisms[1].

1.1 Problem Statement

A disadvantage of DLT systems concerns the risk of loss of user privacy. This stems from the requirement for transactions to remain auditable for all, resulting in their transparent recording within publicly available and append-only blockchain data structures. Consequently, observers may permanently track individual token movements, negatively impacting user privacy [2].

Also, the security model of digital money usually breaks in the scenario of compromised, stolen, or lost keying material used for digital signatures. Although there are various strategies to minimize this risk [3], there is no such thing as absolute security. This becomes especially delicate in systems where a money-issuing authority uses keying material to sign or *mint* new tokens, allowing a successful attacker to mimic minting and thereby counterfeit money.

Lastly, the concern of being quantum-resistant urges adopting new algorithms grounded on *Post-Quantum Cryptography (PQC)*. Therefore, a future-oriented digital money solution must account for this development and upgrade the employed cryptographic primitives to quantum-safe versions. Appendix A provides an overview of the current developments in the realm of PQC.

[1] There exists many different consensus algorithms besides proof-of-work, as surveyed by Alsunaidi and Alhaidari [1].

In summary, the problem encompasses the question of *how to verifiably prove a specific token's validity in a trustless and quantum-resistant setting without revealing which specific token it concerns in order to preserve user privacy*.

This work describes a solution to this problem that employs *Zero-Knowledge Proof (ZKP)* protocols to convince a verifying party of a token's inclusion in a public set of valid tokens, encoded as *Merkle tree* data structure, without revealing the token specifically. Preliminary, the fundamental concepts of cryptographic hash functions, commitment schemes, and Merkle trees are explained, followed by a general introduction to ZKP protocols. In order to contribute to the prospective goal of implementing a minimum viable product of such a proof system, this work additionally provides an overview of identified implementation facilitating automation frameworks, which forms the basis for future endeavors in this topic.

1.2 Outline

The rest of this work is structured as follows. Chapter 2 addresses the essential prerequisites in order to comprehend the remaining content. Chapter 3 delves into the domain of ZKP systems, offering insights into their general characteristics and exploring potential areas of application. Chapter 4 formulates the solution approach to the above problem statement and specifies the mechanism of the proving algorithm. Furthermore, this chapter offers insights into potential frameworks that facilitate implementation, with a more detailed exploration of two frameworks. Finally, Chapter 5 concludes the key insights and outlines prospects for future work.

2 Prerequisites

This chapter establishes the foundational framework for upcoming discussions, exploring three essential sections: cryptographic hash functions, commitment schemes, and Merkle trees. These well-established concepts serve as fundamental tools with versatile applications in the realm of computer science.

2.1 Cryptographic Hash Functions

A cryptographic hash function H represents a one-way function, whereas d is a data input of arbitrary length[1] and $H(d)$ a random but deterministic output of fixed length. This is formally expressed as $H(d) : \{0,1\}^\star \mapsto \{0,1\}^n$, whereas n represents a constant. $H(d)$ is also denoted as *digest* or *hash value* of d. Besides the irreversibility property, such functions are also considered chaotic, meaning that small changes on the input side lead to massive changes in its output. Due to the infinite input size and the finite output size, it is unavoidable that two different inputs lead to the same digest, which defines a *collision*. This is, however, not an issue as long as such collisions are infeasible to find. In this context, infeasibility refers to computational infeasibility, i.e., theoretically possible but considered impractical as it would require excessive resources. Secure hash functions must have [4]:

- **Preimage Resistance:** A hash function is said to be preimage resistance when it is practically infeasible to find a d for a given $h = H(d)$. Due to the fact that no inversion function exists, d can only be found through exhaustive search, also known as *brute force* search.
- **Second Preimage Resistance:** Describes the infeasibility for a given d to find d' with $d \neq d'$ and $H(d) = H(d')$. This property is also known as *weak* collision resistance.
- **(Strong) Collision Resistance:** Describes the infeasibility of finding a pair (d, d') with $d \neq d'$ such that $H(d) = H(d')$. This is similar to the previous property except that the constraint in this problem is less stringent or demanding and thus increases the resistance quality if fulfilled.

[1] In literature also denoted as message m.

2.2 Commitment Schemes

Commitment schemes are cryptographic protocols that allow a prover $\mathcal{P}$ to temporarily hide some information $\mathcal{I} \subseteq \{0,1\}^*$ from $\mathcal{V}$ with no option for subsequent alteration. In other words, it is a tool that forces $\mathcal{P}$ to decide on $\mathcal{I}$ while allowing $\mathcal{V}$ to verify the originality of this decision later in time. Commitment schemes must fulfill two properties [5, 6]:

- **Hiding Property:** The published value that hides $\mathcal{I}$ should not reveal anything about $\mathcal{I}$. Cryptographic hash functions are an excellent choice to fulfill this property, thanks to their unpredictability and irreversibility. Concatenating a random secret value to $\mathcal{I}$ before hashing leverages the quality of this property as it thwarts any search strategy of $\mathcal{V}$[2].
- **Binding Property:** $\mathcal{P}$ must not be able to alter $\mathcal{I}$ after commitment without being detected by $\mathcal{V}$. The deterministic behavior of hash functions enables $\mathcal{V}$ to verify $\mathcal{P}$'s commitment to $\mathcal{I}$ upon revelation. Although hash function collisions exist in theory due to infinite input and finite output possibilities, it remains computationally infeasible for $\mathcal{P}$ to deceive $\mathcal{V}$ at this point.

Use cases requiring two parties to agree on random choices rely on such commitment schemes. Fair gaming is an example of such. Assume Alice and Bob aim to decide on a fair coin toss, where 0 represents heads and 1 tails. Note that this protocol is tailored for sequential message exchange, which is typically the case in Internet communication. Alice chooses her toss outcome $T_A := \{\, t \mid t \in \{0,1\} \,\}$ and sends Bob the commitment $H(T_A \mid r) = h_A$, whereas r is a concatenated («|» operator) secret random value also known as a *blinding factor*. Latter is crucial to sustaining the commitment's hiding property in this particular binary domain case. Bob does the same without knowing T_A and immediately reveals his toss outcome T_B to Alice. Next, Alice reveals T_A together with r so that Bob can verify its originality by recomputing h_A. This convinces Bob that Alice did not sabotage the randomness of their collective coin toss $T_{AB} = \texttt{XOR}(T_A, T_B)$ after she learned T_B, thanks to the binding property. Similarly, Alice can be confident that Bob could not manipulate the final outcome beforehand since he was unaware of T_A at the time of choosing T_B, because of the hiding property.

[2] Committing to small domain values, like age, can be easily revealed using brute force.

2.3 Merkle Trees

The Merkle tree, introduced by Ralph C. Merkle in 1982 [7], is a fundamental data structure with numerous use cases in computer science. Over time, it has become widely embraced in applications where safeguarding and validating data integrity are imperative. Such applications include, for instance, file systems, blockchains, or version control systems.

Usage Benefits

Merkle trees serve as an efficient tool for two types of problems. One is detecting whether some data D has changed compared to an earlier version of it. Hashing D directly and monitoring its digest for changes is a viable solution to this problem. However, using a Merkle tree provides a more efficient approach, particularly beneficial for large D.

