FEDERAL RESERVE BANK of ST. LOUIS
CENTRAL TO AMERICA'S ECONOMY®

An Introduction to Zero-Knowledge Proofs in Blockchains and Economics

Aleksander Berentsen, Jeremias Lenzi, and Remo Nyffenegger

Abstract

With a zero-knowledge proof (ZKP), a party can prove that a statement is true without revealing any information except for whether it is indeed true or not. The obvious benefit is privacy since the prover does not need to reveal any additional information, and the second benefit is that it can significantly reduce the cost of verifying the correctness of a statement. ZKPs are increasingly adopted in blockchain applications, where privacy and efficiency still have a lot of room for improvement. While it is expected that ZKP technology will also become ubiquitous in many other areas, the term remains cryptic to many people without a computer science background. In this review article, we shed light on what ZKPs are and how they improve privacy and efficiency and describe applications for blockchains and other use cases.

JEL codes: E3, E52, E62, F10, O10

Federal Reserve Bank of St. Louis *Review*, Fourth Quarter 2023, *105*(4), pp. 280-94.
https://doi.org/10.20955/r.105.280-94

1. INTRODUCTION

Efficiency in economics means that scarce resources should not be wasted. This statement is true for manufacturing as well as for computations since they use real resources such as hardware, electricity, or human capital. There are various applications in which a computation's correctness must be verified by many other parties. For example, in blockchains, decentralization requires that many network participants recompute the correctness of each block that is appended to the chain.[1] Obviously, reexecuting the same computations is inefficient because it involves using computational resources repeatedly.

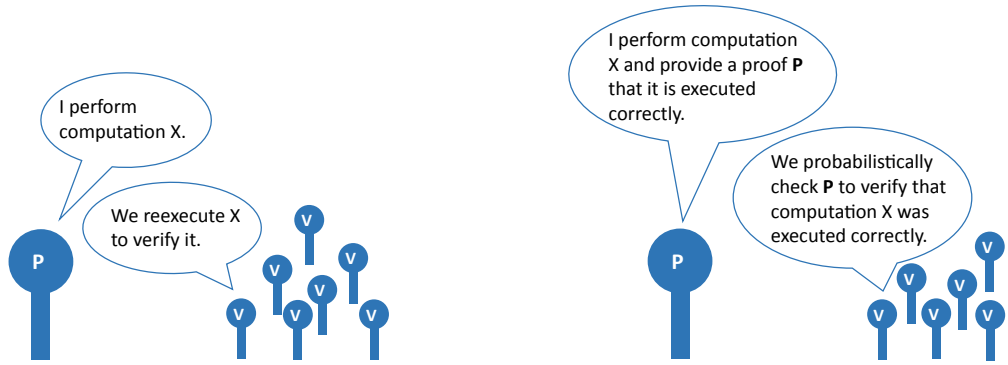
By using a zero-knowledge proof (ZKP), a party can prove to other parties that a computation was executed correctly. There is no need to replicate the computation—only the proof needs to be verified. Ideally, verifying a ZKP needs significantly less resources than reexecuting the computation. This benefit is illustrated in Figure 1; note that the efficiency gains of ZKPs increase linearly in the number of validators.

The second and more obvious benefit of ZKP technology is privacy. By using a ZKP, one can prove the correctness of a computation without revealing any additional information except for whether it is indeed correct or not. For example, a blockchain user can prove that he is indeed allowed to make a payment without revealing his identity to the network. Existing applications are the privacy-protecting cryptocurrency Zcash and the Tornado cash protocol on Ethereum (see Nadler and Schaer, 2023). The privacy and confidentiality

1. To learn more about blockchain technology, see Schaer and Berentsen, 2020.

Aleksander Berentsen is a professor of economic theory at the University of Basel and a research fellow at the Federal Reserve Bank of St. Louis. Jeremias Lenzi is a PhD student at the University of Basel. Remo Nyffenegger is a PhD student and research assistant at the Center for Innovative Finance at the University of Basel.

Figure 1
Reexecution vs. Using a Proof



A. Multiple reexecution B. Using a proof

of data is also important outside of blockchains. Two examples are a person who wants to prove that she voted without revealing her vote, or a company that wants to prove its solvency without revealing its balance sheet.

The theoretical concept of ZKPs was introduced in the late 80s by Goldwasser, Micali, and Rackoff, 1989. Conceptually, there are many different use cases, but none of them have become economically important. This has changed with the advent of blockchain technology, where ZKP technology has been integrated in some applications. ZKP research is rapidly expanding as demonstrated by the increasing number of articles about the technology (see Burger et al., 2022 for an overview).

Most research on ZKPs targets an audience with a computer science or mathematics background, and there is a lack of a comprehensive but intuitive introduction into the topic. This review article fills this gap by providing an accessible but extensive introduction into zero knowledge proofs and their applications. Furthermore, in Berentsen, Lenzi, and Nyffenegger, 2022 we provide a comprehensive example of a ZKP that includes more advanced math and comes with an accompanying Python script.

The rest of the article is organized as follows. In Section 2 we discuss two very simple and intuitive examples of a ZKP, while in Section 3 we dive more into the details. In Sections 4 and 5 we discuss the application of ZKPs for blockchains and in general, and in Section 6 we conclude.

2. INTUITIVE EXAMPLES OF A ZKP

From a technological perspective, we can differentiate two types of proofs, interactive and non-interactive ones. An interactive proof requires some bilateral interaction between the prover and verifier; i.e., the verifier sends personalized random queries/challenges to the prover, who then responds to these queries. This process is repeated over several rounds until the verifier is convinced of the correctness of the proof with a sufficiently high probability. Each additional verifier who also wants to check the proof has, again, a bilateral interaction with the prover and sends personalized challenges over several rounds.

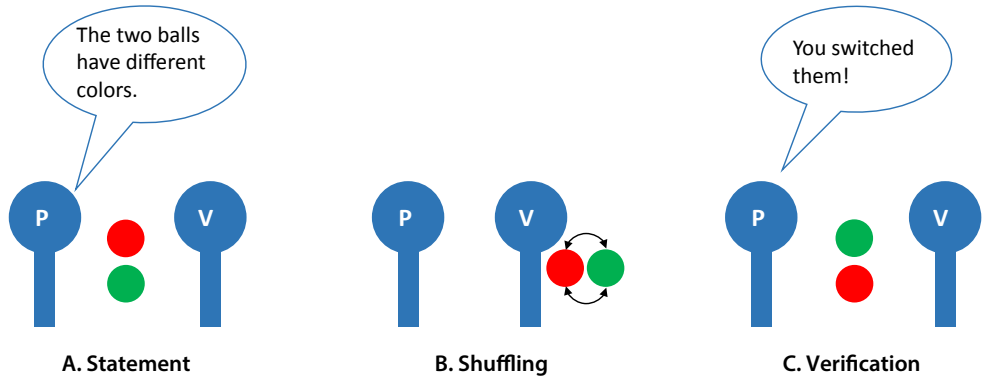
Thus, an interactive proof can be inefficient or infeasible if many verifiers are involved.[2] In a non-interactive proof, no interaction is required and the prover provides only one proving object (e.g., a string) that anybody can verify at any time. This is useful in, for example, blockchain applications. Below, we provide one simple example for both types of proofs: first for an interactive proof and then for a non-interactive one.

