Master's thesis

Computer Science

Zero-knowledge proofs in blockchain applications

Antti Tani

November 26, 2020

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. Valtteri Niemi

Examiner(s)

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Computer Science

Tekijä — Författare — Author

Antti Tani

Työn nimi — Arbetets titel — Title

Zero-knowledge proofs in blockchain applications

Ohjaajat — Handledare — Supervisors

Prof. Valtteri Niemi

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	November 26, 2020	78 pages

Tiivistelmä — Referat — Abstract

The release of Bitcoin marked the birth of blockchain applications. Due, among other things, to the need for public verifiability, blockchain information is often transparent, which in many cases leads to insufficient privacy. Various methods have been developed to obfuscate the blockchain data, which should at the same time maintain public verifiability. A promising cryptographic approach is zero-knowledge proof that enables a statement to be proved without revealing any other information than the validity of the statement.

Zero-knowledge proofs are examined in detail, first focusing on their general properties. With blockchains, the key features for zero-knowledge proof schemes are non-interactivity and succinctness, and schemes that fulfill these requirements are often called as zk-SNARKs. In a limited use, where succinctness is not critical, Fiat-Shamir transform has also been useful. We study the use of zero-knowledge proofs in blockchain applications Zcash, Ethereum and Monero, with a particular focus on privacy and feasibility.

ACM Computing Classification System (CCS)
Theory of computation → Computational complexity and cryptography → Cryptographic protocols
Security and privacy → Security services → Pseudonymity, anonymity and untraceability

Avainsanat — Nyckelord — Keywords

zero-knowledge proof, blockchain, privacy, zk-SNARK

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms specialisation line

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Computer Science

Tekijä — Författare — Author

Antti Tani

Työn nimi — Arbetets titel — Title

Zero-knowledge proofs in blockchain applications

Ohjaajat — Handledare — Supervisors

Prof. Valtteri Niemi

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	26.11.2020	78 pages

Tiivistelmä — Referat — Abstract

Bitcoinin julkaisu merkitsi samalla lohkoketjusovellusten syntymää. Johtuen muun muassa julkisen todennettavuuden tarpeesta, lohkoketjussa säilytettävä tieto on tyypillisesti läpinäkyvää, joka voi olla ongelmallista yksityisyyden kannalta. Yksityisyyden parantamiseksi on kehitetty menetelmiä, jotka hämärtävät lohkoketjun tietojen läpinäkyvyyttä säilyttäen niiden eheyden ja todennettavuuden. Lupaava kryptografinen menetelmä tähän tarkoitukseen on nollatietotodistus, joka mahdollistaa väitteen todistamisen siten, että ainoa todistuksessa paljastuva tieto on väitteen totuusarvo.

Nollatietotodistuksiin perehtyminen aloitetaan niiden teoreettisesta perustasta. Lohkoketjujen kannalta tärkeitä vaatimuksia nollatietotodistuksille ovat ei-interaktiivisuus ja ytimekkyys, ja nämä ehdot täyttäviä todistusrakenteita kutsutaan yleisesti nimellä zk-SNARK. Fiat-Shamir muunnos on käyttökelpoinen menetelmä ei-interaktiivisen nollatietotodistuksen muodostamiseen tapauksissa, joissa ytimekkyys ei ole tärkeää. Nollatietotodistusten käyttöä tutkitaan erityisesti yksityisyyden ja käyttökelpoisuuden kannalta kolmessa lohkoketjusovelluksessa, jotka ovat Zcash, Ethereum ja Monero.

ACM Computing Classification System (CCS)
Theory of computation → Computational complexity and cryptography → Cryptographic protocols
Security and privacy → Security services → Pseudonymity, anonymity and untraceability

Avainsanat — Nyckelord — Keywords

zero-knowledge proof, blockchain, privacy, zk-SNARK

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms specialisation line

Contents

1 Introduction

In 2008, pseudonym Satoshi Nakamoto published a white paper, describing a decentralized payment system called Bitcoin, first of its kind. Shortly thereafter, he released its implementation as open source. Gradually, Bitcoin's peer-to-peer network began to grow. People downloaded Bitcoin software and started running it on their own computers, thus forming new nodes to the p2p network. The nodes validate the data of payment operations, i.e., transactions, and also participate in a process called mining that is essential for the integrity of the protocol.

Bitcoin's success as a decentralized payment system was soon evident. It was also soon realized that its essential structure, known as the blockchain, could be used as a basis for other decentralized systems. We call these blockchain-based decentralized systems as blockchain applications, or if they are strictly payment systems, as cryptocurrencies. Blockchain provides an immutable public ledger and brings an elegant solution to a problem of how to reach consensus in distributed systems.

The early blockchain applications, including Bitcoin, were totally transparent, which means that all the data of atomic operations, i.e., transactions that are stored in the blockchain are publicly visible. The senders and receivers of transactions are tied to the actual users, but this information is obscured by addresses that act as pseudonyms. Ultimately, this obfuscation is not enough to provide sufficient privacy. Addresses may be interlinked or be even linked to actual users by transaction graph analysis that is possibly enriched with side-channel information.

Mixing services have been introduced as a way to increase obfuscation in transparent cryptocurrencies. In a mixing operation the sender and receiver addresses of multiple transactions are mixed with each other. This does not, however, completely remove the linkability threat. Methods to achieve more profound privacy must be sought in cryptography.

A particularly promising cryptographic method is zero-knowledge proof, which enables the statement to be proved without revealing to the verifying party anything other than the validity of the statement. In the blockchain, this means that the transaction could still be validated when the confidential data of the transaction is not disclosed but is replaced with zero knowledge proofs of their validity.

Both the blockchain technology and zero-knowledge proofs are based on cryptography, hence, in Section 2 the required cryptographic concepts are introduced. Section 3 reviews blockchains through the properties of Bitcoin. Also the privacy issues are brought out. Zero-knowledge proofs are handled in Section 4. Treatment starts with interactive zero-knowledge proofs that were historically introduced first, and which form a basis to understand non-interactive zero-knowledge (NIZK) proofs that are reviewed thereafter. Non-interactivity is essential for blockchain applications, where proofs need to be publicly verifiable. NIZK proofs were initially too inefficient for any practical use, with the exception of Fiat-Shamir transform that is also handled in the NIZK proof section.

In the context of blockchains, the most important NIZK proof efficiency metrics are the proof size and validation time of the proof. Proofs are stored in the blockchain, and in the examples we study, to fully participate in blockchain application's network requires storing a local copy of the blockchain. Proofs are validated by every participant, so the validation should also be efficient.

Before 2013 the only NIZK proof scheme that could be used with blockchains was Fiat-Shamir, and that too only as a sub-protocol in transaction validation, because it does not yield short enough proofs to validate the transaction as a whole. In 2013 was introduced the first practical NIZK proof scheme to validate the entire transaction. This scheme, that represents a category called zk-SNARK, is reviewed in Section 4.6.

Examples of blockchain applications that use zero-knowledge proofs are reviewed in Section 5. Making substantial changes to existing blockchain application may be difficult. This is the case with Bitcoin, where the decision making of the protocol changes is also decentralized. Two of the studied examples, Zcash and Monero are cryptocurrencies that have been privacy-focused from the beginning. Third example, smart contract platform Ethereum that also represents a more diverse use of blockchains, has introduced zero-knowledge proofs at a later stage. Privacy properties, that are in these examples achieved using zero-knowledge proofs are examined.

The main contributions of this thesis are to provide a thorough overview of the zero-knowledge proofs in general, and then to investigate their use with blockchain applications; how privacy is achieved with them and how suitable they are to that purpose.

2 Cryptography

Traditionally *cryptography* has mostly dealt with *encryption*. For millenia, the need for a secure communication through insecure channels has dominated its development. Currently cryptography is a multidisciplinary field, and can be seen to deal with the system's ability to handle malicious attempts of abuse [61]. Another trend related to the birth of modern cryptography has been the adoption of rigorous, scientific methodology. Also a paradigm shift from an absolute security requirement to a probabilistic approach, which takes into account adversary's limited computational resources, has been fruitful for theoretical and practical advances of cryptography.

2.1 Basic concepts

In this section we define some basic concepts of computer science, that are also relevant for modern cryptography.

Turing machine A *Turing machine* (TM) is a mathematical model of computation. Turing machine has an internal state, work tape, and finite transition function that defines based on the current state and symbol in the work tape (1) what symbol the machines writes to the work tape, (2) the direction machine moves on the work tape and (3) the next state of the machine. TM also has an initial state where its computation begins, and a set of accepting states where the TM halts if it transfers to such a state. According to Church-Turing thesis, for every algorithm a corresponding Turing machine that simulates it can be constructed.

Nondeterministic Turing machine A *nondeterministic Turing machine* (NTM) is a variant of Turing machine. In contrast to deterministic Turing machine, NTM may have multiple available transitions in any situation, and it has the ability to guess the right transition in the path that eventually leads to an accepting state, if such a transition exists.

Efficient An algorithm is *efficient* if it is polynomial time, i.e., its execution time can be bounded by a polynomial on the size of the input. The execution time, also called as *time complexity*, is the number of steps (elementary operations) the algorithm takes.

Infeasible A problem is *infeasible* if there is no efficient algorithm to solve it [62].

Decision problem A *decision problem* is a problem that on any input, has only two possible outputs: 1 ("yes") or 0 ("no"). For an input set X, decision problem is a function $\omega : X \to \{0, 1\}$.

Language A decision problem ω defines interchangeably a *language* $L_\omega := \{ x \mid \omega(x) = 1 \}$

P Complexity class P is the set of decision problems that can be solved in polynomial time (efficiently) by a deterministic Turing machine. A corresponding language-based definition is: Complexity class P is the set of languages that can be recognized in polynomial time by a deterministic Turing machine.

NP Complexity class NP is the set of decision problems that can be solved in polynomial time by a nondeterministic Turing machine. An equivalent formulation is the following: A language L is in NP if for every true statement $s_x := x \in L$ there exists a proof that is verifiable in polynomial time by a deterministic Turing machine. A widely supported, but not yet proved conjecture is, that $P \neq NP$.

NP-complete A language L is *reducible* to language C in polynomial time if there is a polynomial time function f such that $x \in L \Leftrightarrow f(x) \in C$. A language C is *NP-complete* if $C \in NP$ and every language $L \in NP$ is recucible to C in polynomial time. This means that NP-complete problems are the hardest problems in NP. The existence of an efficient solving algorithm for any NP-complete problem would imply that $P = NP$ (but this is not deemed expected).

Witness Complexity class NP can also be defined in the following way [61]: A language L is in NP if and only if there exists a relation $R_L \subseteq \{0, 1\}^* \times \{0, 1\}^*$ that can be recognized in polynomial time and a polynomial $p(\cdot)$ such that

$$x \in L \Leftrightarrow \exists w : |w| \leq p(|x|) \wedge (x, w) \in R_L.$$

Such a w is called a *witness* for membership of $x \in L$. In other words, it can be said, that NP consists of languages for which there exists a short (polynomial) proof on membership that can be efficiently (in polynomial time) verified.

BPP A *probabilistic Turing machine* may have multiple available transitions in any situation, and this transition is drawn from some probability distribution. So unlike

the "pure" NTM described before, this probabilistic variant is not a "lucky guesser", but a "random guesser". A complexity class *bounded-error probabilistic polynomial time* (BPP) is the set of languages, that can be recognized in polynomial time by a probabilistic Turing machine that has an error probability* smaller or equal than $\frac{1}{3}$. It can be seen as one of the largest classes of problems, that can be solved efficiently.

Negligible A function $\mu : \mathbb{N} \to \mathbb{R}$ is *negligible* [61] if for every positive polynomial $p(\cdot)$ there exists an integer N_p such that for all $x > N_p$,

$$\mu(x) < \frac{1}{p(x)}$$

Therefore negligible function is asymptotically smaller than any inverse polynomial.

Overwhelming A function μ is *overwhelming* if $1 - \mu$ is negligible.

Security parameter A cryptographic scheme is based on some computational problem. A *security parameter* is a input size related measure, that is used to calculate the computational complexity of that problem. Therefore the security of the cryptographic scheme is dependent on the security parameter. For example in the RSA public key cryptosystem, the security parameter determines the size of generated keys, that again dictates the security of the scheme. In short, it can be said that the adversary's probability to break a cryptographic scheme should be a negligible function on the security parameter [7].

Oracle An *oracle* is a black box that replaces a function that is a part of a computational scheme, helping to construct a mathematical proof for that scheme. Oracle is typically an idealization of the function it replaces.

Boolean circuit

A *boolean circuit* is a model of computation. It corresponds to digital logic circuits, which are fundamental building blocks in computing devices. Boolean circuit is a directed, acyclic graph, which consists of boolean *input variables* and *logic gates* that are connected to each other by *wires*. A logic gate has depending on the type, one or two inputs and one output. The input of a logic gate can be a circuit's input variable or the output of

*The error probability that the machine answers wrong, whether $x \in L$ is bounded away from $\frac{1}{3}$. The error probability drops exponentially when algorithm is run multiple times and a "majority vote" is taken from the results.

other gate. The *circuit output* consists of gate outputs that are not connected to any gate inputs. Boolean circuit with n inputs and m outputs can be used to calculate the value of a *boolean function* $f : \{0,1\}^n \to \{0,1\}^m$

There are seven possible types of logic gates for boolean circuit: AND, OR, XOR, NAND, NOR, XNOR and NOT. Some of them can be ignored without limiting the expressive power of the circuit. For example {AND,OR,NOT}, {AND,XOR} and {NAND} are *functionally complete* sets, meaning that every boolean function can be implemented with circuits using only gates in one of these sets.

When we are restricted to circuits that have only one output, the circuit is *satisfied*, if there exists an assignment of input variables such that the output is 1. The *circuit satisfiability problem* (CSAT) is an NP-complete decision problem, where the task is to decide, whether a certain circuit is satisfiable. An example of boolean circuit with satisfying assigment is shown in the Figure 2.1. If there exists an satisfying assignment $x = \{x_1, ..., x_n\}$ for the circuit C, the assignment x is a witness of circuit C membership in the language L, that consists of satisfied circuits.