This more efficient approach involves dividing D into smaller components or blocks $\{d_0, d_1, \ldots, d_i\}$, which then serve as individual hash inputs $h_i = H(t \mid d_i)$ with $t = 0$. Here, t acts as a type identifier that is concatenated to the data block beforehand. In practice, this identifier is usually one byte that is prepended to the hash function's input. It differentiates leaf nodes (`0x00`) from intermediate nodes (`0x01`) as a simple yet effective countermeasure against second preimage attacks [8]. Alternatively, one could also append the appropriate node level.

Figure 2.1 visualizes this concept. The data blocks or *leaf nodes* are shown in green, *intermediate nodes* are blue, and red marks the *root node*. Two subsequent or child digests serve again as input for their parent node on next level, e.g., h_0 and h_1 will together with $t = 1$ create $h_{01} = H(1 \mid h_0 \mid h_1)$. Continuing this hashing pattern of combining two lower node hashes to create a new one ultimately leads to the root

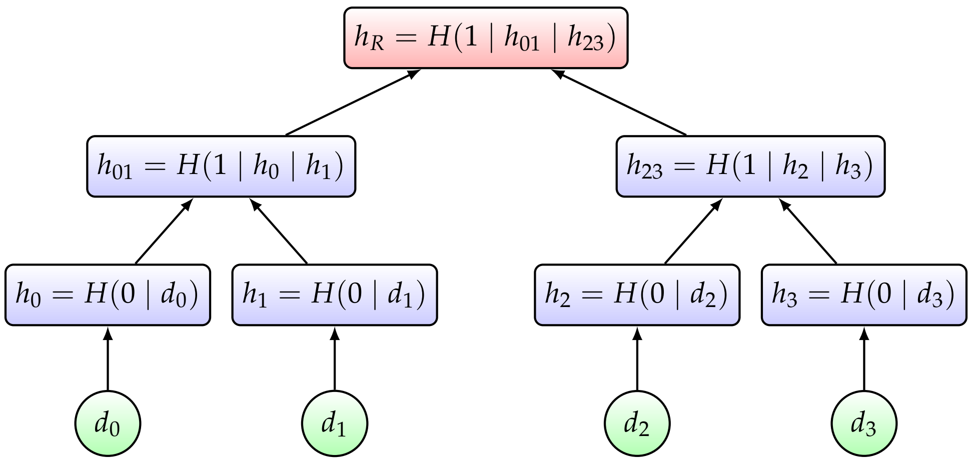


FIGURE 2.1: Merkle Tree

hash of the tree[3]. Little changes in D will inevitably result in a different root hash h_R which is why Merkle trees are a powerful tool for data change detection. This task is considerably more efficient than directly monitoring changes in D as the so-called *authentication path* from a leaf to the root only grows with $\mathcal{O}(\log n)$ and h_R remains of constant size.

The second type of problem where hash trees qualify as an ideal tool has to do with the question of whether or not D contains a specific d_i. If d_i is included in D, the hash chain from the leaf node using the minimum amount of intermediate nodes will result in the expected root node. Proving the inclusion of d_0 in Figure 2.1, for example, requires the elements $\{d_0, h_1, h_{23}\}$ in order to recompute $\{h_0^*, h_{01}^*, h_R^*\}$. A verifier $\mathcal{V}$ will be convinced that d_0 is indeed part of the hash tree with root h_R when $h_R^* = h_R$. The list of required elements is thus also known as *inclusion proof* and synonymous with authentication path. Verifying such a claim remains efficient even for large D thanks to the logarithmic growth behavior of binary trees.

Practical Example

So far, it has been shown how data structured in Merkle trees represent a powerful tool for scalable change detection and inclusion proofs, where hash functions play a central role owing to their distinctive properties. Furthermore, one can utilize hash functions to temporarily conceal information in an integrity-ensuring manner without the option of subsequent unnoticed alteration as required in commitment schemes.

Decentralized digital asset protocols such as, for instance, Bitcoin combine these two applications in a use case called *Simple Payment Verification (SPV)*. Nakamoto [9] sketched this use case, as shown in Figure 2.2.

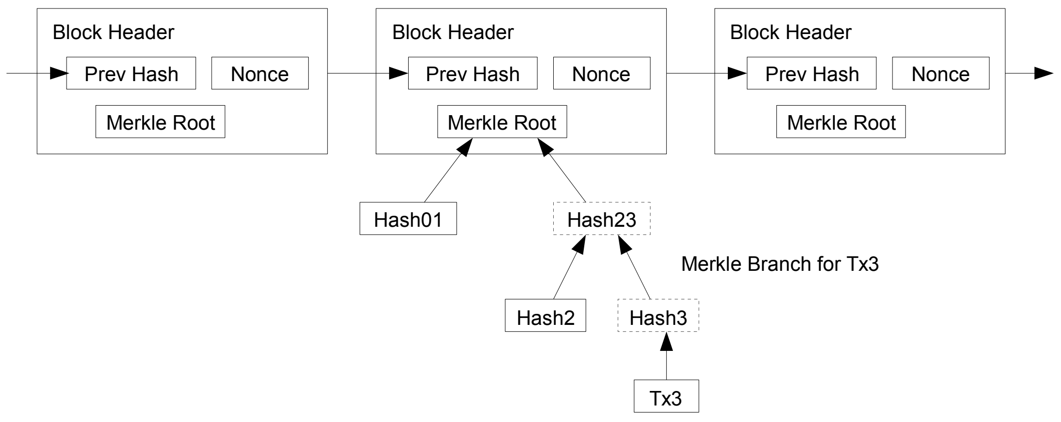


FIGURE 2.2: SPV in Bitcoin [9]

[3] The number of leaves should be a power of 2 to obtain a perfect tree or at least even. In the case of an odd leave count, the last leaf can be duplicated.

A transaction T_X transfers some coins to an address which is the hash value of the public component of the payee's asymmetric key. One considers T_X settled when it is included in a data block, organized as a Merkle tree, which is part of the longest[4] (block)chain available. Asking for an authentication path is a common way[5] to verify this inclusion. Figure 2.2 shows this process for `Tx3`, where $\mathcal{V}$ would recalculate the intermediate node hashes {`Hash3`,`Hash23`} with the help of {`Hash2`,`Hash01`} to verify the Merkle root hash eventually. When the recomputed root hash equals the expected one, $\mathcal{V}$ can be convinced that `Tx3` is settled.

[4] Technically speaking, settlement in Bitcoin is not defined by the longest chain of blocks but by the one that exhibits or proofs the highest amount of (hash) work.

[5] As conducted by so-called *light* nodes in Bitcoin. Alternatively, *full* nodes maintain a verified copy of the entire blockchain which allows them to autonomously verify transactions.

3 Zero-Knowledge Proof Systems

A general proof system is a tool such that $\mathcal{P}$ can convince $\mathcal{V}$ that a certain statement $\mathcal{S}$ is true. When $\mathcal{V}$ learns nothing except the fact that $\mathcal{S}$ is true (or not), the proof is considered zero-knowledge. This chapter outlines the essential characteristics of such systems based on the works of Thaler [10] and provides examples of practical applications as well as implementations.