2.1 Two Balls and the Colorblind Friend

To get a first intuition of a ZKP, we will discuss a trivial example by Chalkias and Hearn, 2019, which is called "two balls and the colorblind friend." Imagine two friends, Peggy and Victor. Peggy is an expert in ZKPs and wants to teach the concept to Victor. She knows that Victor is colorblind and hands him two identical balls except one is green and the other is red. They agree that the balls are exactly identical apart from their color, which Victor cannot evaluate because of his colorblindness. Peggy (the prover) claims that she can prove to Victor (the verifier) that the balls have indeed different colors without revealing which is which (i.e., the zero-knowledge part). This example is illustrated in Figure 2A.

2. There are two main reasons: generating the proof repeatedly increases the overall computing power used, and a prover must be available/online every time a verifier wants to verify the proof.

Figure 2
The Two Balls and the Colorblind Friend



The proof in this example works like a small game. Victor takes the two balls, puts them behind his back, shuffles them (Figure 2B), and then again shows them to Peggy (Figure 2C). He knows whether he has switched the balls or not since he shuffled them consciously. Furthermore, Peggy knows as well because she can differentiate between the colors. She then tells him that he switched, which is an indication that the balls have to be differently colored.

The proof, however, does not end here. Victor might think that Peggy was just lucky and correctly guessed by chance. The probability of doing so is indeed 50 percent, and therefore he repeats the experiment. An excellent mathematician, he knows that the probability of having two balls with different colors is $P(\text{balls have two colors}) = 1 - 0.5^n$, where n is the number of times they repeat the game. Already after $n = 10$ iterations, the probability that the balls are indeed differently colored is P = 99.9 percent. Thus, Peggy can prove her claim that the balls have different colors without revealing which ball has which color. If she intends to convince other people of the proof, she would need to repeat the same steps for each verifier.

2.2 Where's Waldo?

We all know the *Where's Waldo* books with the illustrations of dozens of people in a chaotic environment, where the goal is to find Waldo in the red and white striped jumper. Peter is a very talented Where's Waldo player. After the release of a new book, he wants to prove to the *Where's Waldo*-community that he has found Waldo in an especially difficult illustration. Obviously, he does not want to reveal where exactly Waldo is. We slightly extend the concept of Naor, Naor, and Reingold, 1999 to show how Peter can prove the statement "I have found Waldo" using a form of a non-interactive ZKP proof.

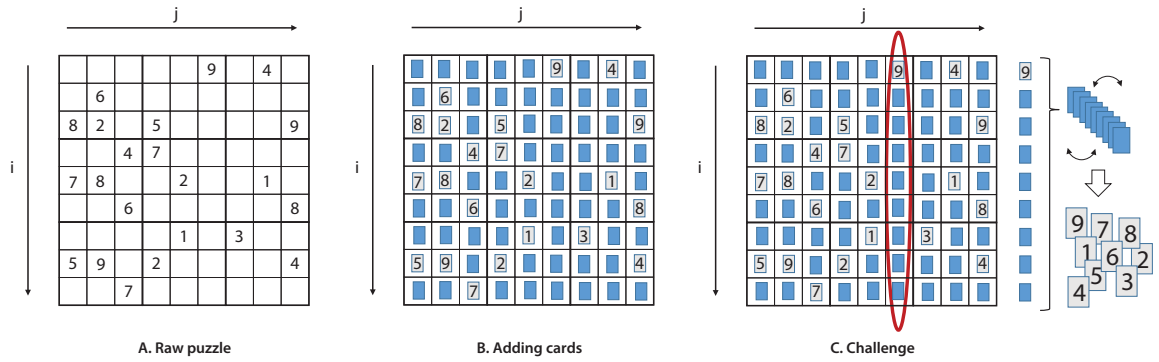
Before the actual proof, the community and Peter agree on the rules.[3] They define an empty room with nothing in it except for the illustration, a photocopy of it, and a pair of scissors. The proof begins with Peter being searched before entering the room to make sure that he does not take anything with him. This arrangement is called an initial set-up or ceremony. All community members attending the ceremony can verify themselves that Peter cannot cheat because the room indeed only contains the illustration and a pair of scissors and Peter was searched properly. He could only cheat if all community members attending the set-up collude. Hence, a community member who did not attend the ceremony can also be convinced of the proof if at least one attendant in the ceremony is honest.[4]

Next, Peter is alone in the room and cuts out Waldo from the photocopy and puts him next to the illustration. He then leaves the room, taking the remaining parts of the photocopy with him. The cropped image of Waldo is the only object that Peter provides as a proof. Now each member of the community can verify the proof by entering the room and checking whether the statement that Peter found Waldo is true. The proof is non-interactive since the verifiers do not need to interact with Peter but can just verify the cropped image.

3. In more complex examples, the prover and the verifier must agree up front on certain conditions that need to hold for the proof to be true.

4. Also, some types of real-world, non-interactive ZKP implementations require these ceremonies. As in our example, as long as at least one party in the ceremony acts benevolently, the subsequent proofs can be trusted. We describe this in more detail in Section 3.3.

Figure 3
Sudoku



3. ADVANCED EXAMPLE AND SOME THEORY

After providing some intuition on ZKPs, we now dive in deeper. A ZKP must fulfill three properties:

- Zero knowledge: A malicious[5] verifier cannot extract any information except for the statement being true or not.
- Completeness: A prover can convince the verifier (who follows the protocol) of a true statement.
- Soundness: A malicious prover cannot convince the verifier of a false statement with a sufficiently high probability.

The examples above satisfy all properties. The two-ball example is zero knowledge because the game structure (the protocol) does not reveal to Victor which ball has which color and he cannot derive it in a different way because of his colorblindness. Completeness is achieved because Peggy can convince Victor that the balls are indeed differently colored. With respect to soundness, a false statement would be that Peggy claims that the balls have different colors even though they are actually identical in that perspective as well.

By playing the game with identically colored balls, Peggy has in each round a 50 percent chance of guessing correctly whether Victor switched the two balls or not. After 10 rounds, the probability of guessing correctly in each round is only 0.1 percent. Thus, Peggy cannot convince Victor of the false statement with a sufficiently high probability, and also soundness is satisfied. The Where's Waldo example is zero knowledge because the verifiers cannot extract any information about where Waldo is, it is complete because Peter can show where Waldo is, and it is sound because showing another character than Waldo would not convince the verifiers.

3.1 Sudoku

We now introduce a more advanced interactive proof of a Sudoku puzzle that follows Gradwohl et al., 2009. The main purpose is to gain more intuition without showing very complicated math. But compared to the examples before, we incorporate some additional features that a ZKP normally entails.

Remember that in a Sudoku game the player is given a 9×9 grid in which some cells are filled with a digit. The objective is to fill the grid such that each row, each column, and each of the nine 3×3 boxes contain all digits from 1 to 9. We denote the row as $i \in \{1, 2, \ldots, 9\}$, the column as $j \in \{1, 2, \ldots, 9\}$, and the respective digit in cell $[i, j]$ as $k \in \{1, 2, \ldots, 9\}$. Furthermore, we denote the boxes as $b \in \{1, 2, \ldots, 9\}$, counting from left to right and top to bottom.

Figure 3A illustrates an example of a Sudoku puzzle. Using the notation above, we can describe the puzzle by a set of vectors of the type (i, j, k). An example of a vector would be $(3, 4, 5)$, which means that in row 3 and column 4, the entry is 5. The puzzle in Figure 3A can thus be represented by a set of 22 vectors, one for each number in the grid.

Peggy is in a Sudoku competition with her friend Victor, and whoever solves the puzzle first wins. She is the first to find a solution and wants to prove to Victor that she solved it without revealing the solution; i.e., she wants to prove the statement "I have a solution to this specific Sudoku game," to which the answer is either true or false. To do so, she proposes to apply a form of a ZKP that only requires the puzzle to be in a big format and some decks of playing cards.