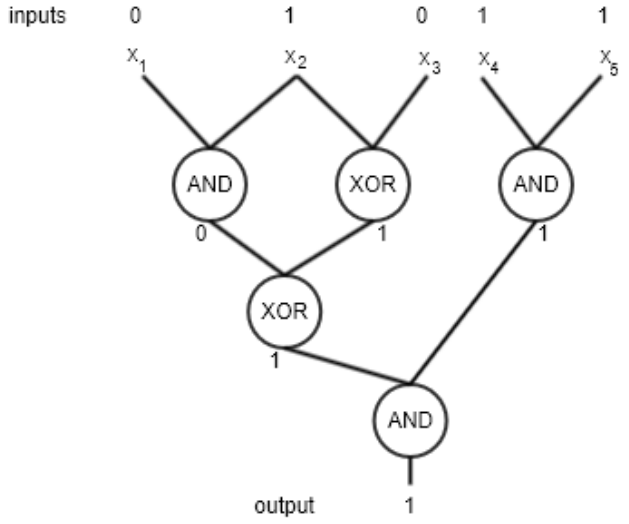

Figure 2.1: Boolean circuit that consists of AND and XOR gates. Input assignment (0,1,0,1,1) is satisfying.

As computation models, boolean circuit and Turing machine cannot be directly compared, because the input size of boolean circuit is fixed, but with Turing machine the input size can be arbitrary. However, the computation of Turing machine M with inputs of lenght n can be simulated by a single circuit C with input length n, that has size (amount of gates) $\mathcal{O}(t(n)^2)$, where $t(n)$ is bound of the running time of M on inputs of length n

[61]. Therefore any polynomial time computation can be performed with polynomial size circuits.

2.2 Cryptographic assumptions and tools

Problems, that are infeasible for adversary, are essential building blocks of cryptographic systems. Usually a legitimate user of the system has an access to some auxiliary information, that enables this user to solve the problem efficiently. For a problem in NP, the auxiliary information could be its NP witness. There should also be a way to create these problems with auxiliary information efficiently - otherwise the system is not useful.

The problem class of NP-complete problems may seem to be promising for cryptographic purposes, the only assumption being that $P \neq NP$. But in general this is not the case, and $P \neq NP$ assumption is not sufficient for cryptography [61]. First of all, NP-completeness implies only, that problem is infeasible *in the worst case*. If it can be solved efficiently in the average case, it is not secure. Secondly, there should be a way to create "hard" instances of NP-complete problems with solutions efficiently. Next we describe an assumption, that is sufficient for many cryptographic problems.

One-way function

Definition 1 (One-way function) *Informally, a function f is* one-way *if it is easy to compute, but hard to invert [62]. More formally, one-way function f is such that:*

- *On input x, there is an efficient algorithm that outputs $f(x)$.*
- *Given $f(x)$ as an input, assuming x is uniformly selected, any efficient algorithm that tries to invert f (to find preimage of $f(x)$) succeeds only with negligible probability.*

Existence of one-way functions is a fundamental assumption for the most of modern cryptography. It is an assumption, because their existence has not yet been proven, and actually such a proof would also imply that $P \neq NP$ [61]. There are, however, strong candidates that have been passed through extensive scrutiny, and have wide support of truly being a one-way function. Probably the most famous example is integer multiplication function $f(x, y) = x \cdot y$, where x and y are sufficiently large primes. It is believed to be a one-way function: the task of inverting f is the *integer factorization* problem.

Trapdoor one-way function

A *trapdoor one-way function* is a one-way function f that is associated with secret information, called as *trapdoor information*, such that inverting f is feasible given that trapdoor information [90].

Strongness of assumptions

It is said that assumption A is *stronger* than assumption B if A implies B, but the converse is false or not known [83]. If the stronger assumption A is found to be false, the weaker assumption B may still be true. It means that cryptographic schemes based on assumption B may still be secure. It is therefore desirable to get a cryptographic scheme proved to secure with the weakest possible assumptions.

Fow example, sometimes we would need a one-way function f, that is also a permutation (i.e. a bijection). This property of *one-way permutation* is stronger than basic one-way property. This is due to the fact that the set of one-way permutations is a strict subset of one-way functions. The existence of one-way functions is a stronger assumption than $P \neq NP$ because the former assumption implies the latter, but the converse implication is not known to be true.

Assumptions are often not comparable in the sense that one implies the another. In that case, qualitative factors, such as what is the confidence in the assumption or how well it has been studied, can be used to evaluate the assumptions against each other [83].

Cryptographic hash function

Definition 2 (Cryptographic hash function [83]) Hash function *is a function that maps an arbitrary size input to an output of fixed size (like 256-bits). A hash function is a* cryptographic hash function *if it is a one-way function and* collision-resistant. *This latter property is defined as follows:*

- *Collision resistance: For a hash function H with a security parameter n, a* collision *is a distinct pair of input values x, y, where $H(x) = H(y)$. Hash function H is* collision resistant *if the probability of finding a collision is a negligible function on the security parameter n.*

Sometimes the assumptions for cryptographic hash function are relaxed by reviewing following property:

- *Second-preimage resistance: A hash function H is* second-preimage resistant*, if for every value x, it is infeasible to find other value y such that $H(x) = H(y)$.*

In the context of cryptographic hash function, the one-way property is often called as *preimage resistance*. Collision resistance implies second-preimage resistance, but not one-way property.

Public key cryptography

Traditionally, cryptography has mainly concerned encryption and decryption of confidential information. Before the mid-70s the only known way to implement this was that the sender and the receiver use the same cryptographic key for both encryption and decryption. Such a key is known as a *symmetric key* (or private key). The key is needed to be exchanged beforehand between the communicating parties, of course securely. This is the Achilles heel of symmetric key cryptosystems. In 1976 Diffie and Hellman introduced an idea of *public key cryptosystem* [40], that uses pairs of keys, where each pair consists of a *public key* that may be distributed freely, and a *private key* that is meant to be known only to the creator of the key pair. Encryption and decryption in public key cryptography can be defined as follows [83]:

Definition 3 (Public key encryption scheme) *A public key encryption scheme is a triple of polynomial time algorithms (Gen, Enc, Dec) such that:*

- *Gen is a probabilistic key generation algorithm: $(pk, sk) \leftarrow Gen(1^n)$. It has a security parameter 1^n as an input, and it outputs a key pair that contains a public key pk and a private key sk. Unpredictability is required for the output keys, so the algorithm must be probabilistic.*

- *Enc is a probabilistic encryption algorithm: $c \leftarrow Enc_{pk}(m)$. It has a public key pk and a message m (called also as* plaintext*) as inputs, and it outputs a* ciphertext *c. For the security reasons the ciphertext needs also be generated probabilistically.*

- *Dec is a determiministic decryption algorithm: $m := Dec_{sk}(c)$. It has a private key sk and a ciphertext m as inputs, and it outputs message m.*

The security parameter n is written as unary notation 1^n when it is a input, for technical reasons, to emphasize that the running time is polynomial. The notation $y \leftarrow A(x)$ is used when y is an output of probabilistic algorithm A.

An another fruit of public key cryptography is a *digital signature*. It is a way to verify authenticity of information in the digital realm, being loosely analogous to handwritten signatures with physical documents.

Definition 4 (Digital signature scheme [83]) *A digital signature scheme is a triple of polynomial time algorithms (Gen, Sig, Ver) such that:*

- *Gen is a probabilistic key generation algorithm, defined as in public key encryption scheme.*
- *Sig is a probabilistic signing algorithm: $\sigma \leftarrow Sig_{pk}(m)$. It has a private key sk and a message m as inputs, and it outputs signature σ.*
- *Ver is a deterministic verification algorithm: $b := Ver_{pk}(m, \sigma)$. It has a public key pk, a message m, and a signature σ as inputs, and it outputs a bit b, where $b = 1$ means "valid" and $b = 0$ means "invalid".*

Also following properties are required:

- *If signature is valid, it must verify: $Ver_{pk}(m, Sig_{sk}(m)) = 1$, except with negligible probability.*
- *It is infeasible to forge signature - i.e. create valid signature for someone's message without her private key. Formally expressed, the signature must be the* existentially unforgeable *under an* adaptive chosen message attack *[72], details omitted.*

Rivest, Shamir and Adleman presented the first implementation of public key cryptosystem [102]. This cryptosystem, known as RSA, is based on the computational hardness assumption of integer factorization. Among the numerous later developed public key cryptosystems we could mention the ElGamal encryption and digital signature schemes named by their creator, Taher ElGamal [46]. These schemes are based on difficulty of the *discrete logarithm problem*. A variant of ElGamal signature algorithm, known as DSA, is more widely used than the original, and currently the latter has a more popular variant called ECDSA, that uses *elliptic curve cryptography*.

Homomorphic encryption

Homomorphic encryption is a form of encryption, which enables certain calculations on encrypted data without decrypting it first. Thus, certain processing can be done on the data while keeping it confidential. In mathematics, a *homomorphism* is a structure preserving map between two algebraic structures of the same type, such as groups or rings. Let $(A, \circ)$, $(B, \diamond)$ be algebraic structures with operations $\circ$, $\diamond$, respectively. Here $\circ$ and $\diamond$ denote abstract operations; they could be additive operation $+$ or multiplicative $\cdot$. A function $f : A \to B$ is a homomorphism, if for all $x, y \in A$ it holds that [113]

$$f(x \circ y) = f(x) \diamond f(y).$$

In cryptography, a desired property is to make operations directly on encrypted data while preserving its integrity, and a cryptosystem with homomorphic properties enables this. In a homomorphic encryption scheme, the function f in the previous equation is a homomorphic encryption function, and algebraic structures A and B are plaintext and ciphertext spaces, respectively.

Cryptographic schemes with homomorphic properties have existed for decades. Sometimes the homomorphism has been deliberately constructed, and sometimes again more or less a byproduct. For example, RSA encryption scheme is multiplicatively homomorphic. Encryption scheme is called *fully homomorphic*, if it is both additively and multiplicatively homomorphic, and the amount of operations is not limited. In 2009 Craig Gentry presented the first fully homomorphic encryption (FHE) scheme [57]. A mere existence of such a scheme was a breakthrough, and subsequently there has been intense research to make FHE schemes more efficient for practical purposes.

Fully homomorphic encryption scheme enables *any* efficient computation to be done on encrypted data, without decryption. Using modulo 2 arithmetic, the addition operation can be represented as XOR gate, and the multiplication as AND gate in boolean circuits. Because these two gates form a functionally complete set, any boolean circuit can be constructed using these two gates. As was stated in Section 2.1, all efficient computation can be done with boolean circuits. Next we examine in more detail how the computation using ciphertexts proceeds on circuit with a FHE scheme.

Let A be a polynomial time algorithm, that is simulated by a circuit family $\{C_i\}$ where i is the input size of the circuit. User Alice possesses an input $x = x_1, ..., x_n$ for the circuit C_n, and she wants to keep it secret. Let $C_n(x_1, ..., x_n)$ denote the output of circuit C_n on input $x_1, ..., x_n$. This time Alice decides to outsource the computation of output.

Using fully homomorphic public key encryption scheme, denoted by $\mathcal{E}$, Alice generates a key pair: a public key pk and a private key sk. Then Alice encrypts her input with encryption algorithm $Enc_{\mathcal{E}}$ that yields ciphertexts $\{\psi_i \leftarrow Enc_{\mathcal{E}}(pk, x_i)\}$. Because the scheme $Enc_{\mathcal{E}}$ is fully homomorphic, it by definition [57] includes an evaluation function $\psi \leftarrow Eval_{\mathcal{E}}(pk, C, \psi_1, ..., \psi_n)$, that on encrypted input computes an encrypted output of circuit. More specifically, when we look at some gate in the first level of circuit, that implements operation $\circ$ (+ or $\cdot$) for plaintext inputs x_i,x_j, the function $Eval_{\mathcal{E}}$ on ciphertext inputs ψ_i,ψ_j, operation $\diamond$, and public key pk yields $\psi_i \diamond \psi_j = Enc_{\mathcal{E}}(pk, x_i) \diamond Enc_{\mathcal{E}}(pk, x_j) = Enc_{\mathcal{E}}(pk, x_i \circ x_j)\}$. This evaluation then continues level by level using intermediate ciphertext outputs as inputs until the evaluation outputs the result for circuit: ψ.

When Alice gets the evaluation result ψ she can now decrypt it: $Dec_{\mathcal{E}}(sk, \psi) = C_n(x_1, ..., x_n)$, except with negligible probability.

Random oracle

In cryptography, term *provable security* refers to a methodology where a security of cryptographic scheme is proved mathematically from some assumptions. If these assumptions relate only to computational hardness of some problems, it is said that scheme is proved secure in the *standard model*. Unfortunately, it is often an insurmountable task to construct a security proof in the standard model. A common technique is to replace a cryptographic primitive used in the scheme with an idealized version to make the proof easier.

Definition 5 *Random oracle A* random oracle *(RO) is an idealized version of cryptographic hash function: a truly random function. It means that for each $x \in X$, the output $H(x)$ of the random oracle H is chosen uniformly from the output domain. By the definition every subsequent call with input value x returns always the same output value $H(x)$.*

There does not exist more efficient way to implement the random oracle than to enumerate output values $H(x)$ for each $x \in X$. If the input domain X is infinite, then the description of random oracle is infinite.

Random oracle is a controversial subject in cryptography. For many cryptographic schemes that have no security proof in the standard model, such a proof have been provided in the RO-model. On the other hand, proof in RO-model does not apply to any practical setting with an implementation of a real cryptographic hash function. Bellare and Rogaway [10] explicitly formulated a methodology, where the security proof for cryptographic scheme

is first done in RO-model, and after that the random oracle is replaced by a "good" cryptographic hash function. This methodology, that has already been used prior to the work of [10], is known hereafter as *random oracle methodology*.

Canetti, Goldreich and Halevi [33] pointed out limitations for this methodology, by providing an example of cryptographic scheme that is secure in RO-model, but insecure in standard model with any cryptographic hash function implementation. Their cryptographic scheme was tailor-made to fulfill their purposes, and also subsequent random oracle counterexamples have been artificial [83]. Nevertheless, it is possible, that some usable cryptographic scheme that is being developed now or in the future passes the examination in RO-model, but proves vulnerable when deployed with concrete cryptographic hash function.

It has been said [83], that security proof for a cryptographic scheme in RO-model implies, that there are no obvious design flaws in the scheme. The vulnerability of such a scheme depends of the actual hash function used. Also, as a general comment [83]:

> "A proof of security in the random-oracle model is significantly better than no proof at all."

3 Review of Blockchain

The term *blockchain* has been quite ubiquitous in the recent years, also as a buzzword. There is no standardized definition for it, but it could be described as a distributed ledger or records called blocks, of which tampering has been made difficult by cryptographic means, and where decentralization is achieved by variety of different, implementation specific strategies.