3.1 Characteristics

Let $\mathcal{S}$ be the statement «*I know that* $x = 10$ *for* $13^x \equiv 2262 \pmod{7919}$ *is valid*». Here, x represents the *witness* of $\mathcal{S}$. A *trivial* proving strategy would involve simply disclosing the witness such that $\mathcal{V}$ can validate $\mathcal{S}$ directly. Large or secret witnesses as well as computationally expensive validation operations, however, ask for different proving strategies that function with partial or even non-disclosure of the witness. Meanwhile, the ability to convince $\mathcal{V}$ remains equally important. This work focuses on such *non-trivial* proof systems.

Describing Properties

ZKP systems exhibit three central properties, as shown below. In order to address these properties, certain cryptographic assumptions apply. This includes, for example, the existence of secure hash functions or the hardness of the so-called *Discrete Logarithm Problem (DLP)*[1].

- **Complete:** When every valid proof generated by an honest $\mathcal{P}$ over a true $\mathcal{S}$ convinces $\mathcal{V}$ then one considers this system as *complete*. In other words, completeness describes the property that $\mathcal{V}$ will always be convinced when presented with a truthful proof.
- **Sound:** Proof systems that are *sound* ensure the property that incorrect proofs or correct proofs of a false $\mathcal{S}$ always fail to convince $\mathcal{V}$. More specifically, one speaks of *statistical* or *information-theoretic* soundness when this property holds against a computational unbounded $\mathcal{P}$. Otherwise, the term *computational* soundness applies.

[1] DLP states that finding x for a given b in $a^x \equiv b \mod p$ is disproportionately harder than finding b for a given x [3].

- **Zero-Knowledge:** A proof that does not disclose any information about its witness while upholding the ability to convince $\mathcal{V}$ is said to be a ZKP. In other words, $\mathcal{V}$ should not learn anything but the boolean outcome of the proof verification process.

Two further properties of proof systems describe the level of interaction between $\mathcal{P}$ and $\mathcal{V}$ in order to end up with a complete and sound proof outcome. One distinguishes between protocols that prescribe active communication and ones that do not, as stated below.

- **Interactive:** *Interactive* proof systems require live communication between $\mathcal{P}$ and $\mathcal{V}$ in the context of repeated challenge-response rounds with random variables.
- **Non-Interactive:** When $\mathcal{V}$ can verify a proof without the need to exchange messages with $\mathcal{P}$, one speaks of *non-interactive* proof systems. This is especially powerful as proving and verifying can be performed independently from each other without any form of required dialogue.

Generally, every interactive system can be transformed into a non-interactive one using the Fiat-Shamir transformation [11] in the so-called *Random Oracle Model (ROM)*. It assumes that both $\mathcal{P}$ and $\mathcal{V}$ have access to a random function R, which for an input x deterministically returns a random value $R(x) = y$. In practice, R is usually a cryptographic hash function.

The core of back-and-forth communication in interactive protocols lies in the randomly chosen challenges to which only an honest $\mathcal{P}$ can consistently respond successfully in the long run. To transform such protocols into non-interactive ones, $\mathcal{P}$ simulates the challenges by prompting R for random challenges. All prompts inside this ROM will be embedded in the proof such that $\mathcal{V}$ can verify the integrity of these prompts at any later point in time non-interactively. The security model holds as it would be unfeasible for a computationally bounded $\mathcal{P}$ to use R in a self-serving manner facilitating the task to generate proofs dishonestly.

Last but not least, proof systems also differentiate themselves from each other based on the following two remaining properties.

- **Succinct:** A protocol that results in proof size and verification time that grows sublinear to the input size while upholding the security level is considered *succinct*. However, other definitions exist where authors associate *succinctness* with polylogarithmic or quasilinear growth. Figure 3.1 visualizes these different growth behaviors.
- **Transparent:** Proof systems that do not require a trusted setup are considered *transparent*. An example of a trusted setup is, for instance, the necessity of

a mutually created secret key that must be destroyed after the protocol's completion[2].

Argument vs. Proof of Knowledge

There exists a slight but important difference in the quality of a system that aims to convince $\mathcal{V}$ that $\mathcal{P}$ knows a valid witness. This difference is addressed below.

- **Argument of Knowledge:** In an argument of knowledge system, $\mathcal{P}$ is known to be computationally bounded and thus only able to create proofs that rely on polynomial-time complexity. This allows $\mathcal{V}$ to assume that the underlying cryptographic assumptions hold. In other words, there is no way for a computationally bounded $\mathcal{P}$ to convincingly prove knowledge of x without *actually* knowing x. A computationally unbounded $\mathcal{P}$, however, could break the underlying cryptographic assumptions and thus generate convincing proofs without actually knowing a valid witness. Argument of knowledge systems are also known as *argument systems* or *computationally bounded proof systems*.
- **Proof of Knowledge:** A proof of knowledge, on the other hand, represents a stronger artifact than an argument of knowledge since a computationally unbounded $\mathcal{P}$ would still fail to convince $\mathcal{V}$ with dishonest proofs despite the ability to break cryptographic assumptions. It follows that proof of knowledge systems possess higher security than argument systems, although the latter is more common in practice.

For the sake of simplicity, this work continues to primarily utilizes the term *proof systems* for both types of systems and uses the explicit specification where contextually meaningful.

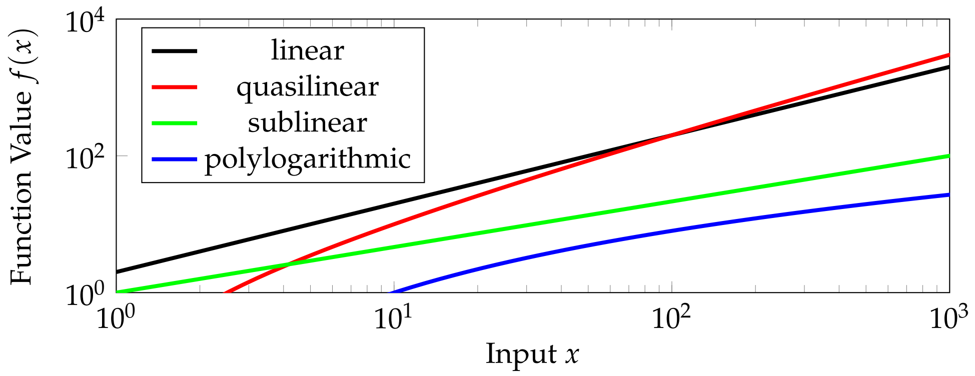


FIGURE 3.1: Different Growth Functions
For example, $f(x) = 2x$ grows linearly, $f(x) = log(x) * x$ quasilinearly, $f(x) = \sqrt{x}$ sublinearly, and $f(x) = log^k x$ polylogarithmically.

[2] Secret information generated during the setup phase of a cryptographic system that later could enable a potential adversaries to compromise the system's integrity is known as *toxic waste*.