5. Malicious means that a party tries to cheat or trick the protocol.

The first step is to represent a filled puzzle in a 9-by-9-by-9 cube, denoted as x that contains binary values in each cell:

$$x(i,j,k) = \begin{cases} 1, & \text{if } k \text{ is the entry in cell } [i,j] \\ 0, & \textit{otherwise}. \end{cases}$$

Next, Peggy and Victor agree on some constraints that need to hold if the solution is correct:[6]

$$\forall j,k : \sum_{i=1}^{9} x(i,j,k) - 1 = 0, \tag{1}$$

$$\forall i,k : \sum_{j=1}^{9} x(i,j,k) - 1 = 0, \tag{2}$$

$$\forall U,V \in \{0,3,6\} : \sum_{i=1}^{3}\sum_{j=1}^{3} x(i+U,j+V,k) - 1 = 0. \tag{3}$$

Equation (1) means that column j contains only one appearance of the digit k, equation (2) represents the fact that in row i there must be only one value of k, and equation (3) states that in each of the nine 3×3 boxes, the digit k appears only once.[7] Note that if constraints (1), (2), and (3) hold, the solution to the puzzle is correct, and by verifying them, Victor does not learn in which cell is which value. This is important for the zero-knowledge property.

Peggy now takes the playing cards with the digits 1 to 9. On the cells with the initial numbers, she puts the corresponding playing cards on top with the digits facing upward.[8] The other cards that correspond to her solution are put facedown onto the puzzle (Figure 3B).

Now the interactive part of the proof begins. Victor randomly chooses either a row, column, or box—here, he selects column $j = 6$. Peggy collects all nine cards of this column, shuffles them, and hands them to Victor. Victor can then look at the cards and check whether constraint (1) holds, i.e., whether each digit from 1 to 9 appears only once. This part represents one round of interaction in the protocol and is illustrated in Figure 3C. Victor makes a query—i.e., randomly chooses a row, column, or box—and Peggy's response is handing him the respective cards, which he can then verify.

Obviously, querying only once is not enough because Peggy might have set up the digits correctly in this column without having a proper overall solution and was just lucky that Victor chose this column by chance. Thus, Victor repeats the procedure by choosing additional rows, columns, and boxes. In each round, Peggy takes the corresponding cards, shuffles them, and hands them to Victor, who then verifies that either equation (1), (2), or (3) holds.

With each querying round, the probability that Peggy has indeed found a proper solution increases and the soundness error decreases. Victor can make as many queries as he wants until he is convinced that the probability of Peggy's statement being false is sufficiently small. He could theoretically also query for all 27 possible elements such that the probability of being cheated becomes 0 percent. However, this takes more time and makes the proof bigger and inefficient. Thus, he can define which soundness error is acceptable for him and then choose the corresponding number of queries. Hence, there is a trade-off between succinctness[9] (proof size) and soundness (security). Depending on the ZKP algorithm, this trade-off can also be prevalent in more complex proofs.

The proof described above fulfills the properties of zero knowledge, completeness, and soundness. It is zero knowledge because it is not possible to derive which number is in which cell because Peggy shuffles the cards, and it is complete because she can convince Victor that she found a correct solution. It is also sound because if she cheated by setting up the numbers in a wrong way, he will catch her with a sufficiently high probability.

This example brings us a bit closer to a more realistic ZKP. We introduced constraints and showed how the prover and verifier interact, how the verifier makes random queries, and how he checks whether the constraints are true. However, we are still far from a proper ZKP in which much more math is involved; to learn more, see Berentsen, Lenzi, and Nyffenegger, 2022.

6. In this example it is easy to come up with mathematical constraints that relate to the statement. In more involved proofs, more complex constraints first need to be transformed to a form that a ZKP can deal with. See, e.g., Buterin, 2016 for an intuitive explanation.

7. The terms U and V allow to "jump" to the next box.

8. Victor will challenge for several rows, columns, or boxes, and hence Peggy needs to put several cards on top of each other. For the sake of simplicity, we here assume that she does not cheat by putting distinct cards on top of each other.

9. Succinctness means that the proof can be efficiently verified.

3.2 Making Sudoku Non-Interactive

So far, we have seen two interactive (two balls and Sudoku) and one non-interactive (Waldo) proofs. We now want to show how it is possible to transform an interactive proof into a non-interactive one, using the Sudoku example. Showing this is a bit complicated, but by doing so, we learn new concepts that are used in real non-interactive proofs. Specifically, we introduce cryptographic hash functions, commitments, and the Fiat–Shamir heuristic, which are all used in ZKPs. To keep this section as simple as possible, we will abstract partly from the zero-knowledge property. Hence, the proof is closer to a validity proof, which is more about proofing the correctness of the statement efficiently rather than the zero-knowledge part.

First, we need to introduce cryptographic hash functions and modular arithmetic. A cryptographic hash function is a deterministic function that is practically infeasible to invert. It takes an input and creates an output of fixed length. Below we illustrate a few examples using the CRC-16 hash function,[10] which is denoted by H and returns an output in hexadecimal[11] form:

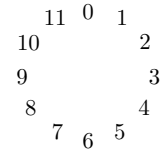
$$H(\text{zero-knowledge proof}) = ec65, \tag{4}$$

$$H(\text{blockchain}) = c964, \tag{5}$$

$$H(375869241) = fbd1, \tag{6}$$

$$H(348972651) = 4a33. \tag{7}$$

Modular arithmetic can be best illustrated by using a clock, and in this case the modulo operator is 12. Whenever an operation yields a result that is larger than or equal to 12, we follow the clock clockwise, starting with 0 again. The following are some examples:



$$11 + 2 \mod 12 = 1, \quad 29 \mod 12 = 5, \quad 6 \cdot 7 \mod 12 = 6$$

We saw in the interactive proof of the Sudoku puzzle that random queries by Victor are crucial. If they were not random and Peggy knew that the queries involved, for example, column $j = 6$, row $i = 9$, and box $b = 1$, she could have ordered the cards correctly only for these queries and cheated. To handle this in a non-interactive proof, we introduce commitments and use cryptographic hash functions to get pseudorandom queries. Furthermore, we stated above that in a non-interactive proof, Peggy provides only one proving object that everybody can use to verify the proof. In this example it is a string of characters denoted as $\mathcal{P}$, and we will show exactly how Peggy creates $\mathcal{P}$ and how a verifier can verify it. Figure 4 depicts the solution to the Sudoku puzzle.

The first part of the protocol requires Peggy to commit to the solution. She does so by applying a cryptographic hash function to all rows, columns, and boxes and including a compressed form of the hash outputs in the proof string. For example, she reads the numbers of the first row $i = 1$, i.e., 375869241, and applies the cryptographic hash function $H()$ to them. The result of this operation is $fbd1$ as shown in equation (6). The output of applying $H()$ to the numbers in column $j = 1$ is $4a33$ (see equation (7)).

Peggy then includes a compressed form of these hash outputs in the proof string whereby she commits to the solution. She cannot include the digits directly because she would reveal the solution. However, due to the deterministic nature of the cryptographic hash function, she can create a link from the solution to the hash output. Furthermore, by seeing only the hash output, another party cannot derive the solution because of the irreversibility property of the cryptographic hash function. The commitment ensures that Peggy cannot maliciously modify the solution in the subsequent steps of the proof; this will become clearer later.