The most famous, and also the first implementation of blockchain is Bitcoin. It is often used as an archetypal example, where other blockchain implementations are compared. We use the Bitcoin as a reference to introduce the structure of blockchain.

Bitcoin, in addition of giving a birth to a concept of blockchain, is the first successful decentralized cryptocurrency. It has no central authority that manages the creation and transactions of its currency, units of bitcoin[†]. Bitcoin is instead managed by equally privileged nodes of *peer-to-peer* (p2p) network, and it uses cryptography to secure its functioning.

Bitcoin white paper, titled *Bitcoin: A Peer-to-Peer Electronic Cash System* [94] was published in 2008 by pseudonym Satoshi Nakamoto on a cryptography mailing list. Nakamoto implemented Bitcoin software, and the first block, also known as *genesis block* was hard-coded on the code base timestamped on 3 January 2009. It is the only block introduced this way, and all subsequent blocks have been created by a collaborative and competitive process called mining. Because the Bitcoin was released as open source, anyone could download the software and participate to its p2p-network, by broadcasting and validating bitcoin transactions.

Bitcoin is a *cryptocurrency*. Definition of cryptocurrency in Merriam-Webster online dictionary is [37]:

> "any form of currency that only exists digitally, that usually has no central issuing or regulating authority but instead uses a decentralized system to record transactions and manage the issuance of new units, and that relies on cryptography to prevent counterfeiting and fraudulent transactions."

[†]Bitcoin is usually written lowercase, when it is being referred to as currency, in contrast to uppercase notation when referred to as protocol, software or network.

The misuse of ordinary, non-cryptocurrencies must be made difficult by the governing bodies, such as banks. But it is not technologically impossible, so the law enforcement is needed as a backup the prevent the system from breaking. Cryptocurrencies do not (in principle) have such a backup, instead they rely solely on technological means, cryptographic mechanisms, trying to make misuse impossible.

3.1 Structure of Bitcoin

In this chapter we study the structure of Bitcoin and how it utilizes certain cryptographic tools. The most important tools are cryptographic hash functions and digital signatures [95].

Cryptographic hash functions are used in multiple roles in Bitcoin protocol. The function used is *SHA-256* hash function, that belongs to *SHA-2* function family that is designed by NSA [107]. At the time Bitcoin was designed SHA-256 was among the strongest hash functions [95], and while it is still viable, later have been developed stronger ones, like the *SHA-3* family.

Hash pointer [95] is a pointer to data associated with a cryptographic hash of that data. If we use a hash pointers in linked list instead of regular pointers, we get a list where each item points to previous item similarly as with regular pointers. Hash pointers gives linked list an additional *tamper-evident* property: Assuming we have stored the hash pointer of the list head, we will find out, if any items of the list are tampered. If adversary modifies some list item, then due to collision resistance, the hash pointer of next item will not match, and he would have to modify it also. This effect propagates to the head of the list, as shown in Figures 3.1 and 3.2. By replacing the term *item* with the term *block*, we have now described the data structure of a *blockchain* at abstract level. This type of data structure, cryptographically secured chain of items was described in 1991 by Haber and Stornetta [79]. They used it to cryptographically secure timestamping of documents, chaining the documents with timestamps by hash pointers. Later they proposed more efficient data structure to be used [6]. There collections of temporally proximate documents would be arranged into units, whose internal structure would be a binary tree of hash pointers, providing fast verification of membership. This tree data structure is called *Merkle tree* after its inventor, Ralph Merkle [91].

Note that the term *blockchain* (or *block chain*, both spellings have been used) has not been used prior to Nakamoto's work [94]. This term has been used afterwards, as a loose

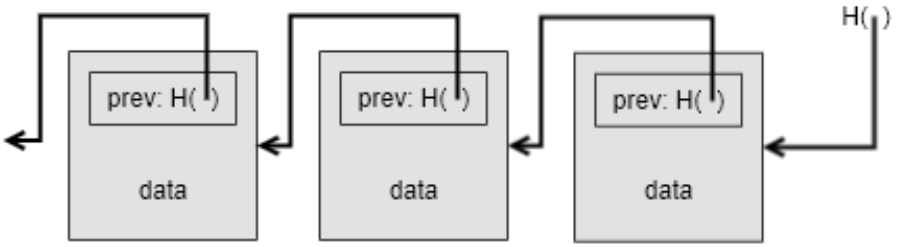

Figure 3.1: Linked list with hash pointers, blockchain's core structure. Here H( ) denotes hash pointer with arrow to data

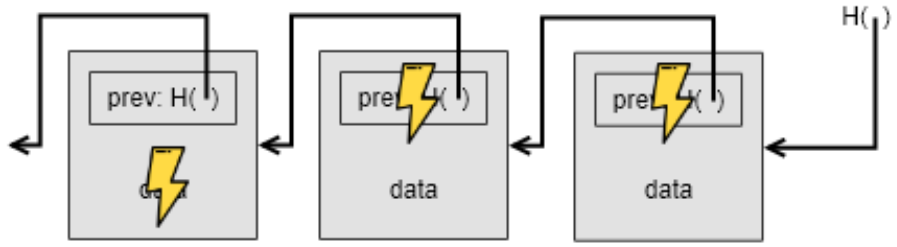

Figure 3.2: If data is tampered somewhere in the list (left item in the picture), the evidence will propagate to the head of the list

umbrella term, lacking formal definition [96]. In addition to data structure properties, also other features has been associated to it, like mechanisms relating to decentralization.

Bitcoin uses both the blockchain and Merkle tree data structures. Individual transactions are collected to blocks, whose internal structure is a Merkle tree. Those blocks then form a blockchain.

Bitcoin uses digital signature scheme. In practice, the public keys associated with the scheme can be treated as identities. Bitcoin has a notion of *address* that is a hash of a public key. The address is a shorter version of the public key, retaining its uniqueness due to the collision resistance of the hash function.

Addresses relate to Bitcoin transactions that we will examine next. A transaction consists of inputs and outputs. Input must refer to *unspent* output o_{ti} of some other transaction t, reference is the hash of t. Outputs are addresses, associated with the values in bitcoins. The sum of output values must not exceed the sum of input values. Transaction must be signed by the private key that corresponds to public key defined in the output o_{ti}. If there are multiple inputs, then transaction needs to be signed by all of the private keys of the

inputs* (see Figure 3.3). Thus, the transaction verification includes checking the validity of the signatures, and that the inputs are unspent, i.e., they have not been used in other transactions. Also the causality is maintained: the transaction cannot be verified, until its inbound transactions are verified.

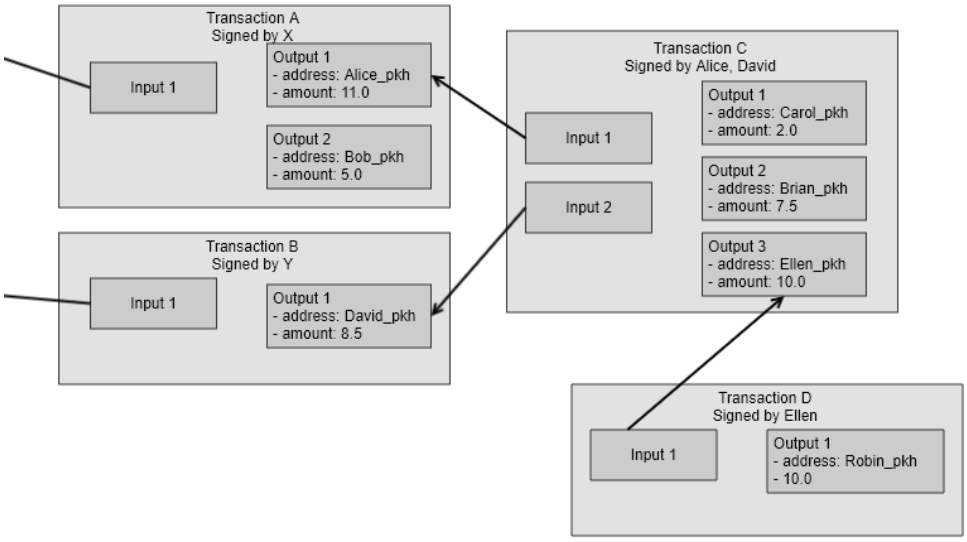

Figure 3.3: Alice signs transaction C with her private key, spending output 1 of transaction A that is addressed to her public key (_pkh denotes public key hash). Transaction C needs to be also signed by David, because his unspent output is spent. Note that the names used here are just pseudonyms of public keys, they could all belong to same person or entity.

3.2 Distributed consensus

In the previous chapters we have outlined how the Bitcoin transaction data could be stored and validated, using hash pointer data structure and digital signatures. One key issue that needs clarification, is how this collection of transactions is maintained. In centralized systems there is one party, that maintains the integrity of the system. Bitcoin is a *distributed system*, and in that domain the integrity maintenance is closely connected to the *consensus* problem. Nodes in distributed systems should act so, that the consensus is reached. Consensus is described as an agreement among multiple nodes for a single

*This is a little simplified description, and is valid in most of the cases. Bitcoin input signature and output public key portions are actually *scripts*, written by specifically built stack-based scripting language called *Script*.

data value. Nodes can be *honest*[*], i.e. following the prescribed rules or protocol, or *faulty*. The goal is to design the system so that it fulfills the requirements of *distibuted consensus protocol*.

Definition 6 (Distributed consensus protocol) *[59] Distributed system contains n nodes. Every node has* initial value *in a mutually agreed domain. Despite the occurrence of failures, nodes eventually agree upon an irrevocable decision value, and following three conditions are met:*

- *Termination: Every honest node must eventually decide some value.*
- *Agreement: The decision of every honest node must be identical.*
- *Validity: If every honest node has the same initial value v, decision value must also be v.*

Let us have a look at the Bitcoin, what are the issues on which consensus is to be reached. Users of Bitcoin broadcast their transactions to p2p network. Nodes must agree on these broadcasted transactions, which transactions are added to the blockchain and in which order. In general the consensus problem is far from trivial to solve. If there would be a certainty, that every node operates honestly, the problem would be much simpler. In the theory of distributed systems there are categorized different failures, that faulty node may exhibit. For example a *crash failure* is where node just stops, but behaves otherwise honestly. Failure type that is most difficult to manage is *Byzantine failure*. It implies any kind of arbitrary behaviour from a faulty node, including that the faulty node sends different kinds of data to different nodes, and also the case that the faulty node might be malicious.

The term *Byzantine* originates from the *Byzantine generals problem*, an illustrative formulation of the problem regarding tolerance for arbitrary fault, devised by Lamport et al. [86]. In their problem setting a group of generals of the Byzantine army tries to reach a consensus on the attack plan, whether attack or retreat, through pairwise messaging[†].

[*]Alternative terms for *honest* are the terms *correct* and *non-faulty*. Often term *node* is replaced by the term *process*.

[†]More specifically the message model is *oral messages model*: In that model 1) the message is delivered correctly. Imperfect message delivery is indistinguishable from traitor, hence it can be considered as an additional traitor to the problem. 2) The receiver knows who sent it. 3) The absence of a message can be detected.

Some of the generals can be traitors, i.e., having Byzantine fault. They showed that if there are k traitors, there need to be at least $2k+1$ honest generals in order to reach a consensus. They also described the protocol for reaching consensus, thus giving an example of how Byzantine fault tolerance, the ability to tolerate Byzantine faults can be reached in their scenario. The result of [86] is one of the famous impossibility results that give bounds to fault tolerance in the field of distributed systems. There are other results for different problem settings, and it is not always clear, whether some result is applicable to given scenario. In Byzantine Generals problem the communication is synchronous, i.e. having bounded delay. A setting, where the communication is asynchronous has been studied, e.g., by Castro and Liskov [34]; they implemented Byzantine fault tolerant network file system (NFS) service.

3.3 Bitcoin consensus mechanism

Bitcoin utilizes a consensus mechanism called *proof-of-work* (PoW). The idea of proof-of-work is, that obtaining a resource or service requires a computationally expensive puzzle (PoW function) to be solved. Asymmetry is an essential feature: PoW function should be moderately hard (yet feasible) to solve, but easy to verify. The concept was first proposed by Dwork and Naor [43]. Their main use case was preventing junk email, by requiring email sender to solve some computational task prior to sending. Later Back studied similar use case by applying his algorighm *Hashcash* [5] as a PoW function. The Hashcash aims to find a partial hash collision for the target value, and this type of PoW function is also utilized in Bitcoin. Nodes compete with each other of the permission to broadcast next block to blockchain. This permission is obtained by completing a proof-of-work.

Definition 7 (Proof-of-work (in Bitcoin)) *[95] Each node in Bitcoin network receives broadcasted transactions, and after verifying them, collects those transactions to their own template of the next block. If node can fulfill following condition by finding sufficient value for nonce*, then it can broadcast this new block to other nodes.*

$$H(nonce|prev\_hash|transactions) < target$$

The target is a value adjusted by Bitcoin network. The target value is dynamically pa-

*In cryptography the term *nonce* refers to random number that is (or can be) used only once

rameterized so, depending on the network's current computing power*, that the average solving time for the "winner" would be around 10 minutes. When writing this, the target in block 616913 is

$$0000000000000000000121ad4000000000000000000000000000000000000000$$

and therefore approximately only 1 out of $6,7 * 10^{22}$ of all possible hash values are below the target. The properties of the hash function ensures, that there is no better method for finding a suitable nonce, than brute-force search. The resulting hash value becomes the header (id) of the new block. Nodes can easily verify that the hash value meets the demands, given the nonce, target and other block data. This activity of validating transactions and solving proof-of-work is called *mining*, and the nodes that execute it are called *miners* .

The proof-of-work seems to solve the consensus problem, which relates to agreeing on new blockchain transactions. The winner node of the "mining lottery" gets its block template as the new block to the blockchain. Other nodes verify the validity of the block b and its transactions. If everything is ok, they start to build their block templates on top of the block b (setting hash of block b as the prev_hash). The Bitcoin network is not perfect, there are latencies, and in any given moment, the exact set of unverified transactions, called as a transaction pool, may vary between nodes. So the particular transaction might not be added to the next possible block, but eventually the unverified transactions are verified and added to blockchain.

Incentives

Bitcoin incorporates incentives in two ways [18]. First, there is the block reward. The first transaction in every block is a special *coinbase* transaction, which is added by the miner of the block. Its transaction value includes a fixed block reward (currently 12,5 BTC), and the miner chooses the output address (which usually belongs to him). This block reward creates bitcoins "out of thin air" and is the only way to create new bitcoins.