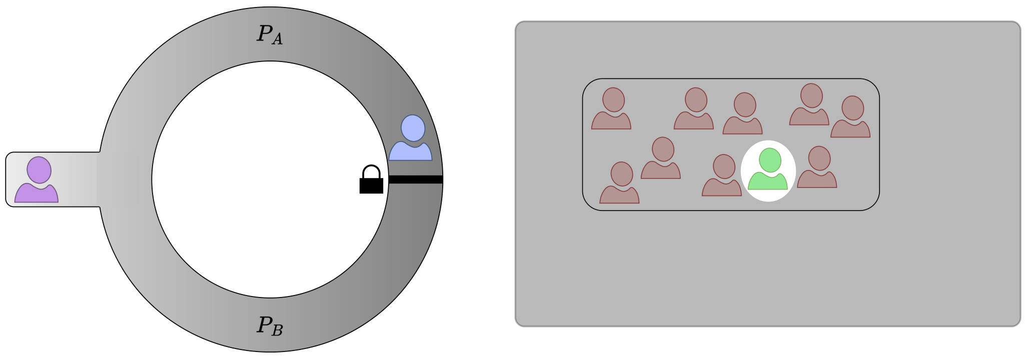
3.2 Applications

The field of ZKP applications is manifold. Beforehand, it is advisable to describe the dynamics of such proof systems by means of two famous analogies, which promote a more intuitive understanding.

Analogies

Subfigure (A) in Figure 3.2 sketches the so-called «Ali Baba Cave» analogy published by Quisquater et al. [12]. $\mathcal{P}$ (blue avatar) would like to prove to $\mathcal{V}$ (purple avatar) the possession of a key capable of unlocking a door (black bar) located in the rear part of a cave that is not visible from its entry and only reachable through either of the paths P_A or P_B. The protocol is designed as follows. $\mathcal{P}$ secretly enters the cave by randomly choosing either of the two possible path $C_P := \{ c \mid c \in \{P_A, P_B\} \}$ and waits in front of the locked door. $\mathcal{V}$ then challenges $\mathcal{P}$ to exit the cave using a specific path $C_V := \{ c \mid c \in \{P_A, P_B\} \}$ that $\mathcal{V}$ chooses randomly as well. Two possible explanations exist when $\mathcal{P}$ fulfills this request as demanded. Either $\mathcal{P}$ luckily chose the same path as $\mathcal{V}$ later challenged as the exit point, or they did not choose the same path and $\mathcal{P}$ is indeed able to pass the locked door. The latter implies that $\mathcal{P}$ possesses the appropriate door key.

In order to qualify this proof system as sound, this procedure is repeated n times to reduce the probability of a lucky $\mathcal{P}$ to 2^{-n}, which makes it an interactive proof. In other words, a cheating $\mathcal{P}$ will immediately be detected when $C_P \neq C_V$, which in turn becomes exponentially more likely to happen with increasing number of rounds, indicating soundness. Also, note that $\mathcal{V}$ never learns something about the key except the fact that $\mathcal{P}$ must know it to pass the door in $C_P \neq C_V$ rounds, which makes the protocol zero-knowledge.



(A) «Ali Baba Cave» (B) «Where's Waldo»

FIGURE 3.2: ZKP Analogies

The second ZKP analogy is known as «Where's Waldo» and was inspired by the eponymous comic book series. Subfigure (B) in Figure 3.2 models an image in which Waldo (green avatar) hides among many other people (red avatars). $\mathcal{P}$ arranges this image behind an opaque and much larger poster with an avatar-like silhouette hole in its middle such that Waldo's distinctive face appears.

By looking at the poster, $\mathcal{V}$ can verify that the silhouette hole indeed shows Waldo and that $\mathcal{P}$ is able to pinpoint his location, which results in convincing proof (completeness). The presentation of a wrong character would immediately be detected during this visual verification stage as Waldo's appearance is public knowledge (soundness). Meanwhile, $\mathcal{V}$ did not learn anything about the location as the image arrangement, i.e., image corners relative to Waldo, remains secret (zero-knowledge). This represents a non-interactive proof system, as the convincing $\mathcal{V}$ does not depend on a reiterating challenge-response interaction.

Real World Examples

The rather generic mechanisms demonstrated by the analogies above can be transferred to more practical implications. So-called *range proofs* allow to prove that a particular value, e.g., an age or salary lies in a specific interval. Questioning adulthood or solvency could be performed in a privacy-protecting manner while the verifying party must no longer manage the risk of storing sensitive *Customer Identifiable Data (CID)*. ZKPs could also be used to attest that a certain financial transaction is settled. Furthermore, they can help to maintain full data integrity in electronic voting systems where the validity of each individual vote must be checked. While not always zero-knowledge, proof systems fulfill the need for verifiable computations. For instance, a cloud provider executing complex computations for its clients can provide proof of the computational integrity of the result. [13]

Another broad area of applications can be found in the domain of cryptocurrencies [14]. One reason is that ZKPs represent a promising solution to mask sensitive details of a transaction. This includes, for example, the participants (payer and payee) or the amount. Besides leveraging privacy, the motivation also roots in the desire to further optimize storage efficiency. As previously mentioned, the partial or non-disclosure of (large) witnesses can positively impact the growth factor of such append-only data structures like blockchains without having to compromise on trustless verifiability thanks to ZKPs.

3.3 Implementations

This work focuses on two prominent implementation families of ZKP, both receiving significant research and development attention. Two prominent ZKP implementation families receive significant research and development attention. Ben-Sasson [15] surveys the terminological landscape of ZKP in greater details.

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) represents the first family of proof systems [16, 17]. They rely on a mutually created secret key required to verify a proof and thus require a trusted setup. The non-interactivity property of zkSNARKs makes them ideal candidates for use cases where $\mathcal{P}$ acts independent of $\mathcal{V}$ and vice versa. Succinct proof size relative to the security parameters promotes storage efficiency while the verification time remains constant. However, the trusted setup introduces two disadvantages. One is the possibility of leaking toxic waste, which facilitates the creation of proofs that were generated dishonestly and remain undetected during verification. A second disadvantage concerns the missing quantum-resistance due to necessary computational assumptions, making zkSNARKs post-quantum insecure.

Zero-Knowledge Scalable Transparent Argument of Knowledge (zkSTARK) defines the other implementation family [18]. Here, the public constraints of $\mathcal{S}$ with secret input x are translated into a so-called *Algebraic Intermediate Representation (AIR)* using arithmetic circuits. For example, $\mathcal{S}$ = «*I know a number* x *s.t.* $(x+3)*5 = 55$» exemplifies such a circuit in Figure 3.3, whereas $(\frac{55}{5} - 3)$ constraints a valid x. The computation is sequentially executed from left to right such that every operation, i.e., state change, can be registered in a trace table. The AIR serves as a basis to represent the constraints for a valid solution to $\mathcal{S}$ in a polynomial form known as *Compositional Polynomial (CP)*. This CP is a linear combination of rational functions that will be polynomials if and only if x is a valid input to $\mathcal{S}$. Besides verifying the correctness of the public constraints, $\mathcal{V}$ will repetitively challenge CP to test whether it is a polynomial or not. In the case of latter, $\mathcal{P}$ failed to prove $\mathcal{S}$.

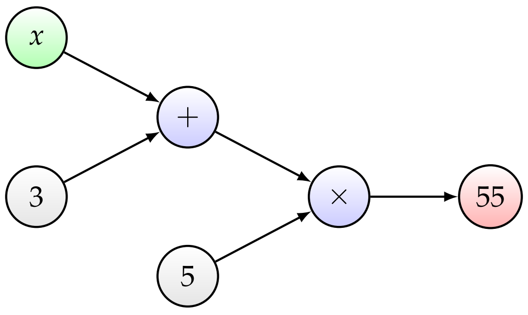


FIGURE 3.3: Example of an Arithmetic Circuit

Other than in zkSNARKs, zkSTARKs do not require a trusted setup and are thus considered transparent. The transparency trait, however, comes at the cost of larger proof sizes. Furthermore, zkSTARKs are considered post-quantum secure as they rely on cryptographic hash functions, which qualifies them as the preferred option in this work's proposed solution to zero-knowledge inclusion proofs.