Peggy could add all 27 hashes for all rows, columns, and boxes to the proof string $\mathcal{P}$. However, this would unnecessarily blow up the proof's size. Thus, she builds a Merkle tree as illustrated in Figure 5. In a Merkle tree, the elements on the bottom consist of the hashes of the rows, columns, and boxes. To go upward, the two elements just next to each other are hashed together in pairs until only one element is left, which is called the Merkle root. We again use the hash function CRC-16.[12] Peggy adds the root to the proof string, i.e., $\mathcal{P} = eb47$, and thereby commits to the solution. The root is the compressed form of the hash outputs mentioned in the paragraph above.

Now, remember that we need to introduce some pseudorandomness to replace the random queries of Victor in the interactive proof; i.e., we need to define how Peggy selects the first row, column, or box. The protocol solves this as follows:

10. https://emn178.github.io/online-tools/crc16.html

11. Hexadecimal is a numeral system that includes the symbols 0–9 and A–F. You can easily convert a hexadecimal number to a decimal or binary one.

12. https://emn178.github.io/online-tools/crc16.html

Figure 4
Sudoku Puzzle Solution

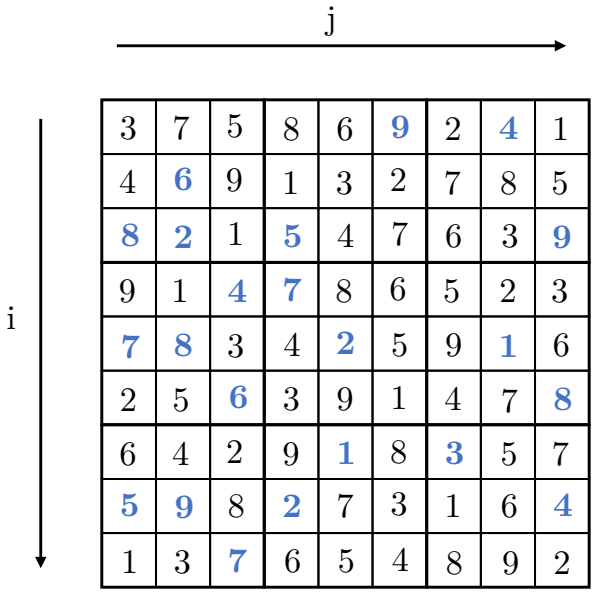
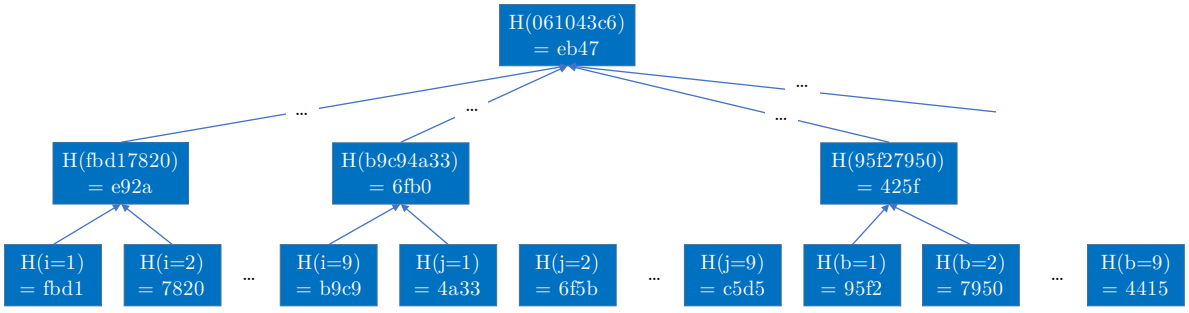


Figure 5
Merkle Tree



1. Peggy rewrites the Merkle root as a decimal number; i.e., *eb*47 = 60231.
2. She uses a mapping to transform the Merkle root in decimal form into $\mathbb{Z}_{27}$, where $\mathbb{Z}_{27} \in \{0, 1, \ldots, 26\}$. An easy way to do that is by applying modular arithmetic; i.e., 60231 mod 27 = 21.
3. If the resulting number is between 0 and 8, Peggy matches it to rows $i \in \{1, 2, \ldots, 9\}$. If it is between 9 and 17, she matches it to columns $j \in \{1, 2, \ldots, 9\}$, and if it is between 18 and 26, she matches it to boxes $b \in \{1, 2, \ldots, 9\}$. Since the result before was 21, she selects box $b = 4$ for the first querying round.

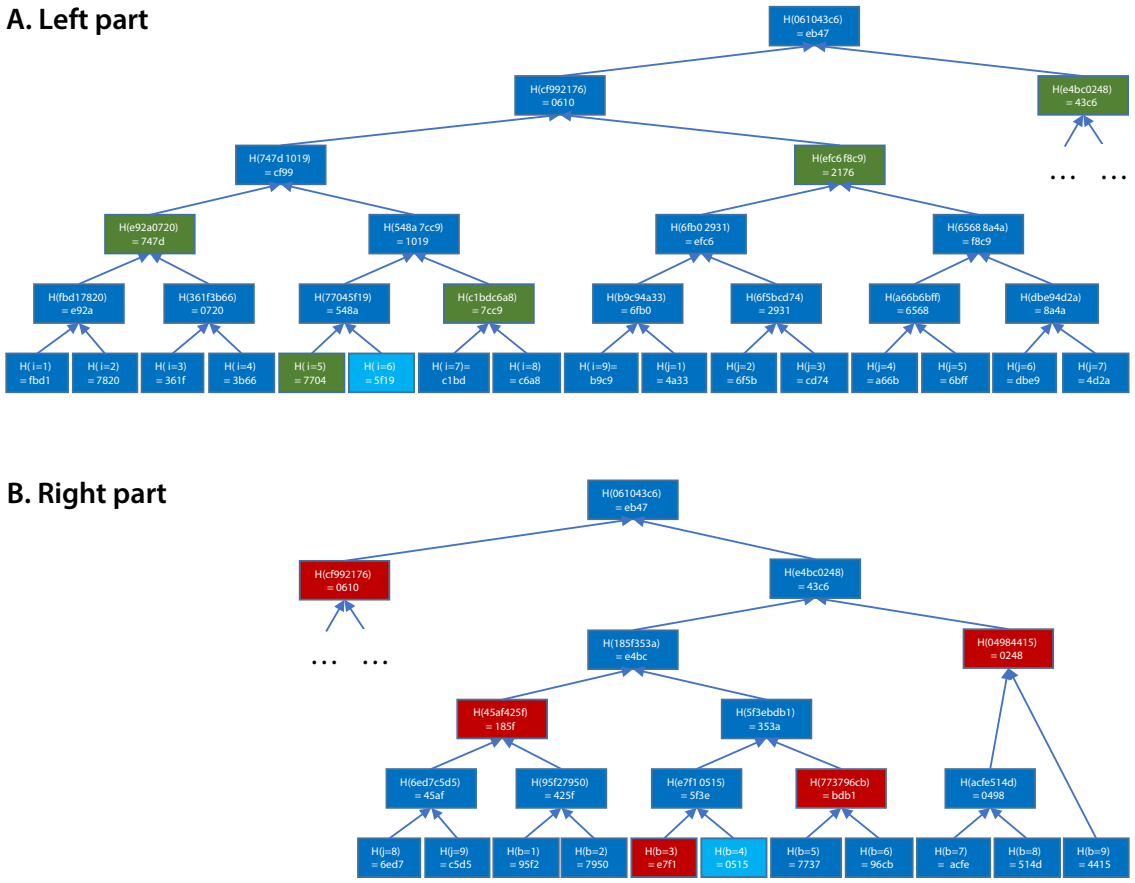
Replacing a random query by using a cryptographic hash function's pseudorandom output is called the Fiat-Shamir heuristic. This heuristic is also used in more complex proofs to make them non-interactive.