Secondly, the transaction may include a *transaction fee*. If the sum of transaction outputs is less than the sum of inputs, the difference, which is the transaction fee, belongs to

*Every 2016 blocks nodes calculate independently is the timestamp difference of the 2016 blocks smaller or greater than the target value, that is exactly two weeks, and difficulty is adjusted correspondingly.

miner. Miner is free to choose which transactions to be added to the block, and the transaction fee affects to the miners' willingness to include the transaction in the block. So the fee may affect how quickly the transaction gets its confirmation. Fees are often dynamically adjusted based on current network traffic. Compared to the block reward, the total amount of transaction fees per block has been so far small, but their relative importance will probably change in the future as the block reward decreases according to a predetermined schedule. Being a Bitcoin miner causes expenses with the needed electricity and hardware. The block reward with the variable exchange value to regular currencies more or less balances the efforts. There is also economical incentive to be a honest miner. Let us say some miner manages to mine a block, where it sets a non-standard, bigger block reward. Honest miners will reject this block, so it will not end up to the blockchain, leaving the non-honest miner empty handed. Next we analyze specific threats to Bitcoin's decentralization, and how it can cope with them.

Forking

It is possible, that two miners broadcast their blocks ($block_a$ and $block_b$) almost simultaneously, both referring to the same parent block. Only one of them can remain in the long-term consensus chain*. How can the network agree on which block to build on? For a while there can be a situation, where part of the nodes build on the $block_a$, and other part on the $block_b$. But eventually one of the branches will become longer, and nodes are following a heuristic to build on the longest valid branch [18]. Transactions of the *orphaned* block(s), that are not yet in blockchain will be broadcasted again.

Double spending

Consider the case, where Bob sends transaction t_1 as a payment of some product (worth x BTC) to some merchant Mike. Let us denote it as $t_1 = \text{Bob} \xrightarrow{x} \text{Mike}$. Mike sees that the transaction is broadcasted and decides to give the product to Bob. Right after that Bob sends another transaction $t_2 = \text{Bob} \xrightarrow{x} \text{Bob}_2$, where he plans to transfer the input of t_1 to another address (it could be his or someone else's). Now that input is *double-spent*.

*The block structure of a blockchain is based on consensus, so at least in theory changes can occur anywhere. Because of the consensus rules used, the probability that a particular block will not remain permanently in the blockchain decreases exponentially as we move towards older blocks. Loosely speaking, the *long-term consensus chain* [95] refers to that part of the blockchain that is no longer subject to change.

In this case t_1 was not yet confirmed (not included in a confirmed block), so it is possible that t_2 will be verified in the next block instead of t_1 (both cannot be verified, the miner probably verifies the one it sees first, rejecting the latter). This situation could have been avoided, if the merchant had waited until the transaction had been confirmed. Only one confirmation is not usually considered safe. For example the transaction t_1 could be in the soon-to-be orphan block of the forking example, and the transaction t_2 could still displace it from the long-term consensus chain. Some rule of thumb is to wait 6 confirmations to consider transaction secure [18] (depth of transaction is then 6 blocks), but this of course depends on the circumstances.

The 51 percent attack

A fundamental requirement to Bitcoin protocol to work is that the majority of miners are honest, or more specifically, the majority of computing power belongs to honest miners. Conversely, a group of colluding miners controlling the currency with their mining power could, for example, prohibit certain transactions. Even if such a group would act honestly, the mere existence of it could destroy confidence in the currency, which, in essence, would not be anymore decentralized. Obtaining the majority of computing power is generally considered prohibitively expensive. On the other hand, perhaps less is enough. Bitcoin miners often belong to some mining pool, which shares rewards proportionally to members contribution. Eyal and Sirer pointed out [47] that certain selfish strategy for a mining pool could provide to them better revenues. This could even lure other honest miners to join their pool, and before long, the pool will grow into majority.

3.4 Anonymity

The transaction data of Bitcoin is public. Transaction values, how many BTCs are sent to output addresses, are plaintext. Input and output addresses of transactions, i.e. the senders and receivers of transactions, are hash values of public keys, so they look random. Thus, addresses are *pseudonyms*, "pseudo-identities" of users. Pseudonymity alone does not guarantee privacy. In pseudonymous systems the privacy depends on the *unlinkability* of the actions where these pseudonyms are involved [100]. Anonymity, that is a strong notion of privacy, requires both pseudonymity and unlinkability.

Two items are unlinkable for a specific adversary, if their occurrence in the system does not

increase the probability that they are related from adversary's point of view, in respect to the adversary's a-priori knowledge [100]. Narayanan et. al [95] enumerated requirements that Bitcoin should fulfill to reach unlinkability. These include the difficulty to link together different addresses that belong to the same user, as well as difficulty to connect transactions that are made by the same user. Third requirement concerns the difficulty to connect the sender of the payment to its receiver.

In the case a payment is executed on a single transaction, the sender and the receiver are of course immediately tied to each other. The payment can also be a chain of multiple transactions, which makes the connection less clear. Nevertheless, it may be possible to deduce from the matching amounts of bitcoins from the input and output of the transaction chain, and from the temporal proximity of the transactions, that these in fact form a single payment.

On a larger scale, the analysis of transaction graphs [89, 104], with the possible help of coincidental real-world events has been used successfully to reveal identities in some cases. These revelations of identity have been made, for example, in criminal investigations, such as in the Silk Road case [73]. The data put to blockchain stays there permanently. Leakage of some casual user's address identity reveals some of his transactions details, regardless of how many years have passed.

Mixing services

One way to mitigate these linkability issues is to use *mixing services*. A purpose of mixing is to break connections between Bitcoin addresses, by taking a set of inputs and outputs from *intended* transactions from different users, and then shuffle the inputs and outputs and make the actual transactions. Making sure, of course, that everyone gets the right amount of bitcoins to the right address. Mixing services are, generally, privately owned services that take some commission, "mixing fee", of mixed bitcoins. Mixing services are categorized to centralized and decentralized solutions.

To give a simplified description of the operation of the centralized mixing service: first, user A sends bitcoins to an address that belongs to the mixing service while specifying the desired output address. Next the mixing service collects a set of these transaction requests made by other service users to mix the inputs and outputs of transactions. After that the mixing service executes transaction to the output address of user A with the input address of other user [95]. The users of the service must trust that they get their

bitcoins back, and that their Bitcoin, and possibly also IP addresses which are inevitably known by the service are not leaked outside. There have been fraudulent mixing services that have stolen bitcoins [36].

A good yardstick to evaluate a transaction mixing operation is the size of the *anonymity set* it provides. For a specific subject, its anonymity set is the set of subjects among which this specific subject is not identifiable [100]. If the adversary tries to figure out the output address o for the input address of specific user that is used in mixing, then the anonymity set of o consists of other output addresses that are involved in the mixing. The anonymity set shrinks if more complex adversary model is considered. For example, transaction timing patters can be exploited in graph analysis to reduce the anonymity set. Therefore, mixing services might provide the possibility to define multiple output addresses, each with its own custom delay for the output transaction.

Most decentralized mixing services are based on the CoinJoin method [87, 19], that enables users to anonymously combine their intended transactions to a one large transaction containing all the inputs and outputs. Technically, the CoinJoin is implemented using a centralized service as a coordinator, at least in the most popular services, like Wasabi Wallet and Whirlpool. The main claim of the decentralization is the fact that users mix bitcoins with each other without having to hand them over even temporarily to an intermediary.

There are dozens of active mixing services, but only a small number of them are generally considered reliable in the Bitcoin community [36]. While the statistics of mixing service popularity in general is difficult to get, the CoinJoin transactions can be quite reliably identified from the blockchain, and according to an analysis conducted in 2019, approximately 4 percent of all transactions were CoinJoin transactions [35].

The mixing services add noise to the transaction graph, thus reducing linkability for users and addresses that are involved in these mixing transactions. The risk of deanonymization still exists. Services that require identity verification, may be a risk factor. These include bitcoin exchanges where bitcoins are exchanged to fiat currencies or to other cryptocurrencies, and also include many bitcoin wallet services that are used to ease bitcoin payments. In the event of a possible data breach, the user data and the addresses associated with it may be compromised.

Protocol level privacy

Dandelion protocol [22] is one of the proposals that aims to increase Bitcoin privacy. It is a privacy enhancement to Bitcoin's p2p network. Other proposal is the utilization of the Schnorr signature algorithm [88] that is more tightly tied to Bitcoin's roadmap than Dandelion. The Schnorr signature algorithm is supposed to partially replace the current ECDSA algorithm. By using Schnorr signatures it is possible to implement "key aggregation" [88] when transaction has multiple inputs. This means that instead of having multiple signatures, one for each address, a single signature can be created by all private keys involved, and it can be verified by using an aggregation of the corresponding public keys. This saves space in transaction, so the transaction fee for users is lower in this combined transaction than what it would have been if each user had done separate transactions. Thus, the use of CoinJoin would also have an economic incentive. Overall, future improvements to Bitcoin appear to have a positive impact on its privacy.

The fact that the transaction amounts are plaintext diminish the overall privacy. Sometimes even acute issues can occur. For example, If Bob has 100 BTC in an address that he uses for a payment of 0,01 BTC to merchant Mike, then Mike knows that Bob has at least a moderate amount of money in his possession. With careful handling of bitcoins, such situations can be avoided. In general it can be said that maintaining privacy in Bitcoin requires active measures. The protocol as such does not guarantee privacy.

It is imperative that Bitcoin transactions can be verified, and it may also seem inevitable that information that compromises privacy will be revealed at the same time. There are, however cryptographic schemes that enable verification without leaking any additional information. They belong to a category of *zero-knowledge protocols* and they are studied in the next section. The prospects of zero-knowledge protocols for blockchain applications were quickly recognized, and some examples are reviewed in Section 5.

4 Zero-knowledge proof (ZKP)

Normally, when a statement is proved, the proof discloses some additional information besides the truth value of the statement (*true* or *false*). An usual way to prove a statement in NP is to provide a witness for that statement. To prove that a boolean circuit is satisfiable it suffices to provide a satisfying assignment of its inputs. The "proof" of Bitcoin transaction validity for miner includes information of unspent outputs it refers, public keys, transaction amounts and signatures. In 1985 Goldwasser, Micali and Rackoff developed theory of *interactive proof systems* [71]. An important notion in their study, using methods of complexity theory, was to analyze the additional knowledge that the proof reveals. Interesting special case is when there is no additional knowledge, in which case the term *zero-knowledge proof* (ZKP) is used.

Informal example: "Colour-blind friend"

First we have a look at an informal example of a zero-knowledge proof. This example of proving to colour-blind friend that two balls are differently coloured occurs in many ZKP introductions. The "prover", who is not color blind, has two balls, red and green, and he tries to convince the colour-blind "verifier" that these balls are indeed differently coloured. In order to keep the proof as zero-knowledge, the prover does not want to reveal any additional information - which ball is red and which is green. The "proof system" is to repeat following procedure sufficiently many times.

1. Verifier holds the balls one in each hand (without differentiating their colours) so that the prover sees them, and then puts his hands behind his back. Then he *randomly* either switches the balls in his hands or not (prover does not see that). After that he brings the hands in the front of him and asks prover: "Did I switch the balls? ".

2. Prover answers truthfully.

Prover's probability to guess right in one round is $\frac{1}{2}$, so when this procedure is repeated k times, the probability to succeed by guessing in each k round is $\frac{1}{2^k}$. With sufficiently large k the verifier becomes convinced, that the prover is not just guessing, but actually

sees by their colour whether the balls have been switched. Yet the verifier has not learned which ball is red and which is green.

In the following chapter we go through the interactive proof systems, in order to understand interactive zero knowledge proofs. Later, we discuss non-interactive zero knowledge proofs, a variant which is essential when implementing zero-knowledge proofs to blockchain technologies. The limitations of zero-knowledge proofs and their challenges in practical implementation are also considered.

4.1 Interactive proof systems

Goldreich [61] primed interactive proofs systems comparing them to proofs in mathematics. Traditionally in mathematics, proofs have a static nature. They are considered as fixed objects that have been written once. A proof gives absolute certainty of the statement it proves. On the other hand, in daily life proofs often have dynamic nature and they are constructed e.g. by legal process in the courtroom. Chance of error is usually acknowledged. Interactive proof systems embrace the latter interpretation of proof.

Interactive proof system (IPS) involves two parties: a *prover* (P) and a *verifier* (V). The prover is not trusted by the verifier, and its computational resources are not limited. Verifier on the other hand has limited resources, its running time is polynomial. Recall that the complexity class NP is a set of decision problems *solvable* in polynomial time (efficiently) by a *non-deterministic* Turing machine (NTM). These solutions are verifiable in polynomial time (efficiently) by a *deterministic* Turing machine (TM). As described in Section 2.1, this gives an equivalent definition for NP. By these terms, NP can be viewed as a class of proof systems, where interaction consist of prover (NTM) providing a proof, and a verifier (TM) verifying it [61].

Basic properties of any proof system are *completeness* and *soundness*. These are defined already in mathematical logic where with logical systems, completeness means that every valid statement is provable, and soundness means that no invalid statement can be proved. In interactive proof systems these properties are relaxed by replacing certainty with “very high“ probability. Randomness is included in the system by allowing verifier to “toss private coins“. The definition of IPS includes requirements for the computational model. Prover and verifier are a pair of *interactive Turing machines*, which for instance includes machine-to-machine communication tapes, that enables interaction (messages) between machines (see Figure 4.1). A random tapes of verifier and prover provides randomness

to their computation and mutual messages. The random tape makes prover and verifier *probabilistic Turing machines.*

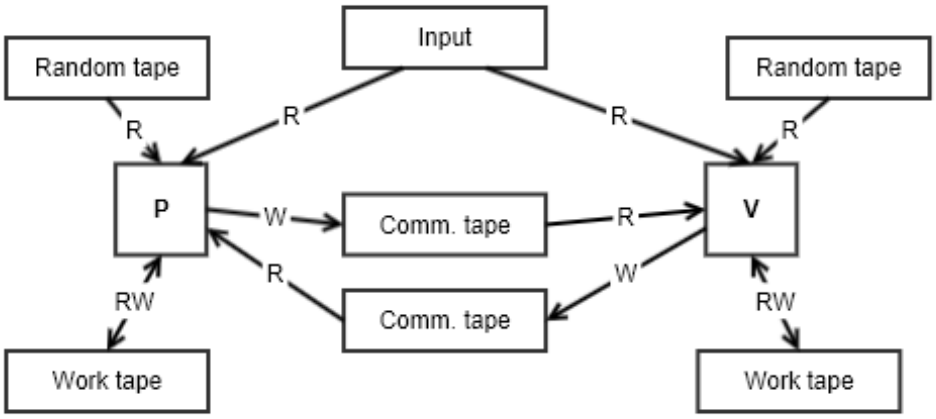

Figure 4.1: A pair of interactive Turing machines (P, V) with conjoined communication tapes, and common input tape, forming a "interactive protocol" for IPS. R, W, and RW denotes read-only, write-only and read/write heads, respectively. Figure adapted from [71].