As mentioned in Section 3.1, interactive proof systems can be deployed non-interactively using the Fiat-Shamir transformation and the ROM. Consequently, non-interactive zkSTARKs are also referred to as transparent zkSNARK, while transparent zkSNARKs with a succinct setup can also be considered as zkSTARKs. [15]

4 Implementing Proof of Inclusion

4.1 Objective

This section describes the proposed solution to the problem stated in Section 1.1 in greater detail and serves as a basis for the remainder of this document. Let the hash value $h = H(v \mid r)$ be a token that was derived by hashing a public identifier v concatenated with a secret blinding factor r. Also, let the set $\mathbb{M} = \{h_0, h_1, \dots, h_n\}$ be the leaf nodes of publicly available Merkle tree data structure with n leaves and the root hash $h_{\mathbb{M}}$.

$\mathcal{P}$ aims to prove to $\mathcal{V}$ the knowledge of r such that the digest of its concatenation with a given v is included in $\mathbb{M}$. Equation 4.1 states this objective formally.

$$\exists \ (r, v) \ \ s.t. \ \exists \ i \ \in \{h_0, h_1, \dots, h_n\} \text{ where } h_i = H(v \mid r) \tag{4.1}$$

To keep the ownership relation between v and $\mathcal{P}$ private, $\mathcal{V}$ must learn nothing except the fact that $\mathcal{P}$ indeed owns a suitable r that leads to token $h_i \in \mathbb{M}$.

4.2 Approach

Prior to the proof creation, $\mathcal{P}$ will query the publicly available Merkle tree of $\mathbb{M}$ to receive the authentication path P_{auth} that proves the inclusion of H in $\mathbb{M}$ with $H_{\mathbb{M}}$. Note that P_{auth} simply is an array of hash values and represents together with (v, r) secret information.

Next, $\mathcal{P}$ executes Algorithm 1 below to recompute all required hashes from the leaf to the root hash h^*_R. These computations represent the core of the ZKP proof as $\mathcal{V}$ will later verify that all computations were executed correctly and the resulting root hash matches the expected one. If this is the case, $\mathcal{V}$ will be convinced that $\mathcal{P}$ knows a suitable $(r, v) \ \ s.t. \ \ H(v \mid r) \in \mathbb{M}$.

The goal is to implement the proposed solution with the help of pre-existing software projects or libraries that provide the required environment for generating and validating a ZKP. This work classifies these projects as *automation frameworks* and presents a selection of them in the following section.

Algorithm 1 Inclusion Proof

```
Input: Private information ($v$, $r$, $P_{auth}$)
Output: Computed root hash $H_R^*$
1: procedure PROOF($v, r, P_{auth}$)
2:     $H \leftarrow h(\texttt{0x00} \mid v \mid r)$                    ▷ create leaf node hash
3:     $n \leftarrow len(P_{auth})$                         ▷ get size of authentication path
4:     $i \leftarrow 0$                         ▷ initialize authentication path access variable
5:     while $i < n$ do                         ▷ loop through all elements in $P_{auth}$
6:         if $P_{auth}[i].isRightHandSide$ then         ▷ decision on concatenation order
7:             $H \leftarrow h(\texttt{0x01} \mid H \mid P_{auth}[i])$         ▷ create intermediate node hash
8:         else
9:             $H \leftarrow h(\texttt{0x01} \mid P_{auth}[i] \mid H)$         ▷ create intermediate node hash
10:        $i \leftarrow i + 1$                         ▷ move to next $P_{auth}$ element
11:    $H_R^* \leftarrow H$                         ▷ last hash value represent root hash
12:    return $H_R^*$
```

4.3 Automation Frameworks

The automation frameworks presented in this chapter serve the purpose of facilitated practical implementation. Table 4.1 lists a selection of promising projects in this area. Following aspects play an essential role in order to be an attractive candidate for later implementation.

- **Maturity:** A mature automation framework qualifies through well-defined and well-tested code. Ideally, a third party has cryptographically audited the code to ensure its security.
- **Efficiency:** Proof creation and verification exhibit fast execution time thanks to performance-optimized code. The code is also designed to consume a relatively low amount of memory, which is important for practical adaptation on smartphones.
- **Usability:** The framework's use of common high-level programming languages makes it highly accessible and easy to integrate with other systems. Ideally, the framework is licensed to be free to use, modify, and distribute.
- **Maintainability:** The framework is under active development, with ongoing work on optimization and new features. It follows good software development practices and common patterns and is accompanied by thorough documentation.

The rest of this section describes two of the identified frameworks in Table 4.1 more closely.

Framework	Focus	License	Language	Resources
bellman	zkSNARK	Apache2.0 / MIT	Rust	[19]
circom	zkSNARK	GPL-3.0	Rust / C++ / Wasm	[20]
gnark	zkSNARK	Apache-2.0	Go	[21]
libiop	zkSNARK	MIT	C++	[22]
libsnark	zkSNARK	MIT	C++	[23]
libSTARK	zkSTARK	Custom	C++	[18, 24]
Marlin	zkSNARK	MIT	Rust	[25, 26]
Miden VM	zkSNARK	MIT	Rust	[27]
OpenZKP	zkSTARK	Apache-2.0	Rust	[28]
RISC Zero	zkSTARK	Apache-2.0	C++ / Rusts	[29, 30]
snarkjs	zkSNARK	GPL-3.0	JavaScript / Wasm	[31]
Valida	zkSTARK	Apache-2.0	Rust	[32]
Winterfell	zkSTARK	MIT	Rust	[33]
Zilch	zkSTARK	MIT	Rust	[34]
ZoKrates	zkSNARK	LGPL-3.0	Rust	[35]

TABLE 4.1: ZKP Automation Frameworks

RISC Zero

The RISC Zero or `risc0` project was published in February 2022 by a homonymous company founded the year before. The framework embraces an open-source virtual machine used to generate verifiable proofs (*receipts*) of any program that runs inside this virtual machine. It allows staging private inputs before running the computations inside the virtual machine in order to address the zero-knowledge property. Accordingly, one refers to it as *Zero-Knowledge Virtual Machine (zkVM)*.

More specifically, it allows to run arbitrary Rust or C++ programs inside the zkVM. To do so, the program (*guest method*) is compiled into an *Executable and Linkable Format (ELF)* binary file that runs on so-called `RISC-V` processors. Other than `x86-64` or `ARM` processors, `RISC-V` is a free and open *Instruction Set Architecture (ISA)* based on the principles of *Reduced Instruction Set Computing (RISC)* [36, 37]. Algorithm 1, for example, could serve as a guest method.

To create a proof, the *executor* runs the guest method inside the zkVM to create the *execution trace*, i.e., an array of all machine states over the execution time. The result of this execution, together with its public variables, is stored in a *journal* which in turn is *sealed* through a zkSTARK proof. The journal and the seal collectively form the *receipt* file[1]. The receipt serves to persuade $\mathcal{V}$ that the journal was genuinely produced through the anticipated circuit of the publicly known guest method. When this is the case, $\mathcal{V}$ can examine the logged computed result from the journal form its

[1] See the proof system sequence diagram for more detailed information.