Peggy writes down the result of box $b = 4$, which is 914783256, and adds it to the proof string that already contains the Merkle root *eb*47; i.e., $\mathcal{P}$ = *eb*47 **914783256**. Now the commitment becomes important. Peggy could have written down any random combination of the digits 1 to 9 that does not correspond to the correct solution. However, she can prove correctness by using the Merkle tree as illustrated in Figure 6.

When building the Merkle tree, she uses the solution in box $b = 4$ to get a leaf at the bottom with hash 0515. If she chooses any different sequence of digits, a different hash would result in the bottom leave. Consequently, the hashes in the upper leaves would change as well, and a different Merkle root would result. But since she uses the Merkle root to select the box in the first place, she would end up with another row, column, or box choice in the first round, making it quite hard for her to cheat.

A verifier can check whether Peggy provides the correct sequence of digits that corresponds to the solution committed in the Merkle tree as follows. The verifier first hashes the substring in $\mathcal{P}$ to get the bottom leaf; i.e., $H(914783256) = 0515$, represented by the light blue box in Figure 6B. He then uses the result 0515 and

Figure 6
Sudoku



hashes it pairwise with all the hashes of the red boxes in Figure 6b to reconstruct this path of the tree. Finally, he checks whether the resulting Merkle root equals the one in $\mathcal{P}$. If they are equal, he is convinced that Peggy did not choose any wrong digit combination.

For the verifier to be able to perform these steps, he needs to know the hashes in the red boxes. Thus Peggy must provide them in the proof string in the proper order; i.e., $\mathcal{P}$ = *eb*47 914783256 **e7f1 bdb1 185f 0248 0610**. This concludes the first querying round. Peggy used the initial values of the Sudoku puzzle to pseudorandomly determine which column, row, or box she should choose. She added the sequence of digits in this row to the proof string and also attached certain hashes of the Merkle tree to convince a verifier that she did not just write down any random digits into the proof string.

For all subsequent rounds, we define the protocol as such that the pseudorandomness comes from the solution in the last round. Thus, the second querying round would work as follows. First, Peggy calculates the solution's hash in the last round; i.e., $H(914783256) = 0515$. She then rewrites it as a decimal number, i.e., $0515 = 1301$, and applies the mapping; i.e., $1301 \bmod 27 = 5$. She proceeds to translate 5 to row $i = 6$ for the next query and adds the solution of row $i = 6$, i.e., 256391478, to the proof string. Last, she attaches the hashes of the Merkle tree nodes needed to check whether the result matches the commitment to $\mathcal{P}$. They are illustrated in the green boxes in Figure 6A. The proof string now is

$$\mathcal{P} = eb47\ 914783256\ e7f1\ bdb1\ 185f\ 0248\ 0610\ 256391478\ 7704\ 7cc9\ 747d\ 2176\ 43c6.$$

Peggy could perform additional querying rounds, but we stop here to not blow up the proof size. She propagates her proof to the Sudoku community such that any verifier can now check the proof by only using the proof string and the initial Sudoku puzzle.

Next we describe how a verifier named Victor checks the proof. He is aware that Peggy queried twice, and since he knows the protocol, he can derive exactly which characters in the proof string have which meaning;

i.e., the first 4 characters represent the Merkle root, the subsequent 9 represent a solution for a row, column, or box, the next 20 characters are five hashes used in the Merkle tree, etc. Victor then verifies the proof as follows:

1. He reads the first four characters in the proof string that correspond to the Merkle root. He calculates which row, column, or box is queried first by transforming the Merkle root to a decimal number in $\mathbb{Z}_{27}$ as described above. He finds that the first query concerns box $b = 4$.
2. From the proof string, he reads the characters 5 to 13, which correspond to the solution in row $i = 5$ and then hashes it; i.e., $H(914783256) = 0515$.
3. He reads the subsequent 20 characters from the proof string and splits them up in five hashes with four characters each; i.e., $h_1 = \{e7f1, bdb1, 185f, 0248, 0610\}$. He then uses the solution's hash above—0515—and hashes it pairwise with the hashes in h_1. He can then check whether this matches the Merkle root, i.e., the first four characters in the proof string.
4. He uses the solution's hash in row $b = 4$ and transforms it to a decimal number in $\mathbb{Z}_{27}$. He will find that the next query is row $i = 6$.
5. From the proof string, he reads the characters 34 to 42, which correspond to the solution in column $i = 6$ and then hashes it; i.e., $H(256391478) = 5f19$.
6. He reads the last 20 characters from the proof string and splits them up in five hashes with four characters each; i.e., $h_2 = \{7704, 7cc9, 747d, 2176, 43c6\}$. He then uses the solution's hash above—$5f19$—and hashes it pairwise with the hashes in h_2. He can then check whether this matches the Merkle root, i.e., the first four characters in the proof string.

Any other verifier can replicate these steps to verify Peggy's proof. Obviously, there are many caveats in this still-easy example. First, the proof is not zero knowledge. With each additional round, Peggy reveals new information on the solution. To keep it zero knowledge, she would need to apply more complex mathematical methods, which would go beyond the scope of this article (see Berentsen, Lenzi, and Nyffenegger, 2022).

Second, the proof is not succinct as the proof string $\mathcal{P}$ has 62 characters. Since this example is very simple, propagating the digits in all fields of the puzzle would only contain either 81 or only 59 characters, depending on whether the initial values are included or not. However, for larger Sudoku puzzles, succinctness becomes relevant. A puzzle of size 25×25 with 119 initial values would require Peggy to send 506 characters if she sends the whole solution. Applying the protocol from above, she needs 4 characters for the Merkle root and in each round 25 characters for the solution and $7 \times 4 = 28$ for the hashes in the Merkle tree. This yields a proof size of $4 + 53n$, where n is the number of querying rounds. The proof size in this example is smaller than just sending the whole solution for $n < 10$.

It is possible to transform most interactive proofs into non-interactive ones. However, as seen above, the transformation makes the proof more complex. Interactive proofs are often simpler and more efficient than non-interactive ones (ZKProof, 2022). Since non-interactive proofs can be verified by any party at any time, they still have broad use cases as in, for example, blockchain applications.

3.3 Different Proof Protocols

When it comes to real-world applications, several different proof protocols can be used. Non-interactive proofs are the ones mostly used, and the generic term is often denoted as a SNARK, which was first introduced by Bitansky et al., 2012. SNARK stands for **S**uccinct **N**on-Interactive **AR**gument of **K**nowledge. "Succinct" means that verifying a proof is so much more efficient than reexecuting the computation and non-interactive that a prover can provide a single proving object that can be verified by everybody.

To understand the definition of "argument," we need to clarify the distinction between a proof and an argument. A proof needs to be sound (see Section 3, p. 283) even against a computationally unbounded malicious prover.[13] On the other hand, an argument relies on a weaker assumption and only requires soundness against a computationally bounded malicious prover (ZKProof, 2022). A computationally unbounded malicious prover could, for example, find collisions in cryptographic hash functions, which would be a serious security concern.[14] "Of knowledge" means that the prover must also prove that the secret input data are known (Szepieniec, 2022).

There are several different proof protocols in the line of a SNARK such as Groth16 (Groth, 2016), PlonK (Gabizon, Williamson, and Ciobotaru, 2019), Merlin (Chiesa et al., 2019), Bulletproofs (Bünz et al., 2018), or Nova (Kothapalli, Setty, and Tzialla, 2021). Many SNARKs rely on elliptic curves[15] and thus on the assumption

13. The term "computationally unbounded" refers to a theoretical concept in which an entity has such a huge amount of computing power that any computation can be executed rapidly. In practice there are hardware limitations that make certain theoretical attacks unfeasible.