Definition 8 (Interactive proof systems) *[71] Let $(P,V)(x) \in \{0,1\}$ denote verifier V decision value for input x and prover P, when the verification procedure, i.e., interactive computation ends (1 = accepts, 0 = rejects). A pair of interactive Turing machines (P,V) is called an* interactive proof system *for language L, if V is polynomial-time machine and the following two conditions hold:*

- *Completeness:* $\forall x \in L$: $Pr((P,V)(x) = 1) > 1 - \nu(|x|)$
- *Soundness:* $\forall x \notin L, \forall P'$: $Pr((P',V)(x) = 1) < \nu(|x|)$

Where ν is a negligible function.

Note that the soundness condition concerns all potential provers, in contrast to completeness, which refers to the specified prover P. There are alternative definitions of IPS [61] where the completeness and soundness error bounds are not negligible, but are instead some fixed values, like $\frac{1}{3}$. These errors bounds can be made negligible by repeating the (randomized) verification procedure, and taking a majority vote of the outcomes.

Complexity class IP (Interactice Polynomial time), is defined as the class of languages for which there exists an interactive proof system [71]. Equivalently, it is the class of decision

problems solvable by an interactive proof system. It is generally believed, although not proved, that the IP is larger than NP. There are examples of languages that are in IP, but not known to be in NP. One example is the language of pairs of graphs that are non-isomorphic to each other [66]. Without verifier's randomness the complexity class collapses to NP [62]. It has been shown [52] that every language that has IPS, has one with perfect completeness, so removing completeness error does affect to the complexity class IP.

Independently, about the same time when IPS was introduced, Babai [4] published a similar proof system called *Arthur-Merlin protocol*, which can be considered as a variant of IPS. In Arthur-Merlin protocol the "coin tosses" of verifier are public (to prover), so it has been referred as a *public coin* protocol, in contrast to IPS with private coins. In the public coin protocol the verifier sends uniformly chosen random message, often referred as a "challenge", to the prover in each round. The challenge space, whereof the challenge is chosen is correspondingly defined, and it varies depending on the setup of the protocol. It can be just a one bit $\{0, 1\}$, or substantially larger.

It may seem that the public coin property significantly impairs verifier's ability to expose a cheating prover (hindering the soundness), as the verifier cannot hide anything from the prover. However, these public coin systems have been proved [69] to be equally powerful than private coin systems, when it comes to language recognition. Every language that can be recognized by an IPS, can also be recognized by an Arthur-Merlin protocol.

4.2 Interactive zero-knowledge proofs

Interactive proof system for language L is said to be *zero-knowledge* if for every $x \in L$, the verifier gets from prover essentially no other knowledge, than the fact that $x \in L$. In addition to above-mentioned completeness and soundness, there is a third condition for the zero-knowledge. Verifier might try to get prover to reveal something, so the zero-knowledge is the prover's objective. The zero-knowledge property of the prover P can be informally described as "whatever can be efficiently computed after interacting with P on input $x \in L$ can also be efficiently computed from x (without any interaction)" [71]. This means, that for every $x \in L$ and for every verifier V^* there exists a simulator (algorithm) M^* so that the output of V^* (after interacting with P on common input x), and the output of M^* on input x are "computationally indistinguishable". More formal description follows next.

The simulator M^* is assumed to be a probabilistic polynomial time Turing machine similarly as the verifier V^* is. This captures the notion of "what can be efficiently computed". Because the machines M^* and V^* are probabilistic, their outputs are *random variables* R. Furthermore, when we use the set L as an index set, the outputs of M^* and V^* can be described as a set of random variables, that are indexed by $x \in L$. The outputs of M^* and V^* are then denoted as $\{R_x\}_{x \in L}$, which is in cryptography often called as a *probability ensemble*.

Definition 9 (Computational indistinguishability) *[61] Let a* distinguisher D *be a probabilistic polynomial time algorithm. Two probability ensembles* $\{R_x\}_{x \in L}$ *and* $\{S_x\}_{x \in L}$ *are* computationally indistinguishable *[61] if for every* distinguisher D*, for every polynomial function* p*, and for all sufficiently long* $x \in L$ *it holds that*

$$|Pr[D(R_x) = 1] - Pr[D(S_x) = 1]| < \frac{1}{p(|x|)}$$

Where $D(R_x) \in \{0, 1\}$ *is the output of the distinguisher* D *on a polynomially time processable sample of* R_x*. Output* $D(R_x)$ *is probabilistic, because* D *is probabilistic.*

This definition means, that there is no efficent algorithm D that can tell the difference between ensembles R and S, excluding a negligible probability.

Definition 10 (Zero-knowledge) *[71] [61] Let* (P, V) *be an interactive proof system for language* L*. System* (P, V) *is a* zero-knowledge *proof system (ZK proof system), if for every probabilistic polynomial time verifier* V^* *there exists a probabilistic polynomial time simulator* M^* *such that the following two probability ensembles are computationally indistinguishable:*

- $\{(P, V^*)(x)\}_{x \in L}\}$*, which is the output of verifier* V^* *after it interacts with the prover* P *on common input* x
- $\{M^*(x)_{x \in L}\}$*, which is the output of simulator* M^* *on input* x

Goldreich, Micali and Widgerson showed [66] that all languages in NP have zero-knowledge proof systems, provided that one-way functions exists. First they presented zero-knowledge proof system for graph 3-colorability (G3C). This is an NP-complete problem, so every language L in NP can be reduced to it. Next it was sufficient to show that in addition to G3C, the zero-knowledge proof system can also incorporate the reduction. Zero-knowledge proof of 3-colorability (sketch of it) is presented in next chapter.

Zero-knowledge proof of graph coloring

An important ingredient of the G3C zero-knowledge proof is the concept of *commitment*. In cryptography, the commitment is an action, where one party commits a value while keeping it secret from others. It is digitally analogous to a physical world scenario, where one party puts a value to sealed envelope. Later, when the commitment is revealed, it can only yield the value, which was already determined when the commitment was done. Not only with ZK proofs, the commitment scheme is widely used construct in cryptographic protocols.

We review commitment scheme informally, following notation from [95], more formal treatment can be found in [61]. A *commitment scheme* includes a polynomial time algorithm *commit* that outputs a commitment c for the input values m and r, where m is the value to be committed, and r is a random value. Commitment scheme fulfills following properties:

- Hiding: For a commitment value c it is infeasible to find a pair (m, r) such that $commit(m, r) = c$.
- Binding: It is infeasible to find two pairs (m, r) and $(m^{'}, r^{'})$ such that $m \neq m^{'}$ and $commit(m, r) = commit(m^{'}, r^{'})$.

In addition to *commit* algorithm, commitment scheme defines also how the committed value m can be revealed. In interactive commitment scheme the sender of the commitment reveals the committed value m by sending the pair (m, r) to the verifier who checks that the *commit* algorithm yields the commitment value $c = commit(m, r)$.

The language *graph 3-coloring* (G3C) consists of all graphs whose vertices can be colored with three colors so that any two adjacent vertices do not have the same color. A graph $G = (E, V)$ is *3-colorable*, if there exists a mapping $\varphi : V \to \{1, 2, 3\}$ such that $\varphi(u) \neq \varphi(v)$ for every edge $(u, v) \in E$.

The idea of the ZK proof for G3C is the same as with many other ZK proofs. The proof is divided into multiple sub-proofs, each of which increases the verifier's trust to the validity of the statement, but individually does not reveal any additional knowledge. When the verifier checks enough of these sub-proofs, he will be convinced of the statement. Next we define the protocol of the "sub-proof", which is also illustrated in Figure 4.2.

Construction of the ZK proof for G3C

- Common input: graph $G = (V, E)$, where $|V| = n$, and $V = \{v_1, ..., v_n\}$.
- Prover's first step: Let φ be a 3-coloring of G. The prover selects random permutation π of the set $\{1, 2, 3\}$. Denote value $\varphi_\pi(v_i) := \pi(\varphi(v_i))$ for each $v_i \in V$, which is the permutated color of v_i. For every $v_i \in V$ prover assigns secret random value s_i, and makes commitment c_i using function *commit* for each $v_i \in V$: $c_i := commit(\varphi_\pi(v_i), s_i)$. The prover sends committed values $\{c_1, ..., c_n\}$ to the verifier.
- Verifier's first step: The verifier selects randomly an edge $(v_i, v_j) \in E$ and sends it to the prover.
- Prover's second step: The prover reveals committed colors of v_i and v_j by sending values $(s_i, \varphi_\pi(v_i))$ and $(s_j, \varphi_\pi(v_j))$ to the verifier.
- Verifier's second step: The verifier checks whether the revealed values corresponds to the committed values c_i and c_j, and whether revealed colors differ: $\varphi_\pi(v_i) \neq \varphi_\pi(v_j)$ and belong to $\{1, 2, 3\}$. If these conditions are fulfilled, then verifier accepts, otherwise verifier rejects.

The commitment scheme allows the prover to offer to the verifier a free choice of the revealed edge, while the verifier can be convinced that the prover cannot cheat (change the answer) after he has made the choice.

The completeness of this protocol is perfect, if graph G is 3-colorable, then the prover can commit a 3-coloring of G, and every edge the verifier chooses is revealed as two-colored with colors in $\{1, 2, 3\}$. The soundness error is $1 - 1/|E|$. In the worst case 3-coloring applies to the graph G with all but one edge, and the probability that verifier selects that edge is $1/|E|$. When this sub-proof is repeated $n|E|$ times, the soundness error goes down to $(1 - \frac{1}{|E|})^{n|E|} \approx e^{-n}$, which is negligible.

The zero-knowledge property can be intuitively seen to be valid, because each sub-proof shows only whether vertices have same color or not. The revealed colors are random, and do not necessarily correspond to 3-coloring of the graph (it if exists). Because the color permutation is drawn for each sub-proof, verifier is not able to collect the information of 3-coloring with subsequent repeats of the protocol - the colors are unrelated. Formal proof through computational indistinguishability is omitted.

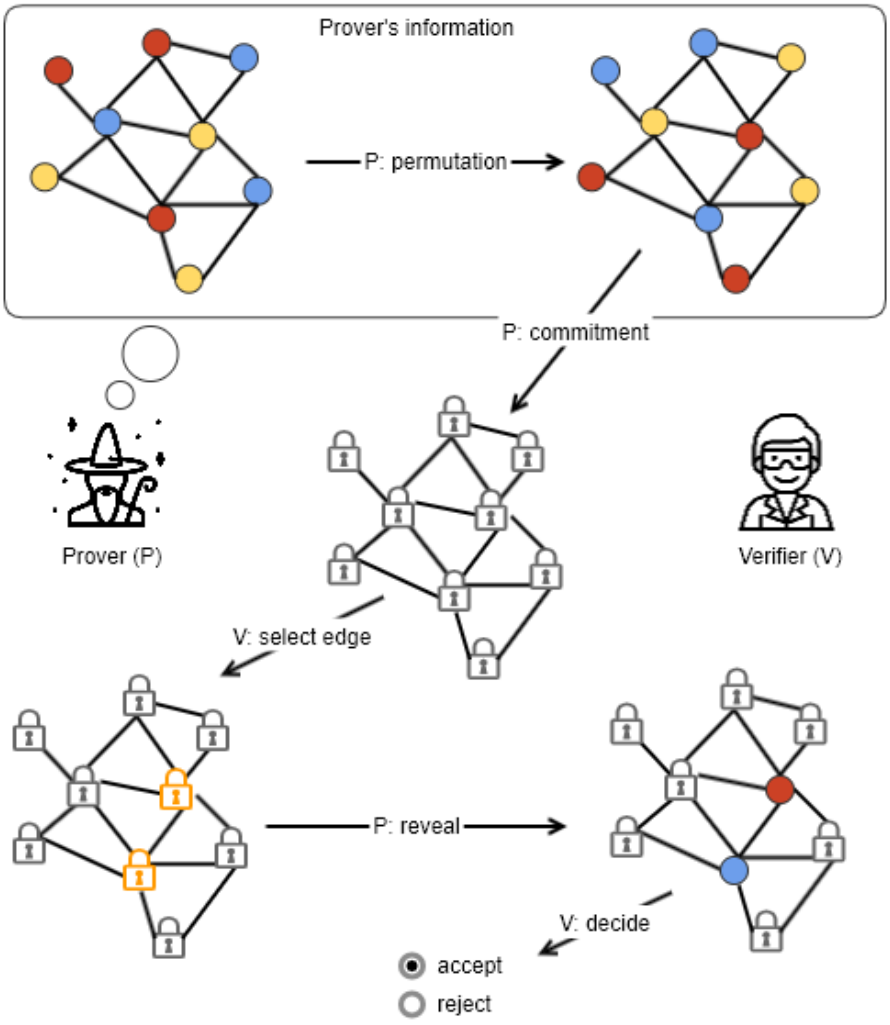

Figure 4.2: Illustration of the protocol of the graph 3-coloring zero-knowledge proof.

4.3 Zero-knowledge definitions and variants

Verifier's view

An alternative formulation for the zero-knowledge, which is more convenient in some occasions, replaces the output of the verifier V^* at the previous definition with the "view" (of information) that the verifier gains from the interaction with the prover. Let the $\text{View}^P_{V^*}(x)$ be a random variable that describes the content of the random tape of the V^* and the messages V^* receives from the interaction with P on a common input x. In the alternative formulation the probability ensemble $\{\text{View}^P_{V^*}(x)\}_{x \in L}$ replaces the output of

V^*.

Zero-knowledge condition

There are also different definitions of zero-knowledge, which are not equally powerful. The definition above is actually of a *computational zero-knowledge*. The most demanding is the definition for *perfect zero-knowledge*. It requires that for every verifier there exist a simulator which produces *identical* probability ensembles to what the verifier produces. *Statistical zero knowledge* requires that the above-mentioned probability ensembles are statistically close the each other: their statistical difference is negligible.