Aspect	RISC Zero	Winterfell
Maturity	Relatively young project, tests and examples exist, not yet cryptographically audited.	Slightly older project, tests and examples exists, not yet cryptographically audited.
Efficiency	Overhead through zkVM, slow proving but fast verifying time.	Direct implementation, parallelization option, fast proving and verifying time.
Usability	Easy to use thanks to high-level programming language, abstracted complexity.	More complex to use due to required AIR, abstracted complexity.
Maintainability	Ongoing development, thoroughly documented.	Ongoing development, barely documented.

TABLE 4.2: Assessment of RISC Zero and Winterfell

final verdict. This could be, for instance, whether a recomputed Merkle root hash equals an anticipated root hash.

First tests with RISC Zero exemplified the straightforward setup. It also showcases the potency of encoding provable computations in familiar high-level programming languages (in this case, Rust). However, the added complexity of the zkVM makes it more challenging to comprehend the internal operations fully and is also expected to have a detrimental effect on efficiency. Nevertheless, this automation framework is a promising tool that warrants further investigation in future work.

Winterfell

Winterfell represents the second promising automation framework investigated so far. It was published by Meta[2] in April 2021 and implements a zkSTARK prover and verifier for arbitrary computations.

Other than RISC Zero, Winterfell does not rely on a virtual machine to create proofs. Instead, the underlying computation (e.g. Algorithm 1) must be provided as AIR directly. While this zkSTARK-friendly format reduces the accessibility for new developers, it promises faster proof times as no virtual machine overhead exists. Also, the proof generation process can be parallelized at the cost of higher memory consumption. Although this strategy does not harmonize with potential use cases in the smartphone domain, the framework aims to utilize parallelization in order to distribute the computational effort on multiple machines.

[2] Formerly known as Facebook.

To this point, only a few hands-on tests with Winterfell could be performed. As expected, the execution times to create proofs are significantly lower compared to RISC Zero. Therefore, it also represents a desirable tool worth further investigation in future work.

Table 4.2 captures a first-impression assessment of how RISC Zero and Winterfell address the previously stated desired framework aspects.

4.4 Minimal Viable Product

So far, a minimal viable product that adequately fulfills the aforementioned objective remains unrealized. The received experiences during this exploration phase steered the focus into promising directions that will be pursued in forthcoming endeavors.

The dynamic evolution of ZKPs, coupled with the increasing interest they generate, is set to impact their adoption within the software domain positively. This, in turn, will enhance accessibility for both novice developers and various utilization scenarios. While closely following this progression, the above-stated endeavors will also contribute to it.

5 Conclusion

This work presented a solution to the question of how to verifiably proof a specific token's validity without revealing which specific token it concerns using a zero-knowledge inclusion proof.

Preliminary, the basic concepts of cryptographic hash functions, commitment schemes, and Merkle trees were provided, followed by a general characterization of ZKPs. This includes discussing key aspects like completeness, soundness, zero-knowledge, and (non-)interactivity. For the promotion of intuitive understanding, the deliberation encompassed two well-known analogies as well as examples for real-world applications. Subsequently, two specific ZKP implementations were elaborated, namely zkSNARK and zkSTARK. Latter is considered post-quantum secure and thus preferred in this solution. Thanks to the Fiat-Shamir transformation, interactive proofs can also be transformed into non-interactive ones, which is required in this case.

Also, the work stated a formal description of the objective to prove knowledge of a private blinding factor such that the digest of its concatenation with a public token identifier is contained in a public set of valid tokens encoded as Merkle tree. The proposed implementation approach of the latter encompasses the employment of automation frameworks that serve as programmable interfaces for efficient zkSTARK proof creation and verification. Insights into the mechanics and initial usage experiences were provided for two such automation frameworks, namely *RISC Zero* and *Winterfell*. Both exhibit promising features that allow to create and verify proofs with an abstracted layer of complexity.

Future endeavors will focus on further investigating these automation frameworks. Once a minimum viable product is achieved using the most suitable framework, efforts will be directed toward optimizing both proof creation and verification runtime. Memory space optimization will also be a concurrent area of attention along with these performance enhancements, as it represents a critical requirement for mobile applications Ultimately, the chosen automation framework and the implemented solution must undergo a cryptographic audit before deployment in operational systems.

A Post-Quantum Cryptography

Recent advances in quantum computing introduce new threats to traditional cryptographic primitives. The security of this primitives primarily rests on the asymmetrical complexity inherent in mathematical functions and their inverses. So far, the yardstick for measuring this complexity has been linked to binary-based computer systems. Quantum computers, however, transcend this binary restriction unlocking enhanced computational capabilities that now poses a new threat to traditional cryptography.

The field of PQC emerged to address this threat. It encompasses the research in new cryptographic primitives designed to withstand potential quantum computing attacks while maintaining robustness and practicality in traditional computing environments. The key innovation of these new algorithms is that they rely on different mathematical problems that are hard to solve for both binary and quantum computers. PQC aims to foster the development of new provable secure protocols with the ultimate objective of introducing novel implementation standards for cryptographic applications. Ideally, this undertaking outpaces the progression of quantum computing.

For example, the DLP features such an asymmetrical difficulty and is thus widely employed in cryptographic primitives nowadays. The security of these primitives relies on computing the discrete logarithm of a large number modulo a prime is believed to be hard for classical computers. However, quantum computers have the potential to break these primitives efficiently using *Shor's algorithm* [38].

Shor's algorithm facilitates factoring integers and computing the discrete logarithms by exploiting the quantum properties of superposition and entanglement to perform computations in parallel. By running it on a quantum computer, an attacker could theoretically defeat the security of cryptographic primitives that rely on the DLP.

Starting in 2016, the *National Institute of Standards and Technology (NIST)* leads the current standardization process of PQC algorithms[1]. This community project aims to evaluate the security and efficiency of several proposed algorithms in a round-based manner to determine superior candidates. NIST plans to proclaim the finalist algorithm as a new standard in the near future.

Algorithm Types

The security of PQC algorithms rely on a different set of mathematical problems that cannot be solved efficiently by both traditional and quantum computers. The following paragraphs provide an overview of four popular types of such algorithm based on Aumasson and Green [5].

- **Lattice-Based:** These algorithms utilize lattices, i.e., mathematical structures that describe a repeating pattern in an n-dimensional space, to construct the problem of finding optimal combinations of given vectors to reach a desired target point in that lattice. One generalizes such problems as *Closest Vector Problem (CVP)*, whereas *Short Integer Solution (SIS)*, *Learning With Errors (LWE)*, or *Learning With Rounding (LWR)* represent more specific examples of it. It is worth mentioning that the CVP serves only as a secure cryptographic tool for traditional and quantum computers in its hardest instance. To create practical cryptosystems, however, hardness must also be guaranteed in average cases.
- **Code-Based:** These types of cryptographic algorithms were inspired by so-called *error-correcting codes*, which denote the theory of making bit transmissions over noisy channels more robust to receiving errors. In general, this is achieved by adding redundancy to the transmitted bit sequence on the sender side, which eventually helps the receiving side autonomously correct errors to a certain degree. A popular class of error-correcting codes is known as *linear codes*. Here the transmitted bit sequence is encoded as a vector v to be multiplied with a matrix G to receive the code word $w = v \cdot G$. Depending on how G was instantiated, the recipient can correct up to t-bit errors in w to recover v. Code-based cryptographic algorithms such as, for example, the *McEliece* [39] scheme, compose a publicly known G of multiple secret matrices acting as the private key. The security of code-based cryptography resides on the NP-hard problem of decoding a linear code, meaning that it is computationally infeasible to solve using a classical or quantum computer within a reasonable amount of time[2].