14. Finding collisions of the mostly implemented cryptographic hash functions requires an enormous amount of computational power. In practice, a computationally bounded attacker cannot successfully perform this attack with a very high probability.

15. Also, the private-public key encryption used to access crypto assets in a wallet normally relies on elliptic curves.

that discrete logarithms are difficult to compute (Thaler, 2022). However, this assumption is not postquantum secure.[16]

Furthermore, using elliptic curves in a ZKP requires that certain parameters are set up in advance. These parameters are then used when creating a proof. The problem is that if somebody knows how these parameters were created, he could cheat and create false proofs. As a result, these parameters are normally created in initial set-ups called ceremonies, in which several parties participate. All parties contribute to the set-up by secretly creating random elements that are then combined to get the parameters. If it is somehow possible to get hold of all random elements, it would be possible to create false proofs. Thus, these elements are often called toxic waste. However, if at least one party in the ceremony destroys the personal random element, it is not feasible to derive the composition of the parameters and the proofs are secure.

Due to these shortcomings, there has been an effort to create ZKPs that do not rely on elliptic curves. The most prominent protocol that works without elliptic curves are STARK proofs, introduced by Ben-Sasson et al., 2018. The main cryptographic technique used are cryptographic hash functions. Hence, STARK proofs are postquantum secure and do not need an initial set-up or ceremony, and therefore a S**T**ARK is called **T**ransparent. Furthermore, the **S** in **S**TARK does not stand for succinct but for **S**calable. In addition to having efficient verification, a STARK entails some efficiency assumptions on the creation of the proof by the prover (Szepieniec, 2022).

An advantage of a SNARK over a STARK proof is that the proof size is considerably smaller. This becomes especially relevant when it comes to blockchains in which block space is costly and thus users are interested in shorter proofs. Furthermore, in STARKs there is an inherent trade-off between verification costs and security that is not prevalent in SNARKs, which rely on elliptic curves (Thaler, 2022).

4. APPLICATIONS IN BLOCKCHAIN

When it comes to blockchains, there are two main applications of ZKPs: data confidentiality and efficiency. An example for the former use case is privacy-protecting payments, whereas for the latter, it is increased scalability by implementing rollups.

4.1 Data Confidentiality

An example where data confidentiality is applied are privacy-protecting transfers. In standard blockchains, each transaction is registered on the blockchain and is hence visible to everybody. Even though a user remains pseudonymous, it can be possible to derive a real identity using transaction data from and to an address. By using ZKPs, it is possible to prove that someone is allowed to spend a certain amount without revealing the sender and receiver address and amount to the whole network.

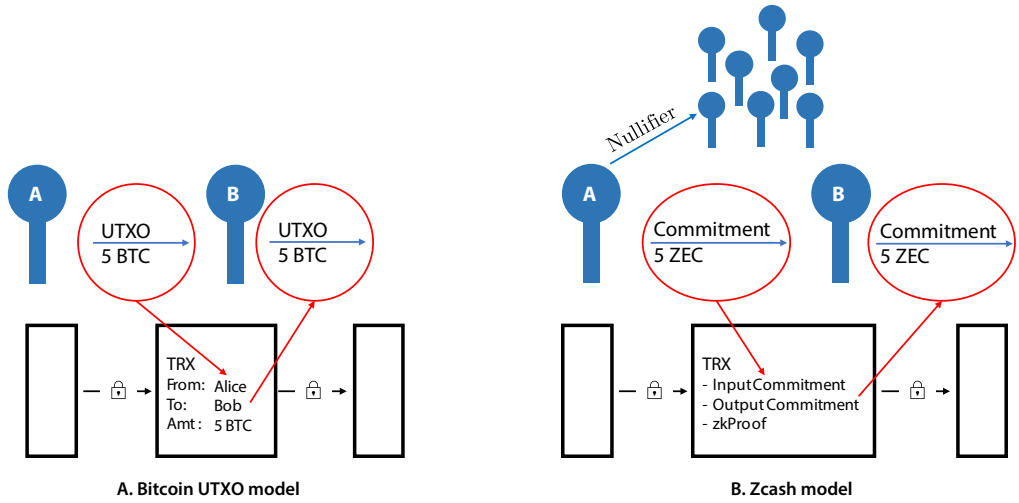
Zcash, introduced by Ben Sasson et al., 2014, is probably the most famous cryptocurrency that leverages ZKPs for private transactions. To understand how Zcash works, we first recap Bitcoin's (BTC) transaction model. The Bitcoin network uses unspent transaction outputs (UTXOs) to determine which address is allowed to spend a specific amount of the currency, illustrated in Figure 7A. Consider an example where Alice receives 5 Bitcoin (BTC) in a transaction from another network participant. The network stores the information that a UTXO worth 5 BTC exists. If Alice wants to send 5 BTC to her friend Bob, she creates a transaction in which she references the UTXO. She also needs to sign the transaction with her private key to prove that she can rightfully spend the UTXO. Next, Alice propagates the transaction to the network, which then includes it in a block. By sending the 5 BTC to Bob, a new UTXO is created that he can use again.

In Zcash a similar construct is used called commitment. Say that Alice has received funds via the Zcash blockchain, which comes in the form of a commitment that, contrary to a UTXO, does not reveal the address and the available amount to spend to the network. Conceptually, it is a hash of her address, the spendable amount, and a serial number denoted as s; i.e., $Commitment = H(address, amount, s)$. If Alice creates a transaction to send the funds to Bob, she does the following (see Figure 7B):

1. She prepares a ZKP to prove that she is allowed to spend the input commitment that corresponds to the commitment's hash.
2. She creates a new output commitment, which is a hash of Bob's address, the amount, and the serial number s.
3. She propagates the commitment together with her ZKP to the network, which then includes it in a block.

16. A quantum computer can very efficiently factor integers using Shor's algorithm, which can be used to break elliptic curve cryptography. However, current quantum computers are far from having enough computational power to break the cryptography in a reasonable time see, e.g., Webber et al., 2022.

Figure 7
Difference between the UTXO model in Bitcoin and the Zcash Model



4. Simultaneously, she creates a nullifier that is derived from the serial number *s*. She propagates it to the network, which stores all nullifiers. The nullifier set is a list of hashes that show which input commitments have been spent. If she wanted to double spend the same input commitment again, the same nullifier would result and the transaction would be considered invalid by the network.
5. She privately sends Bob the information that the output commitment relates to his address, the amount included, and the serial number *s*. Bob needs this information to be able to spend the funds himself.

The ZKP proves she is in possession of the private key corresponding to the address and that the amount of the input commitment equals the amount of the output commitment without revealing her address nor the amount.

4.2 Efficiency

A trade-off that many blockchain networks face is the trilemma of security, decentralization, and scalability. Many protocols that emphasize their role of a decentralized network, such as, for example, Bitcoin and Ethereum, face at some point the problem that the demand for block space is much higher than the supply. In other words, the demand for sending transactions exceeds what the network can handle, which in turn makes sending a transaction in these networks pretty expensive.