With perfect and statistical zero-knowledge even adversary with unbounded computational power cannot distinguish the real proof from the simulated proof. So these two types of zero-knowledge provide security even against unbounded adversary, whereas computational zero-knowledge protects from computationally bounded adversary. The latter property is easier to achieve and often sufficient. This difference in complexity is revealed also in the fact that statistical zero-knowledge proof systems cannot be constructed for all* languages in NP [50, 1].

Soundness condition

Also the soundness property has variants: there are notions of *statistical soundness* and *computational soundness*. The statistical soundness is the one we have used here so far. Statistical soundness holds against unbounded adversary, that in this case is the prover. So far prover's resources have not been limited. But if we require also the prover to be computationally bounded, to execute in probabilistic polynomial time, we get to the notion introduced by Brassard, Chaum and Crépeau [29]: the computational soundness. When it prevails, it is infeasible, that computationally bounded prover can cheat the verifier (to accept a false statement). With computationally sound protocols it is common to use a term *argument* instead of the term "proof" in the nomenclature. So when zero-knowledge is involved, we often see the term *zero-knowledge argument*. A notable difference between proof and argument systems is, that there exist statistical zero-knowledge argument systems for all languages in NP [29].

*These proof systems can be constructed to languages in complexity class AM $\cap$ coAM that is not believed to include NP.

Auxiliary input

An often used augmentation to the zero-knowledge proof model is the notion of *auxiliary input*. This captures the idea of ZK proof system not being an isolated process, but belonging to a larger system, whereof verifier might also get some a priori information. Let us denote this information as z. It might be for example, that information z relates to input x in ZK proof, and could help verifier to extract more information from prover, potentially destroying zero-knowledge property.

Definition 11 (Auxiliary input zero-knowledge) *[98],[67] An interactive proof system (P, V) for language L is* auxiliary input zero knowledge *if for every probabilistic polynomial time verifier V^* there exists a probabilistic polynomial time simulator M^* such that the following two probability ensembles are computationally indistinguishable:*

- $\{P(x), V^*(x, z)\}_{x;z\in D_1}$*, which is the output of verifier V^* after it interacts with the prover P on common input x and auxiliary input z*
- $\{M^*(x, z)\}_{x;z\in D_1}$*, which is the output of simulator M^* on input (x, z)*

where $D_1 = \{(x, z)|x \in L, z \in \{0, 1\}^*\}$

As was noted, the verifier's a priori information z is relevant for the examination of the zero-knowledge property. The prover's a priori information, which could be denoted as y, is not significant in that sense. Prover's auxiliary input is often used in problem settings to contain a "proof" of the membership $x \in L$. It enables prover's execution to be polynomial time, when prover's auxiliary input y is a *witness* for the membership of $x \in L$.

Protocol composition

Formally, the more general problem that the construction of the auxiliary input zero-knowledge aims to solve is called *sequential composition* (of zero-knowledge proofs). It means that the protocol is repeated sequentially. As was noted, the verifier might be able to gain some additional knowledge from subsequent executions of the protocol. It is still desirable that the zero-knowledge property would hold in sequential composition. It has been shown [65] that this is not the case for the "original" definition (10) of zero-knowledge, i.e is not closed under sequential composition. On the other hand, the auxiliary input zero-knowledge has been proven to be closed under sequential composition [67].

Other related setups are *parallel composition* and *concurrent composition*. In the parallel composition polynomially many instances of the protocol are invoked at the same time synchronously so, that they proceed at the same pace. In the concurrent composition the synchronization of parallel invocations is relaxed to be completely asynchronous or to follow a specific timing-model [44].

Parallel (or concurrent) composition would help to decrease the protocol's *round-complexity*, i.e., the number of rounds needed in the protocol. Unfortunately it is harder to maintain zero-knowledge property with these setups, compared to the sequential composition. Zero-knowledge, when based only on the existence of one-way functions, is not closed under parallel composition [65]. Under number-theoretical assumptions, namely, the existence of claw-free permutations, there exist protocols where zero-knowledge property holds in parallel composition [64] and in concurrent composition [60]. Goldreich and Kahan showed [64] how to make a parallel version of the zero-knowledge proof for graph 3-coloring (its sequential version was sketched above).

Proof of knowledge

Proof systems we have reviewed so far have dealt with whether the prover can convince the verifier that the statement $x \in L$ is valid. If NP-statement $x \in L$ is true, then there exists a witness w such that the validity of $x \in L$ can be verified from w in polynomial time. Note that there may be several such witnesses. However, if the prover can convince the verifier that $x \in L$ is valid, it does not necessarily mean that the prover "knows" any such witness.

The expression that "a machine knows something" may sound vague, but in the field of cryptography it has been formalized as a notion. It is said that a machine P *knows* w, if w can be "efficiently learned" from the P. This means that there exists a probabilistic polynomial time extractor E that by using P as a subroutine* can extract w from P. Proof system (P, V) is a *proof of knowledge* for language L if

$$V(P(x)) = \text{"accepts"} \Rightarrow P \text{ knows a witness } w \text{ for } x.$$

The proof of knowledge can be seen as a reinforcement for the property of computational soundness, with the prover then being, of course, polynomial time. More rigorous treatment is found from [9].

*This is often phrased as "using P as an oracle".

4.4 Non-interactive zero-knowledge proofs

There are scenarios, where the zero-knowledge property would be definitely useful, but the interaction becomes a burden. For instance with blockchain implementations and their distributed nature, pairwise communication between prover and verifier does not fit well in the picture. Anyone should be able to verify blockchain "statements" easily. The needs for non-interactivity already existed before blockchains. The notion of *non-interactive zero-knowledge* (NIZK) was introduced by Blum et al. [21]. In their proof system, the interaction consists of a single message from the prover to the verifier, making it non-interactive. As the interactivity is removed, the system needs some other properties in order to maintain zero-knowledge, and to be useful at the same time.

Oren proved [98], that if language L has one round (single message from the prover to the verifier) zero-knowledge interactive proof system, then $L \in BPP$. The cryptography, on the other hand, is largely based on problems that cannot be solved efficiently, only verifying might be efficient. A large part of cryptography is based on assumption, that $NP \nsubseteq BPP$. When language $L \in BPP$, it cannot contain any complexity-based cryptographic constructs, like hash functions or digital signatures. Other way to look at this is, that language $L \in BPP$ has *trivial* zero-knowledge proof system. Verifier can check by itself if $x \in L$.

The result of Oren applies for NIZK proof systems in the standard model. Therefore, NIZK proof systems needs stronger model. Two main examples of such models, where NIZK proof systems have been successfully developed, are *common reference string* (CRS) model and *random oracle* model.

Common reference string (CRS) model

Let us start from a zero-knowledge proof: the prover wants to convince the verifier that $x \in L$. At this time the prover is helpful and wants to save verifier's effort. The prover sends a transcript of a simulated proof, containing the interaction between prover and a simulated verifier. In the simulation, the prover obviously plays the simulated verifier's part according to the protocol, but special attention needs to be paid to the randomness of the simulated verifier.

First, the protocol of this zero-knowledge proof is a *public coin* protocol (described in Section 4.1). The prover plays the simulated verifier's part, so prover must also know the

coin tosses. Secondly, it is crucial that the true verifier can trust the randomness used by the simulated verifier. If the prover has all power over the randomness of the simulated verifier, then he can choose the seed of pseudorandom generator that enables the seemingly random behaviour of the simulated verifier. This destroys the soundness of the proof. The prover could choose the seed deliberately so, that the simulated verifier asks "randomly" that kind of questions from the prover which enable prover to produce a convincing proof of the membership of x in L while $x \notin L$.

If we want to obtain non-interactive zero-knowledge proofs through this paradigm of simulated proof, we need to solve how the randomness of the simulated verifier could be beyond the reach of the prover. For clarification, because the term “simulation“ is used also in other context with zero knowledge proofs, and not to confuse the verifier with the simulated verifier, we can use other interpretation for the latter one. Informally, the role of the simulated verifier could be described as a set of randomized challenges that are included in the proof.

Blum et al. [21] introduced the *common reference string* (CRS) model to solve this issue with trusted randomness. Common reference string is a shared string that both prover and verifier can read. This string is assumed to be sampled from some distribution D. In the case of [21] distribution D was discrete uniform distribution of $\{0, 1\}$. This means that the CRS is completely random. In the construction of the zero-knowledge proof using CRS, it is essential that the randomness of its randomized challenges (simulated proof paradigm) originates from the CRS.

A drawback with the CRS is, that it needs to be trusted. Verifier must be able to trust it as a source of randomness, beyond prover's intervention. Also the prover has something to fear, does the zero-knowledge property anymore hold if the credibility of the CRS is compromised. The generation of the CRS is a *trusted setup*. When comparing to interactive zero-knowledge proof systems in general, they do not necessitate such a setup phase where trust is required. This requirement of trust in the setup can be an annoyance, especially in scenarios like blockchain implementations, where decentralization is advocated. Previously the generation of the CRS has usually needed a trusted third party, but more distributed means have been developed, like in the form of a multi-party computation [11].

To approach the definition of the zero knowledge proof systems let us introduce following notations. Function $\nu(n)$ is a negligible function. Common reference string, that the prover and the verifier share, is denoted by σ. Let language $L_R \in NP$ be characterized by polynomial time recognizable relation R, where for each $(x, w) \in R$, w is a witness to the

statement "$x \in L_R$". The prover has the witness w as an auxiliary input. The proof that prover sends to verifier is therefore denoted as $P(x, w, \sigma)$, where only input w is hidden from verifier. The decision value of the verifier is either 1 (accepts) or 0 (rejects).

Definition 12 (Non-interactive proof systems) *[21], [48] A pair of Turing machines (P, V), where V is probabilistic polynomial time, is a* non-interactive proof system *for language L_R if the following two hold:*

- *Completeness:* $\forall(x, w) \in R : Pr(V(x, \sigma, P(x, w, \sigma)) = 1) > 1 - \nu(|x|)$
- *Soundness:* $\forall P', \forall x \notin L_R, \forall w' : Pr(V(x, \sigma, P'(x, w', \sigma)) = 1) < \nu(|x|)$

First constructions of NIZK proof systems using CRS [21] allowed the prover to prove exactly one statement with the same CRS. We now introduce a definition of zero-knowledge for this "bounded" case. The applicability of the CRS for proving multiple statements, while preserving the zero-knowledge property, is not guaranteed with this setup.

Definition 13 (Bounded non-interactive zero-knowledge) *[21, 48] A non-interactive proof system (P, V) for relation R is* bounded zero-knowledge *if there exists a probabilistic polynomial time simulator S, such that $\forall(x, w) \in R$ the two probability ensembles $\{(x, \sigma, P(x, w, \sigma))\}$ and $\{S(x)\}$ are computationally indistinguishable.*

Later, Feige, Lapidot and Shamir [48] showed how to construct a NIZK proof system, that enables polynomially many provers to prove polynomially many statements, using a single CRS. Using a notation from the article of Groth, Ostrovski and Sahai [77] we present the definition for this "multiple prover" case. By this definition zero-knowledge holds also against adaptive distinguisher*, denoted here as adversary $\mathcal{A}$, where $\mathcal{A}$ may select the statement-witness pair (x, w) after seeing the CRS. The adversary $\mathcal{A}$ is expected to be *non-uniform* polynomial time, which means that $\mathcal{A}$ can be represented by a family of polynomial size Boolean circuits. The CRS generation algorithm K is included to the definition.

Definition 14 (Multi-theorem adaptive non-interactive zero-knowledge) *[48, 77] Let $S = (S_1, S_2)$ be a probabilistic polynomial time simulator such that S_1 generates a simulated CRS along with a trapdoor information τ that enables the second simulator S_2 to*

*The distinguisher was referred in the definition 9 (computational indistinguishability).

create a simulated proof for statement x without the witness w. A non-interactive proof system (K, P, V) is adaptive multi-theorem zero-knowledge *for language L_R if there exists a simulator $S = (S_1, S_2)$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$:*

$$Pr[\sigma \leftarrow K(1^k); \quad (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(x, w, \sigma) \quad : \mathcal{A}(\pi) = 1] -$$
$$Pr[(\sigma, \tau) \leftarrow S_1(1^k); \quad (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow S_2(x, \tau, \sigma) \quad : \mathcal{A}(\pi) = 1] < \nu(k),$$

where P and S_2 are expected to output $failure$ if $(x, w) \notin R$.

In some setups the CRS is explicitly divided to two portions, *proving key* and *verification key*, where the first portion is needed by the prover and the second portion by the verifier. This is relevant especially in the case when the verification key is made very small compared to the proving key, thus reducing verification complexity.

In both cases ([21], [48]) the constructed NIZK proof systems were proven to exist for all languages in NP.

Fiat-Shamir transform

The work of Fiat and Shamir [49] was one of early applications of random oracle, before the explicit random oracle methodology was described [10]. They first presented an interactive identification protocol between prover and verifier, which they then transformed to a non-interactive protocol*. The interactive protocol consists of three moves and is a public coin protocol. Let us denote the statement to be proved as x. The prover sends a message m to the verifier, whose role is then to pick a challenge c randomly from challenge space, and send it to the prover. Finally the prover sends a response r to the verifier. Verifier's decision can be denoted as a function $V(x, m, c, r)$ that outputs either 1 (accept) or 0 (reject).

In the transformation (illustrated in Figure 4.3), verifier's role of providing a challenge is replaced by a cryptographic hash function H. The random challenge to prover is $H(m)$, the hash value of the message m that prover would send to verifier in interactive protocol. Prover's response is now (m, r), thus providing to the verifier both the final message r and the intermediate value m so that the hash function usage is transparent to the verifier. Verifier's decision function is denoted now as $V(x, m, H(x, m), r)$. This way it is possible to transform a three-move protocol into a non-interactive one-move protocol.

*The interactive protocol was identification protocol, that was transformed to digital signature protocol. This has been since an important use case for Fiat-Shamir transform.

Bellare and Rogaway [10] formalized the transformation of Fiat and Shamir under random oracle model, where hash function is replaced by an idealized version of it, a random function. They also generalized this transformation to three-move public coin zero-knowledge arguments, that are transformed to non-interactive zero-knowledge arguments. Recall the difference between proof and argument, stated in Section 4.3, so Fiat-Shamir transform ensures only computational soundness.