[1] See NIST's project webpage for current state and additional information.

[2] See article by Aaronson [40] on quantum computers limits relating computational problem classes.

- **Multivariate:** The security of this cryptographic algorithm type relies on systems of equations with multiple unknowns. More concretely, it depends on another NP-hard problem that can be constructed by means of a random quadratic system of equations known as *multivariate quadratic equations*. Since the hardness of such problems relies on various parameters that must be chosen wisely, multivariate cryptography is not widely used in productive systems.
- **Hash-Based:** While the security of the previous algorithm types relies on their mathematical hardness, this type utilizes the security of collision-resistant cryptographic hash functions. Section 2.1 discusses these functions in detail. As of the time of writing, the properties of such one-way functions are not endangered by quantum computers. Consequently, systems that rely on the difficulty of finding collisions, such as *Winternitz* [41] or *SPHINCS* [42, 43], are considered quantum-resistant. In other words, the security of hash-based signature schemes stems from the fact that quantum computers cannot efficiently find collisions in cryptographic hash functions.

Notable Implementations

The current standardization effort by NIST has brought forward several PQC algorithms with the potential to become state-of-the-art cryptography tools in future. Table A.1 provides an overview of notable candidates in this context.

Generally, one categorizes these algorithms based on their indented core function. One category describes public-key encryption and key establishment algorithms, also known as *Key Encapsulation Mechanism (KEM)*. Such a mechanism represents a public-key or asymmetric encryption scheme used to encrypt and decrypt data, usually for exchanging a symmetric key [6]. The other category includes algorithms used to proof data authenticity through signatures, known as *Digital Signature Algorithm (DSA)*. Such algorithms allow that a signer to sign any piece of data using a private key so that any other party can verify this signature to the corresponding data using the associated public key.

Cryptographic Agility

As stated earlier, cryptographic primitives are omnipresent in today's digitalized world. Basically, every user activity on the internet, such as requesting a webpage or login into a web platform, relies on such primitives. Let alone the fundamental Transport Layer Security (TLS) protocol which is responsible for exchanging information between computing devices.

Algorithm	Function	Type	Website	Reference
BIKE	KEM	Code-Based	↗	[44]
Classic McEliece	KEM	Code-Based	↗	[39, 45]
CRYSTALS-Kyber	KEM	Lattice-Based (LWE)	↗	[46]
CRYSTALS-Dilithium	DSA	Lattice-Based	↗	[47]
Falcon	DSA	Lattice-Based	↗	[48]
NTRU	KEM	Lattice-Based	↗	[49]
Rainbow	DSA	Multivariate	↗	[50]
SABER	KEM	Lattice-Based	↗	[51]
SPHINCS+	DSA	Hash-Based	↗	[43]

TABLE A.1: Notable PQC Algorithms.

Establishing new standards regarding quantum-resistant algorithms alone does not suffice to mitigate quantum attacks. Equally vital is the ability of systems to flexibly accommodate evolving security needs within the realm of cryptography. The term *cryptographic agility* or *crypto-agility* refers to this ability. All actors in the digitalized landscape must inevitably confront the task of evaluating their individual exposure to cryptographic systems in order to remain cryptographically agile and thus secure.

Bibliography

[1] S. J. Alsunaidi and F. A. Alhaidari. “A Survey of Consensus Algorithms for Blockchain Technology”. In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. 2019 International Conference on Computer and Information Sciences (ICCIS). Sakaka, Saudi Arabia: IEEE, Apr. 2019, pp. 1–6. ISBN: 978-1-5386-8125-1. DOI: 10.1109/ICCISci.2019.8716424.

[2] E. Androulaki et al. “Evaluating User Privacy in Bitcoin”. In: *Financial Cryptography and Data Security*. Ed. by A.-R. Sadeghi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 34–51. ISBN: 978-3-642-39884-1. DOI: 10.1007/978-3-642-39884-1_4.

[3] R. Bögli. “A Security Focused Outline on Bitcoin Wallets”. Focus Project 1. Rapperswil-Jona: OST Ostschweizer Fachhochschule, 2023. URL: https://eprints.ost.ch/id/eprint/1103/.

[4] W. Stallings. *Network Security Essentials: Applications and Standards*. 4th ed. Boston: Prentice Hall, 2011. 417 pp. ISBN: 978-0-13-610805-4.

[5] J.-P. Aumasson and M. D. Green. *Serious Cryptography: A Practical Introduction to Modern Encryption*. San Francisco: No Starch Press, 2017. 282 pp. ISBN: 978-1-59327-826-7.

[6] J.-P. Aumasson. *Crypto Dictionary: 500 Tasty Tidbits for the Curious Cryptographer*. San Francisco: No Starch Press, 2021. 145 pp. ISBN: 978-1-71850-140-9.

[7] R. C. Merkle. “Method of providing digital signatures”. U.S. pat. 4309569A. Univ Leland Stanford Junior. Jan. 5, 1982.

[8] B. Laurie, A. Langley, and E. Kasper. *Certificate Transparency*. RFC6962. RFC Editor, June 2013, RFC6962. DOI: 10.17487/rfc6962.

[9] S. Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. Oct. 31, 2008.

[10] J. Thaler. “Proofs, Arguments, and Zero-Knowledge”. In: *Foundations and Trends® in Privacy and Security* 4.2–4 (2022), pp. 117–660.

[11] A. Fiat and A. Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Advances in Cryptology — CRYPTO’ 86*. Ed. by A. M. Odlyzko. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1987, pp. 186–194. ISBN: 978-3-540-47721-1. DOI: 10.1007/3-540-47721-7_12.

[12] J.-J. Quisquater et al. “How to Explain Zero-Knowledge Protocols to Your Children”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Ed. by G. Brassard. Vol. 435. New York, NY: Springer New York, 1990, pp. 628–631. ISBN: 978-0-387-97317-3. DOI: 10.1007/0-387-34805-0_60.

[13] A. M. Tran. “Theoretical and practical introduction to ZK-SNARKs and ZK-STARKs”. MA thesis. Masaryk University, Faculty of Informatics, 2022. URL: https://is.muni.cz/th/ovl3c/.

[14] E. Ben-Sasson et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *2014 IEEE Symposium on Security and Privacy*. 2014 IEEE Symposium on Security and Privacy. May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[15] E. Ben-Sasson. *A Cambrian Explosion of Crypto Proofs*. NAKAMOTO. Jan. 8, 2020. URL: https://nakamoto.com/cambrian-explosion-of-crypto-proofs/ (visited on 08/04/2023).