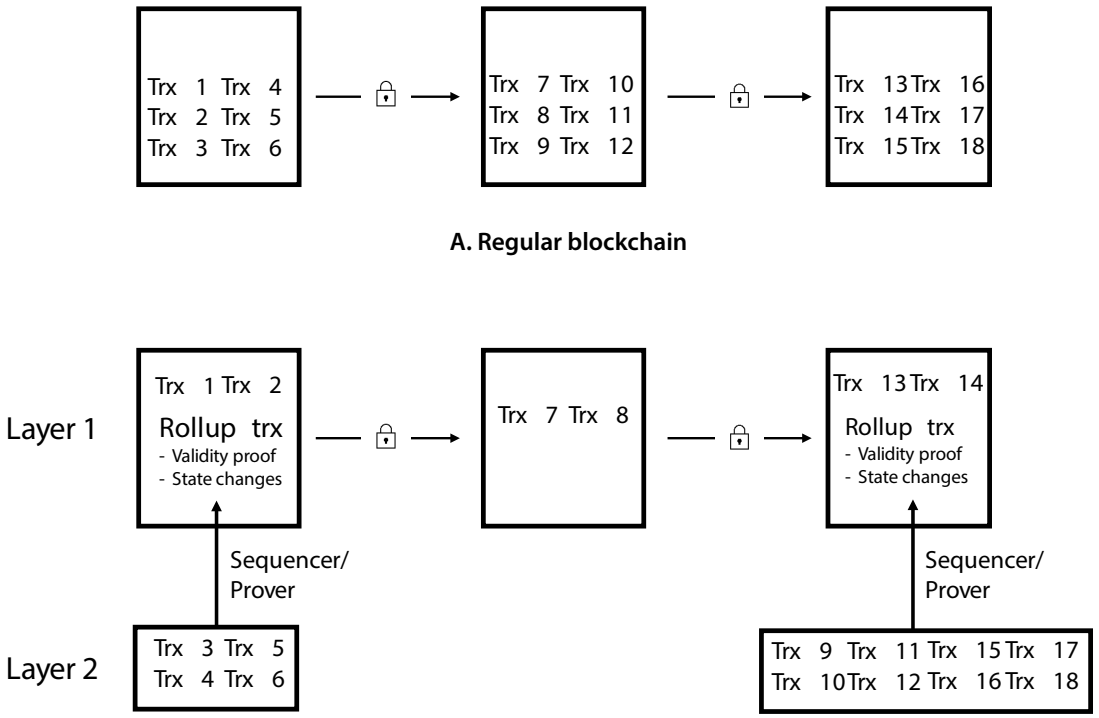
It would be relatively easy to increase scalability by raising the number of transactions that can be included in a block. However, a highly decentralized network relies on the assumption that any regular person who is interested in participating in the network and verifying that the state of the network is correct can do so with a reasonable hardware (e.g., with a regular desktop computer). But if more transactions are sent over the network, the network needs to store more data and the verification of all the transactions becomes more expensive, which in turn requires better hardware and excludes some participants from verifying the state of the network. Thus, blockchain developers are interested in finding solutions that increase scalability without harming decentralization or security.

The use of ZKPs promises to remedy exactly that through the use of so called rollups, which are a layer 2 scaling solution.[17] In a regular blockchain all transactions are included within blocks of the base chain as illustrated in Figure 8A. As discussed earlier, verifying all transactions can be very expensive—a big computational burden is to check all the signatures to ensure that someone is allowed to execute a certain transaction. The idea of a rollup is that a network participant called a sequencer[18] batches (or rolls up) transactions off-chain (on layer 2) and creates a proof that proves the validity of these transactions on layer 1. This process is depicted in Figure 8B.

17. Layer 1 would be the base chain (e.g., Ethereum). A layer one blockchain can be scaled directly by, for example, increasing the block size. Rollups scale a layer 1 blockchain indirectly and hence the scaling is referred to as a layer 2 scaling solution.

18. In theory, the sequencer could be decentralized or a centralized party, but in practice, sequencers are currently centralized in real implementations. However, there is work ongoing to decentralize the sequencer as, for example, in the case of the StarkNet rollup (see, e.g., StarkWare, 2022).

Figure 8
Rollups Illustrated Conceptually



Consequently, network participants do not have to verify each individual transaction and signature but only the compressed proof, which is much less expensive. The proofs are based on ZKPs but currently do not incorporate privacy and instead focus on proving the validity of the transactions in a succinct way. Thus in this context, they are often called validity proofs and not ZKPs.[19] Basically, what happens is that on layer 1 (on the main chain), a smart contract is created that acts as the proof's on-chain verifier. The sequencer is the prover who creates proofs off-chain on layer 2 and sends a transaction to the on-chain smart contract, which verifies the proof.

So far, we only described the validity of the transactions. However, we also need to record in the blockchain who sent a transfer to whom. This recording is done by publishing the state changes[20] in the call data[21] of the transaction sent to the verifier smart contract. In that way, the current account balances are always updated on-chain. Hence, executing (i.e., checking the validity of the transactions and signatures) is done off-chain, but settling the transactions (i.e., state changes) is on-chain.[22] By using a rollup, much less block space is used on the

19. Note that there is another widely used type of rollup, i.e., optimistic rollups. Optimistic rollups do not rely on ZKPs, which is why we do not discuss them here. They are optimistic in the sense that any network participant can propose a state change on-chain. Afterwards, some network participants check the state change, and if a transaction in it is invalid, a fraud proof can be posted on-chain, which reverts the state back. Thus, optimistic proofs have a period of about one week until a transaction is final, in which network participants have time to check the state change. The difference between ZKP rollups and optimistic rollups may become more intuitive in the following joke: Two rollups walk into a bar. The barman asks, "Can I see your ID's?" The ZKP rollup says, "I can prove to you I'm over 18, but I won't show you my ID." The optimistic rollup says, "If nobody can prove I'm underage in 7 days, that means I'm over 18."

20. Ethereum works with an account-based model. At each period in time, each account is assigned a specific balance, which can be interpreted as a huge spreadsheet and is called the state. The state is hashed in a tree, and the root is written into each newly produced block. If a transaction is sent from Alice to Bob, the balance of Alice decreases, whereas the one of Bob increases; this process is called a state change. After the transaction, there is a new state, which is again published to the blockchain.

21. The call data are an additional field in a transaction in which some arbitrary data can be written. Note that writing data into the call data costs gas (i.e., the native way to pay for the inclusion of a transaction in the Ethereum network). However, there is a discussion in the Ethereum network to reduce the call data gas cost to make it more rollup-friendly (Ethereum Improvement Proposal 4488).

22. By now, there are many different types of rollups depending on where the data are stored, who performs the execution, etc. See,

base chain. Additionally, since the computational effort used to verify the proof scales logarithmically with the size of the input data (i.e., the number of transactions), the fee per transaction when using a rollup decreases in the number of transactions in the rollup and is generally much lower compared to sending a transaction directly on layer 1.

Note that it is still possible to make regular transactions on layer 1, as illustrated in Figure 8B, where some transactions are batched on layer 2 but some are still on the main chain. Moreover, a rollup does not need to send a transaction batch each block. The sequencer could wait until enough transactions are accumulated and only publish a batch every few blocks and in doing so reduce the fee per transaction. This process, however, has the disadvantage of a delayed finality time, and if the sequencer is not decentralized, it comes with some counterparty risk.

On layer 2, developers can build any type of smart contract or application. It is even possible to think of a layer 3 or layer 4 to further increase scalability. In each layer transactions would be batched and sent to the lower layer. However, the more layers that are introduced, the longer the finality time for the upper layers.

There are also other blockchain applications that use ZKPs for efficiency reasons. For example, Filecoin uses them for a decentralized storage application. Instead of storing data with a large company providing a cloud service, Filecoin enables storing data in a decentralized way. To do that in a confidential way, the data to be stored are split up into small chunks and are stored by several nodes across the network.[23] The storage providers earn a reward by storing the data, and they use a ZKP to efficiently prove that they do store them correctly.