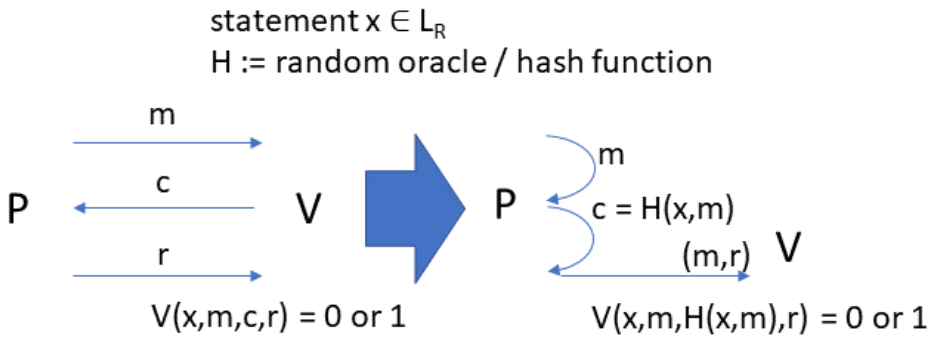

Figure 4.3: Fiat-Shamir transform, based on the definition from [42]. On the left side is interactive protocol, and on the right side is non-interactive protocol that is yielded with Fiat-Shamir transform. In the context of zero-knowledge proofs the prover's message m is typically a commitment.

The Fiat-Shamir transform is an efficient way to construct NIZK arguments. The starting point, the existence of underlying three-move interactive protocol, also referred as Σ-*protocol*, is of course a requirement by itself. Informally can be said, that in Σ-protocol the challenge space must be big enough, so that the verifier can be convinced of the prover's response for just one challenge, i.e., probability that prover succeeds in cheating in a single challenge is negligible.

The pitfalls found in the random oracle model were discussed in Section 2.2. Similarly, a specific scheme that is secure in random oracle model and is rendered insecure with any real hash function, has also been found relating to Fiat-Shamir transform [70]. Despite these negative theoretical results, Fiat-Shamir transform has been used in practical implementations to obtain NIZK arguments, like in e-voting [74].

4.5 Efficient NIZK proofs

After the theoretical foundation for the non-interactive zero-knowledge proofs has been built, a substantial portion of the related research has been focused to its practical feasibility. Related aspects include the size of the proof (or argument), prover's execution time, verifier's execution time and the size of the CRS. Smaller proof or CRS size decreases communication costs and it typically also means that verifying the proof is computationally easier. This section handles exclusively NIZK with CRS model: the Fiat-Shamir transform is in its simplicity convenient for many practical implementations at the cost of controversial security.

It is a goal worth pursuing to find a NIZK proof system that can be used for all NP statements (for any statement of any language in NP). This kind of proof system is constructed for some NP-complete language. Any statement in NP can be transformed to that chosen language by polynomial time (and polynomial space) reduction. After that the NIZK proof system can be used for that transformed statement.

Various NP-complete problems have been used in NIZK proof system construction, like 3-satisfiability problem (3-SAT) [20], Hamiltonian cycle problem (HC) [48] and circuit satisfiability problem (CSAT) [38]. Damgård noted the benefits of constructing the proof system for CSAT: the reduction process from the given statement to an CSAT instance is more straightforward compared to 3-SAT and HC, and also the resulting instances are smaller [38]. The reduction of an NP statement s to CSAT means finding a circuit C_s that is satisfiable if and only if s is true.

NIZK proof system, like any cryptographic scheme, is based on some computational assumption(s). As was noted in Section 2.2, in general, a weaker assumption yields a better security than a stronger assumption, at least if the scheme is viewed like a black box and only this assumption aspect is scrutinized. On the other hand, it may be harder to find a security proof that is based on weaker assumption, or the resulting scheme may be more complex. It is an open question whether NIZK proof system for NP can only be based on existence of one-way functions. Various other computational assumptions have been used. Notable examples include quadratic residuocity [20, 38] that is a specific number-theoretic assumption, and existence of trapdoor permutation [48].

In 2006 Groth, Ostrovski and Sahai [78] introduced a NIZK proof system for CSAT, that was significantly more efficient compared to previous systems. With CSAT problem the NIZK proof and CRS size can be expressed in terms of the circuit size $|C|$ and the inevitable

security parameter k. In previous NIZK proof systems for CSAT both the CRS size and the proof size* were $\mathcal{O}(|C|k^2)$ at its best [85, 28]. In the new proof system [78] the CRS size was $\mathcal{O}(|k|)$ and the proof size $\mathcal{O}(|C|k)$. The Boneh-Goh-Nissim cryptosystem [24], based on pairing-based cryptography, played a pivotal role in the efficiency gains.

Even better results have been gained by utilizing fully homomorphic encryption. When Gentry introduced the first FHE scheme, he also outlined NIZK proof system that uses FHE [56]:

NIZK proof for CSAT using FHE

- Common input: circuit C. Prover's input: witness $w = w_1, ..., w_n$ that is a satisfying assignment for circuit C. Prover's statement: there exists witness $w^{'}$ such that $C(w_1^{'}, ..., w_n^{'}) = 1$, i.e. C is satisfiable.
- Using the fully homomorphic encryption scheme $\mathcal{E} = (Gen_{\mathcal{E}}, Enc_{\mathcal{E}}, Dec_{\mathcal{E}}, Eval_{\mathcal{E}})$ prover generates:
  - A public key pk (a key pair (pk, sk)).
  - The input (witness) ciphertexts $\{\psi_i \leftarrow Enc_{\mathcal{E}}(pk, w_i)\}$.
  - The output ciphertext $\psi \leftarrow Eval_{\mathcal{E}}(pk, C, \psi_1, ..., \psi_n)$.
- Using a NIZK scheme prover constructs following NIZK proofs†:
  - The public key pk is a valid public key
  - Each input ciphertext ψ_i is a ciphertext of 0 or 1.
  - The output ciphertext ψ is a ciphertext of 1
- The verifier verifies NIZK proofs and confirms that the output ciphertext ψ evaluates from the input ciphertexts: $\psi = Eval_{\mathcal{E}}(pk, C, \psi_1, ..., \psi_n)$. The properties of the encryption scheme ensures that: The output ciphertext ψ is a ciphertext of 1 $\Rightarrow$ the output of the circuit with plaintext input is 1: $C(w_1, ..., w_n) = 1$.

*As represented in [77], original articles [85, 28] have more detailed description.

†Describing these NIZK proofs would require a walkthrough of subtle details of the (FHE) encryption scheme, which is omitted. We note, however, that similar NIZK proofs can also be done with non-FHE scheme: example of a NIZK proof for the statement that a commitment contains 0 or 1 is found in [78]. As usual, the random bits needed to construct the NIZK proofs are in the CRS.

Gentry's NIZK proof construct provides proofs of size $\mathcal{O}(|w|poly(k))$, where $|w|$ is the witness size and k is the security parameter. Subsequently the size of NIZK proof for CSAT with FHE has been reduced to $\mathcal{O}(|w| + poly(k))$ [58], so the proof size is proportional to witness. According to current knowledge [63, 68], the proof size cannot be substantially smaller than the witness, so in essence the theoretical limit has been reached in the asymptotic efficiency. But when the soundness property is relaxed to computational soundness, and we are therefore limited to argument systems, substantial gains in efficiency can be reached, as is shown in the next section.

4.6 Succinct NIZK arguments

Development of NIZK argument systems has lead to *succinct* arguments, meaning that the argument size is sublinear with respect to the input. The following paradigm [23] has proven to be successful in the construction of succinct argument systems, and also cryptographic proof systems in general:

1. First construct a proof system, referred as *information theoretic proof system*, that provides security even against computationally unbounded parties, because its security is derived from information theory. This proof system usually falls into a category of *probabilistically checkable proofs* (PCP), which are explained later in this section.

2. Next use cryptographic tools, referred as *cryptographic compiler*, to bind the system to respect desired properties. Examples are the zero-knowledge property and non-interactivity. The cost is typically restriction to security against computationally bounded prover instead of an unbounded one. This restriction leads to an argument system. Examples of such "compilers" are random oracle, *extractable collision-resistant hash function* (ECRH) [16] and CRS model.

A *SNARG* means a *succinct non-interactive argument*, and correspondingly *SNARK* is a *succinct non-interactive argument of knowledge*. So the SNARK is a SNARG with proof of knowledge property. If also the zero-knowledge property prevails, we get to the notations *zk-SNARG* and *zk-SNARK*.

Next we introduce the notions of PCP and its variant known as *linear PCP*, and review some SNARG/SNARK constructions that are done with these models.

Probabilistically checkable proofs

The notion of *probabilistically checkable proofs* (PCP) [3] defines a hierarchy for complexity classes. Following the PCP definition, languages are classified based on how efficiently a verifier can check membership proofs for them. In PCP the probabilistic polynomial time verifier has a random access to the proof, and it can query from the proof a certain, small amount of bits at a time. The explicit notion of prover is not needed, but it can be thought of as an oracle that responds to verifier's queries by providing the bits from the position of the proof that verifier asks. A language L is in the complexity class $PCP[r, q]$ if there exists a verifier V that uses r random bits, reads q bits from the proof and the following two properties hold:

- if $x \in L$, then there exists a proof π such that $Pr[V \text{ accepts } (x, \pi)] = 1$;
- if $x \notin L$, then for every proof π $Pr[V \text{ accepts } (x, \pi)] \leq \frac{1}{2}$.

Let n be the size of the input x. It follows from the definition that $NP = PCP[0, \mathcal{O}(poly(n))]$. A deterministic polynomial time verifier does not require any random bits, and it queries at most a polynomial amount of bits from the proof. The *PCP theorem* [2] gives a new characterization for the *NP*:

$$NP = PCP[\mathcal{O}(\log n), \mathcal{O}(1)].$$

This means that for every language in *NP* there is a verifier that uses at most $\mathcal{O}(\log n)$ random bits and reads only $\mathcal{O}(1)$ (constant amount) of bits from the proof. The soundness error $\epsilon = \frac{1}{2}$ as described above follows the original definition, but it can also be added to the PCP definition as an extra parameter: $\epsilon < 1$.

Naturally a "classical" type of proof is not suitable for the PCP proof system, because the verifier could have to go through almost the whole proof to find an erroneous place that falsifies it. The PCP introduces a new kind of a proof that is more suitable for probabilistic checking. In the PCP proof system a proof of a false statement is guaranteed to have so many errors that the verifier is able to find one with high probability by reading only a small part of a proof.

Over time various PCP proof constructions have been described. A starting point is some specific NP-complete language, so the NP-reduction is an overhead that is usually required. The PCP proof construction has an analogy to *error correcting codes* (ECC) that are used to detect errors in data transmission over unreliable channels. When the message

is encoded to ECC format, it contains redundant information in a way, that enables the receiver detect if the message is corrupted. In the PCP proof of Dinur [41] an integral part is a "gap amplification", that is iteratively used to double the "erroneous places" of the proof with only linear increase to the proof size.

Already at an early stage, PCPs were introduced in cryptographic applications. Using PCPs, Kilian [84] constructed an interactive argument system where the communication complexity between the prover and the verifier is polylogarithmic, i.e., succinct. Micali [92] showed how to make this kind of argument system non-interactive in the random oracle model, by using a hash function along with the PCP string, to simulate verifier's PCP queries. Bitansky et. al replaced the random oracle with ECHR, that is a collision resistant hash function with an additional *extractability* property [16].

Linear PCP

Initially PCP has been regarded as a string to which a verifier has oracle access. Gennaro et. al [55] noted that while the regular, "general-purpose" PCP enables the creation of SNARGs, some other model could fit better for cryptographic purposes. They introduced the models of *quadratic span programs* (QSPs) and *quadratic arithmetic programs* (QAP). These models are linear algebraic constructions that are converted from input circuits. QSP is converted from a boolean circuit, and QAP from an arithmetic circuit. The reduction from the specific problem to the input format, a boolean or an arithmetic circuit is still needed to be done, so the user of the scheme can decide which one of these suits better. Their scheme [55], also known as GGPR13, yields arguments of size $\mathcal{O}(1)$ with $\mathcal{O}(1)$ verifier's complexity. Prover's complexity is $\mathcal{O}(n \log n)$, and the CRS size is $\mathcal{O}(n)$, where n is the number of gates in circuit.

We omit the details of these conversions but basically what is obtained from the QSP or QAP, is a set of polynomials. The size and degree of each polynomial is roughly the number of the gates in the circuit. We now review the QAP in more detail and show how it can be used to obtain zk-SNARKs. Before that we need to formalize arithmetic circuit and the satisfiability therein.

An *arithmetic circuit* is a directed, acyclic graph that consists of *arithmetic gates*. The output of the gate is a result of an arithmetic operation to input value or values. Figure 2.1 of a boolean circuit can be used to familiarize oneself with circuit structure in general. The gates considered here are *bilinear*: the operation can be addition, multiplication or

multiplication by scalar. Arithmetic circuit C with n inputs and m outputs is therefore a function $f : \mathbb{F}^n \to \mathbb{F}^m$, where $\mathbb{F}$ is a field, typically a prime field that consists of integers mod p, where p is a prime. By definition, each output of the circuit represents a polynomial of the input values.

The satisfiability of an arithmetic circuit is defined analogously to the boolean case. Let n, h, l respectively denote the statement, witness and output sizes. The input is divided into two parts, the statement and the witness. The *arithmetic circuit satisfiability problem* [12] of a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ is defined by the relation $R_C = \{(x, w) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, w) = 0^l\}$. The corresponding language is $L_C = \{x \in \mathbb{F}^n : \exists w \in \mathbb{F}^h, C(x, w) = 0^l\}$.

Next we define quadratic arithmetic program and its satisfiability, which were originally introduced in [55], following the notation of [12]. First, let us denote by $\mathbb{F}[z]$ the ring of univariate polynomials over field $\mathbb{F}$ and by $\mathbb{F}^{\leq d}[z]$ the subring of polynomials of degree less than or equal to d.