[16] N. Bitansky et al. “From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS ’12. New York, NY, USA: Association for Computing Machinery, Jan. 8, 2012, pp. 326–349. ISBN: 978-1-4503-1115-1. DOI: 10.1145/2090236.2090263.

[17] E. Ben-Sasson et al. “Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture”. In: *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014, pp. 781–796. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.

[18] E. Ben-Sasson et al. *Scalable, transparent, and post-quantum secure computational integrity*. 2018. URL: https://eprint.iacr.org/2018/046. preprint.

[19] *bellman*. Version 0.13.0. Zero-knowledge Cryptography in Rust, May 7, 2022. URL: https://github.com/zkcrypto/bellman (visited on 08/05/2023).

[20] *circom*. Version 2.1.6. iden3, June 22, 2023. URL: https://github.com/iden3/circom (visited on 08/05/2023).

[21] G. Botrel et al. *ConsenSys/gnark*. Version 0.7.0. Zenodo, Mar. 2022. URL: https://doi.org/10.5281/zenodo.5819104.

[22] *libiop*. Version 0.2.0. SCIPR Lab, Aug. 13, 2020. URL: https://github.com/scipr-lab/libiop (visited on 08/05/2023).

[23] *libsnark*. Version 20140603. SCIPR Lab, June 3, 2014. URL: https://github.com/scipr-lab/libsnark (visited on 08/05/2023).

[24] E. Ben-Sasson et al. *libSTARK*. 2018. URL: https://github.com/elibensasson/libSTARK (visited on 08/05/2023).

[25] *Marlin*. Version 0.3.0. arkworks, June 7, 2021. URL: https://github.com/arkworks-rs/marlin (visited on 08/05/2023).

[26] A. Chiesa et al. *Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS*. 2019. URL: https://eprint.iacr.org/2019/1047. preprint.

[27] *Miden Virtual Machine*. Version 0.6.1. Polygon Miden, June 30, 2023. URL: https://github.com/0xPolygonMiden/miden-vm (visited on 08/05/2023).

[28] *OpenZKP*. ZeroEx (0x), 2020. URL: https://github.com/0xProject/OpenZKP (visited on 08/05/2023).

[29] *RISC Zero*. Version 0.16.1. RISC Zero, July 13, 2023. URL: https://github.com/risc0/risc0 (visited on 08/05/2023).

[30] J. Bruestle and P. Gafni. "RISC Zero zkVM: Scalable, Transparent Arguments of RISC-V Integrity". Draft. July 29, 2023. URL: https://www.risczero.com/proof-system-in-detail.pdf (visited on 08/05/2023).

[31] *snarkjs*. Version 0.7.0. iden3, May 24, 2023. URL: https://github.com/iden3/snarkjs (visited on 08/05/2023).

[32] *Valida*. Valida. URL: https://github.com/valida-xyz/valida (visited on 08/05/2023).

[33] *Winterfell*. Version 0.6.4. Meta, May 26, 2023. URL: https://github.com/facebook/winterfell (visited on 08/05/2023).

[34] *Zilch*. Trustworthy Computing Group. URL: https://github.com/TrustworthyComputing/Zilch (visited on 08/05/2023).

[35] *ZoKrates*. Version 0.8.7. ZoKrates, Apr. 22, 2023. URL: https://github.com/Zokrates/ZoKrates (visited on 08/05/2023).

[36] A. Waterman et al. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1*. UCB/EECS-2016-118. EECS Department, University of California, Berkeley, May 31, 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html.

[37] A. Waterman et al. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.9*. UCB/EECS-2016-129. EECS Department, University of California, Berkeley, July 8, 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html.

[38] P. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 35th Annual Symposium on Foundations of Computer Science. Santa Fe, NM, USA: IEEE Comput. Soc. Press, 1994, pp. 124–134. ISBN: 978-0-8186-6580-6. DOI: 10.1109/SFCS.1994.365700.

[39] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *Coding Thv* 4244 (1978), pp. 114–116.

[40] S. Aaronson. "THE LIMITS OF Quantum". In: *Scientific American* 298.3 (2008), pp. 62–69. ISSN: 0036-8733. JSTOR: 26000518. URL: https://www.jstor.org/stable/26000518.

[41] R. C. Merkle. "A Certified Digital Signature". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by G. Brassard. Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 218–238. ISBN: 978-0-387-34805-6. DOI: 10.1007/0-387-34805-0_21.

[42] D. J. Bernstein et al. "SPHINCS: Practical Stateless Hash-Based Signatures". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2015, pp. 368–397. ISBN: 978-3-662-46800-5. DOI: 10.1007/978-3-662-46800-5_15.

[43] D. J. Bernstein et al. "The SPHINCS+ Signature Framework". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 6, 2019, pp. 2129–2146. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363229.

[44] N. Aragon et al. "BIKE: Bit Flipping Key Encapsulation". In: (2017).

[45] D. J. Bernstein et al. "Classic McEliece: conservative code-based cryptography". In: *NIST submissions* (2017).

[46] J. Bos et al. "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018 IEEE European Symposium on Security and Privacy (EuroS&P). London: IEEE, Apr. 2018, pp. 353–367. ISBN: 978-1-5386-4228-3. DOI: 10.1109/EuroSP.2018.00032.

[47] L. Ducas et al. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Feb. 14, 2018), pp. 238–268. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i1.238-268.

[48] P.-A. Fouque et al. "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU". In: *Submission to the NIST's post-quantum cryptography standardization process* 36.5 (2018).

[49] C. Chen et al. "NTRU - Algorithm Specifications and Supporting Documentation". In: *Brown University and Onboard security company, Wilmington USA* (Mar. 30, 2019).

[50] J. Ding and D. Schmidt. "Rainbow, a New Multivariable Polynomial Signature Scheme". In: *Applied Cryptography and Network Security*. Ed. by J. Ioannidis, A. Keromytis, and M. Yung. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 164–175. ISBN: 978-3-540-31542-1. DOI: 10.1007/11496137_12.

[51] J.-P. D'Anvers et al. "Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM". In: *Progress in Cryptology – AFRICACRYPT 2018*. Ed. by A. Joux, A. Nitaj, and T. Rachidi. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 282–305. ISBN: 978-3-319-89339-6. DOI: `10.1007/978-3-319-89339-6_16`.

List of Abbreviations

AIR	Algebraic Intermediate Representation	13
CID	Customer Identifiable Data	12
CP	Compositional Polynomial	13
CVP	Closest Vector Problem	22
DLP	Discrete Logarithm Problem	8
DLT	Distributed Ledger Technology	1
DSA	Digital Signature Algorithm	23
ELF	Executable and Linkable Format	17
ISA	Instruction Set Architecture	17
KEM	Key Encapsulation Mechanism	23
LWE	Learning With Errors	22
LWR	Learning With Rounding	22
NIST	National Institute of Standards and Technology	22
PQC	Post-Quantum Cryptography	1
RISC	Reduced Instruction Set Computing	17
ROM	Random Oracle Model	9
SIS	Short Integer Solution	22
SPV	Simple Payment Verification	6
TLS	Transport Layer Security	23
ZKP	Zero-Knowledge Proof	2
zkSNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge	13
zkSTARK	Zero-Knowledge Scalable Transparent Argument of Knowledge	13
zkVM	Zero-Knowledge Virtual Machine	17

List of Figures

List of Tables