5. APPLICATION IN OTHER AREAS

ZKPs potentially have many other use cases apart from blockchains. Even though the theoretical concepts have been around for some time, ZKPs have not been widely adopted yet. There are many possible applications with regard to data confidentiality—this includes privacy and identity management—or efficiency. A basic requirement to use ZKPs in real-world problems is that for each application there must be some underlying cryptographic set-up. When it comes to blockchains, this is naturally existent. However, in real-world examples we would often need to rely on some trusted third parties, as we show below.

5.1 Data Confidentiality

Being private about personal characteristics is one example where ZKPs could be used for data confidentiality. As an example, Carlos wants to prove at the entrance of a nightclub that he is 18 without revealing his exact age or any other personal characteristic. Judy wants to enter a fancy club she has a membership to but does not want the doorman to know who she is. Both could create a ZKP to prove that they are allowed to enter. For both examples, we would need some kind of underlying cryptographic set-up. In the former case, the government might have provided a digital identity, whereas in the latter case the club owners could have issued a digital membership card that they signed. The doorman could verify that the proof includes the signature of the club owner and would let Judy in.

ZKPs could also be used in areas in economics that are not directly linked to blockchains. Assume the company funInvest wants to prove that it is solvent without revealing its exact portfolio composition. It is not straightforward to use a ZKP in this environment since an underlying cryptographic set-up with respect to their asset holdings is missing. If, for example, funInvest owns some shares of company X, a trusted third party, the company X itself could instead sign a message that funInvest owns shares above a certain amount without revealing the exact amount. This becomes easier when funInvest holds some crypto asset. It could then prove that it holds the private key to an address that has an amount of assets above a certain threshold, without revealing the exact amount or the address.[24]

A proposal from ING, 2017 goes into a similar direction. For example, ZKPs could be used for a mortgage applicant to prove that her salary is in a specific range needed to receive the mortgage without revealing her exact salary.

5.2 Efficiency

An example where ZKPs increase efficiency is when you can create a short ZKP instead of filing a ton of paper work. ZKProof, 2022 discusses one example about airplanes. Airplanes undergo a lot of checks and maintenance, which a regulatory body checks. Anyone who works on the plane files some papers, which

e.g., Charbonneau, 2022 for an overview.

23. This storing is done with some redundancy such that even if some storage providers fail, the data are still available.

24. Even though the verifier learns that funInvest holds funds above the threshold, the zero knowledge property still holds. Zero knowledge means that the verifier cannot learn more than the statement to be proven and the statement is explicitly about the threshold.

the airline then collects and forwards to the regulatory body. Checking all the paperwork is quite inefficient. Instead, the regulatory body could define which workers are trusted to make the checks. After conducting the checks, the workers can cryptographically sign that they did the work instead of filing papers and provide a digital receipt to the airline. The airline could then combine all the digital receipts and provide them to the regulatory body, which proves that the workers did indeed do the checks and maintenance work. The proof can then be efficiently verified by the regulatory body.

6. CONCLUSION

In this article we discussed ZKPs and their applications. ZKPs can make certain digital processes considerably more efficient and can be used in privacy applications. There are a lot of use cases when it comes to blockchains but also in broader areas of economics. For blockchains, ZKPs can be used to improve scalability or to protect privacy, such as when using the Zcash cryptocurrency.

With this review article, we aim to shed light on the topic and provide a basic understanding to a broad audience. For the more ambitious reader, Berentsen, Lenzi, and Nyffenegger, 2022 go through the math of an easy ZKP and provide an accompanying Python script.

REFERENCES

Ben Sasson, Eli, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: decentralized anonymous payments from bitcoin. *2014 IEEE Symposium on Security and Privacy,* 459–474. https://doi.org/10.1109/SP.2014.36.

Ben-Sasson, Eli, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *International Association for Cryptologic Research,* ISSN: 1746-8094. https://eprint.iacr.org/2018/046.pdf.

Berentsen, Aleksander, Jeremias Lenzi, and Remo Nyffenegger. 2022. *A walk-through of a simple zk-stark proof.* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4308637.

Bitansky, Nir, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12.,* 326–349.

Bünz, Benedikt, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: short proofs for confidential transactions and more. *2018 IEEE Symposium on Security and Privacy (SP).*

Burger, Elena, Bryan Chiang, Sonal Chokshi, Eddy Lazzarin, Justin Thaler, and Yahya Ali. 2022. Zero knowledge canon, part 1 & 2, September. Accessed September 21, 2022. https://a16zcrypto.com/zero-knowledge-canon/.

Buterin, Vitalik. 2016. Quadratic arithmetic programs: from zero to hero. Accessed April 19, 2022. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

Chalkias, Konstantinos, and Mike Hearn. 2019. Demonstrate how zero-knowledge proofs work without using maths. *CORDACON 2017 Conference* (February).

Charbonneau, Jon. 2022. The complete guide to rollups. Accessed September 22, 2022. https://members.delphidigital.io/reports/the-complete-guide-to-rollups.

Chiesa, Alessandro, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. 2019. Marlin: preprocessing zksnarks with universal and updatable srs. *Cryptology ePrint Archive, Paper 2019/1047.*

Gabizon, Ariel, Zachary J. Williamson, and Oana Ciobotaru. 2019. Plonk: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive, Paper 2019/953.*

Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing* 18 (1): 186–208.

Gradwohl, Ronen, Moni Naor, Benny Pinkas, and Guy N. Rothblum. 2009. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. *Theory of Computing Systems* 44 (2): 245–268. ISSN: 14324350. https://doi.org/10.1007/s00224-008-9119-9.

Groth, Jens. 2016. On the size of pairing-based non-interactive arguments. *Advances in Cryptology – EUROCRYPT 2016,* https://doi.org/10.1007/978-3-662-49896-5.

ING. 2017. Ing launches zero-knowledge range proof solution, a major addition to blockchain technology. Accessed April 16, 2022. https://www.ingwb.com/en/insights/distributed-ledger-technology/ing-launches-major-addition-to-blockchain-technology.

Kothapalli, Abhiram, Srinath Setty, and Ioanna Tzialla. 2021. Nova: recursive zero-knowledge arguments from folding schemes. *Cryptology ePrint Archive, Paper 2021/370.*

Nadler, Matthias, and Fabian Schaer. 2023. Tornado cash and blockchain privacy: a primer for economists and policy makers. *Federal Reserve Bank of St. Louis Review* 105 (2).

Naor, Moni, Yeal Naor, and Omer Reingold. 1999. Applied kid cryptography or how to convince your children you are not cheating. *Journal of Cryptology.*

Schaer, Fabian, and Aleksander Berentsen. 2020. *Bitcoin, blockchain, and cryptoassets: a comprehensive introduction.* MIT Press, September.

StarkWare. 2022. Starknet: on to the next challenge. Accessed May 13, 2022. https://medium.com/starkware/starknet-on-to-the-next-challenge-96a39de7717.

Szepieniec, Alan. 2022. Anatomy of a stark, part 1: stark overview. Accessed September 22, 2022. https://aszepieniec.github.io/stark-anatomy/overview.

Thaler, Justin. 2022. Snark security and performance. Accessed September 22, 2022. https://a16zcrypto.com/snark-security-and-performance/.

Webber, Mark, Vincent Elfving, Sebastian Weidt, and Winfried K. Hensinger. 2022. The impact of hardware specifications on reaching quantum advantage in the fault tolerant regime. *AVS Quantum Science* 4 (1): 013801. ISSN: 26390213. https://doi.org/10.1116/5.0073075.

ZKProof. 2022. Zkproof community reference (version 0.3), https://docs.zkproof.org/pages/reference/versions/ZkpComRef-0-3.pdf.