Definition 15 (Quadratic arithmetic program [12]) *A* quadratic arithmetic program *(QAP) of size m and degree d over $\mathbb{F}$ is a tuple (A, B, C, Z) where*

- *$A = (A_0, A_1, ..., A_m)$ with $A_i \in \mathbb{F}^{\leq d-1}[z]$,*
- *$B = (B_0, B_1, ..., B_m)$ with $B_i \in \mathbb{F}^{\leq d-1}[z]$,*
- *$C = (C_0, C_1, ..., C_m)$ with $C_i \in \mathbb{F}^{\leq d-1}[z]$ and*
- *$Z \in \mathbb{F}[z]$ has degree exactly d.*

Definition 16 (QAP satisfiability problem [12]) *The* satisfiability problem *of a QAP (A, B, C, Z) of size m is defined by the relation $R_{A,B,C,Z}$ of pairs (x, s) such that*

- *$x \in \mathbb{F}^n$, $s \in \mathbb{F}^m$ and $n \leq m$;*
- *$x_i = s_i$ for $i \in [n]$ (i.e, s extends x); and*
- *the polynomial $Z(z)$ divides the following polynomial*

$$(A_0(z) + \sum_{i=1}^{m} s_i A_i(z)) \cdot (B_0(z) + \sum_{i=1}^{m} s_i B_i(z)) - (C_0(z) + \sum_{i=1}^{m} s_i C_i(z)).$$

The vector s can be called as a "QAP witness".

Reduction from arithmetic circuit satisfiability to QAP satisfiability can be done efficiently, as was shown by Gennaro et. al [55]. If the number of wires in circuit C is α and the number of gates is β, then there exists a polynomial time algorithm QAPinst that produces QAP with size $m = \alpha$ and degree $d = \beta + 1$ [12]. Next we outline the zk-SNARK construction from QAP very informally; missing details can be found from [55, 12].

Besides the QAPinst, there exists also a polynomial time algorithm QAPwit [12] that enables the prover to construct the QAP witness s. Let us disregard the zero-knowledge property for a while, and assume that the prover just wants to prove that they know a witness w for a statement $x \in L_C$. If the prover just sends the QAP witness s to the verifier, the proof size is proportional to the circuit size that may be thousands of gates. However, due to the properties of the underlying mathematics and the zk-SNARK construction scheme in question [55], the verifier does not need so much information to verify the divisibility of polynomials described in the definition 16. It suffices that verifier evaluates certain constraints in only a few different points in polynomials, so the proof needs to have data only for these few evaluation points. The proof is thus of a constant size, i.e., succinct.

If the prover would know what the evaluation points are, they could fabricate the proof so that the constraint checks are passed, although the QAP-satisfiability of the polynomials is not fulfilled. Crucially, the zk-SNARK scheme provides a way to evaluate these polynomials "blindly", so that neither the prover nor the verifier know the actual evaluation points. An additively homomorphic encryption scheme is used to enable constraint checking with encrypted evaluation points. In the CRS generation the plaintext values of these evaluation points were sampled randomly, and their encryptions were included in the CRS. Afterwards the plaintext values are not needed, and because they could be used to make forged proofs, destroying them is recommended.

Zero-knowledge property is achieved so that the prover samples random values $\delta_A, \delta_B, \delta_C \in \mathbb{F}$, and while constructing the proof incorporates them to the QAP polynomials A,B,C so that the divisibility still holds. This way the verifier does not learn anything else than that the divisibility holds. In fact, this scheme is statistically zero-knowledge, because the encrypted elements in the proof are statistically indistinguishable from other encrypted elements that satisfy the constraints. While not outlined here, the scheme provides also proof of knowledge.

PCP is categorized as *linear PCP* [17] if the proof oracle that answers to verifier's queries is a linear function $\pi : \mathbb{F}^n \to \mathbb{F}$. The previously mentioned constraint checks performed to

ensure the divisibility of polynomials in the QAP structure (and correspondingly also in the QSP) can thus be seen as linear PCP queries [17].

Practical implementations

The zk-SNARK scheme introduced by Gennaro et. al [55], henceforth called GGPR13, has been a basis for several implementations. Pinocchio [99] was among the first such implementations. Pinocchio is a program that can build QSP or QAP based zk-SNARK proof systems from boolean circuits or arithmetic circuits, respectively. Furthermore, Pinocchio provides a compiler to generate arithmetic circuits from input programs that are written in certain subset of C language. Together this forms an end-to-end solution from describing NP statements in a high-level programming language to a zk-SNARK proof system for these statements.

A zk-SNARK proof system developed by Ben-Sasson et. al [12] (BCTV14) is based on the schemes in GGPR13 and Pinocchio. It has some efficiency improvements, and it provides more flexibility for defining the input C programs for circuit compiler than Pinocchio. It also was the initial proof system for the cryptocurrency Zcash that is reviewed in the following section.

In recent years numerous other zk-SNARK implementations have been created*, and many of them are based on other constructions than QAP. Some of those non-QAP based schemes are mentioned in Section 5.4.

*https://zkp.science/

5 ZKP's utilization with blockchain technologies

In this section we review three examples of blockchain applications that utilize zero-knowledge proofs to improve privacy. The first two, Zcash and Ethereum use zk-SNARKs, and the third, Monero, uses NIZK proofs based on the Fiat-Shamir transform. The last subsection summarizes the findings related to these examples. In particular, privacy, security and viability aspects are reviewed.

5.1 Zcash

Zcash is a cryptocurrency that utilizes zk-SNARK based zero-knowledge protocol to achieve privacy. Zcash software is open source and is built on the Bitcoin codebase. These two cryptocurrencies have similarities in mechanisms and also in details, like the same total supply of units (21 million). Zcash has a proof-of-work consensus mechanism like Bitcoin, but it uses different PoW algorithm called *Equihash* [15]. Zcash has been created by a privately held company Electric Coin Co. that also has the main responsibility of Zcash's further development. The initial release of Zcash was in 2016 under the version name *Sprout*. The current version is called *Sapling*, and substantial changes have been done to the Zcash protocol since the initial release.

Zcash has two types of addresses, transparent addresses (t-addresses) and *shielded* addresses (z-addresses). Transactions between t-addresses, called as transparent transactions, work similarly like in Bitcoin. Input and output addresses and transaction values are publicly visible. Transactions between z-addresses, called as shielded transactions, are encrypted, so the input and output addresses and transaction values are not disclosed. Shielded transaction contains a zero-knowledge argument, zk-SNARK, of its validity. It is also possible to create a transaction from t-address to z-address, known as shielding, and from z-address to t-address, known as deshielding. Shielded parts of shielding and deshielding transactions are covered by zk-SNARKs. The set of the shielded addresses is called as *shielded pool*.

Since the shielded transactions are a characteristic of Zcash, the existence of transparency

requires justification. Zcash site mentions [114] that transparent addresses enable easier adoption of the currency, and let users to make themselves a choice of their privacy level. As is discussed in Section 5.4, the existence of transparent addresses has been practically imperative to support compatibility with other cryptocurrency infrastructure.

Zerocash

Zcash is based on the *Zerocash* cryptographic currency protocol [106]. Over the years the Zcash protocol [80] has somewhat changed comparing to the original Zerocash. There were already differences during the release of Zcash, but the Zerocash still constitutes a major theoretical foundation for the current Zcash. One contribution of Zerocash article [106] is the *decentralized anonymous payment scheme* (DAP) that is meant to be a generic, privacy preserving primitive that could be used as a foundation for decentralized currencies. Zerocash is an implementation of DAP, and thus to some extent Zcash also implements it.

DAP is modelled as an extension to a regular blockchain based currency, that is referred to as *Basecoin*. So the blockchain and network mechanisms of the Basecoin are assumed to be given. One example of a Basecoin is Bitcoin, that is used as the basis for Zerocash.

Definition 17 *[106] A* decentralized anonymous payment scheme *(DAP) is a tuple of polynomial time algorithms (Setup, CreateAddress, Mint, Pour, VerifyTransaction, Receive) that are described as follows:*

- *The algorithm* Setup*, having as input a security parameter λ, generates public parameters pp (the CRS). The setup is done only once, by a trusted party.*
- *The algorithm* CreateAddress *generates a new address-key pair $(addr_{pk}, addr_{sk})$. The key $addr_{pk}$ is a public address where coins can be sent to. Hereafter these coins can be spent with the corresponding private key $addr_{sk}$.*
- *The algorithm* Mint *on given value v and address $addr_{pk}$ generates* coin *c that has a unique serial number sn and previously mentioned address and value. A mint transaction $mint_{tx}$ that contains a coin commitment $cm(c)$ is created. This coin commitment hides the address, value and serial number of the coin.*
- *The algorithm* Pour *transfers value from input coins into new output coins, input coins are set consumed. From input coins the algorithm also needs the private keys*

of addresses. Output coins are described as (value, output address) pairs. A pour transaction $pour_{tx}$ is created, that contains serial numbers of input coins which are therefore considered consumed. The serial number is necessary to prevent double spending. Pour transaction contains also coin commitments of output coins.

- *The algorithm* VerifyTransaction *checks the validity of transaction ($mint_{tx}$ or $pour_{tx}$).*
- *The algorithm* Receive *can be used by address owner. On given address key pair $(addr_{pk}, addr_{sk})$ algorithm Receive outputs unspent coins for that address.*

DAP scheme should fulfill *completeness*, which means that unspent coins can be spent. Security related properties are *ledger indistinguishability*, *transaction non-malleability*, and *balance*. Ledger indistinguishability means that only the information that is intended as public can be learned from the blockchain. Adversary's inability to alter the transaction before it is stored in the blockchain is referred as non-malleability. Balance property assures that the amount of unspent coins cannot be exceeded by spending.

The concrete implementation of DAP scheme, Zerocash, deploys standard cryptographic machinery from public key cryptography and hash functions, tools that are already widely used in Bitcoin. In this perspective, deployment of zk-SNARKs is the most distinctive new feature. The place where zk-SNARKs are used is in the Pour transaction where coins are transferred between addresses.

zk-SNARK generation in Zerocash

zk-SNARK, the construction of which is described below, is intended to be a zero-knowledge proof of the validity of a Pour transaction. The succinctness of zk-SNARK is essential because it is included in the Pour transaction, replacing its otherwise public information of sender, receiver and transaction value. Let us start the construction of zk-SNARK from the NP statement. In the case of Pour transaction $pour_{tx}$, its NP statement $pour_x$ is the visible content of transaction, that includes the serial numbers of old coins, and coin commitments of new coins. Let us denote the a user who creates transaction $pour_{tx}$ as prover. The prover must attach to $pour_{tx}$ a proof, zk-SNARK, that its statement $pour_x$ belongs to a language of valid transactions.

The witness w of statement $pour_x$ consists of prover's private information regarding the public content of the transaction. It should be noted that the witness is the part of the transaction data that is to be replaced by the zk-SNARK. Witness w contains all the

information of old and new coins in transaction, and the secret keys of old coins' addresses. The witness w is valid for a statement $pour_x$ if the following conditions are met*: 1) coin commitments of old coins are in the blockchain, 2) secret keys of old addresses match the corresponding public addresses, 3) sum of old coin values is equal to the sum of new ones.

Zerocash uses the BCTV14 system [12], that is a quadratic arithmetic programming implementation of zk-SNARK proof system. Therefore an arithmetic circuit is needed to be "etched" for $pour_x$. Actually, only one circuit C_{pour} is needed to be constructed in the system's lifetime. This circuit can validate any statement $pour_x$. The authors of Zerocash noticed, that the verifying of SHA-256 hash function output dominates the computational work in $pour_x$ statement verification; it is relatively expensive and it is done many times in one statement's verification. They handcrafted an efficient circuit for SHA-256, that has only 27904 gates. Using a programmatic circuit generator like Pinocchio [99] would have yielded at least twice as large circuit. The overall circuit C_{pour} was obtained by combining various subcircuits; circuits of SHA-256 and some others. The size of C_{pour} is about 4 million gates, more than 99% of which are dedicated to SHA-256 verifications†. With the circuit C_{pour}, zk-SNARKs are created using the BCTV14 system.

Last we present some metrics of the difficulty of creating and verifying zk-SNARKs of C_{pour} in Zerocash. With an ordinary laptop machine in 2014, proof generation for C_{pour} took about 3 minutes, and its verification time with VerifyTransaction algorithm was 8,5 milliseconds [12]. The proof size is constant, 288 B. This proof, zk-SNARK, that is included to $pour_{tx}$ transaction is lightweight enough to be added to blockchain, and its verification does not cause significant overhead for the miners. The work of the prover is more heavyweight, and as is discussed later, it has probably hindered the adoptance of zk-SNARK related shielded transactions in Zcash. The one time job of generating CRS took about 8 minutes and the resulted CRS size was about 900 MB. In the underlying zk-SNARK scheme [12] the CRS is divided into two parts: *proving key* that is needed by prover, and *verification key* that is needed in verification. From the total size of 900 MB the verification key was only 749 B.

*In more detailed technical level there are few other conditions [12].

†SHA-256 is represented in the circuit 146 times. Both old coin commitments are retrieved from Merkle tree data structure in which these commitments are stored. Each Merkle tree layer needs a SHA-256 invocation and the number of layers is 64 in the Zerocash implementation. Arithmetic circuit does not have loops, so for each SHA-256 invocation, its subcircuit must be replicated in the total C_{pour} circuit.

Trusted setup

The issues relating to trusted setup of CRS were not handled in the Zerocash paper [106]. Zcash, a cryptocurrency used in production, has had to resolve these issues. A key question is whether the CRS can be trusted. Loosely speaking, the party that generated the CRS may possess the related trapdoor information that it can use for abuses in the system. The definition of non-interactive zero-knowledge (14) necessitates an existence of such a trapdoor. As was shown by Bellare, Fuchsbauer and Scafuro [8], this trapdoor information could be exploited by adversary in various ways, depending of the system. At minimum either the soundness or the zero-knowledge property is lost in such attack.

In the Zcash's initial release, the CRS was created by following a multi-party protocol [26], that ensures that even if only one party is acting honestly, the resulting CRS is secure. Acting honestly here means that the party destroys the trapdoor information, also referred to as "toxic waste", they obtain when participating in the protocol, or at least does not combine this data with other parties. The execution of the multi-party protocol, by six participants, was elaborate and comprehensively documented ceremony. Regardless, if the security of the CRS were compromised, in the case of Zcash's zk-SNARK-system, the soundness would be lost, but not the zero-knowledge property. This means that the adversary could forge zero-knowledge proofs of shielded transactions, and so could counterfeit money without anyone noticing, but then again the privacy of the system's users would not be endangered.

zk-SNARK solutions in Zcash

We note that there are structural differences between Zerocash and Zcash. Unlike Zerocash, Zcash has no separate transaction types, but these operations are embedded in the original Bitcoin transaction as extra data. In Zcash the Pour operation is split to two different operations that both employ zk-SNARKs, Spend and Output transfers. Nevertheless, the zk-SNARK generation described above for Zerocash is essentially valid for Zcash as well.

Originally, in the Zcash Sprout release, the proof system for zk-SNARK was BCTV14 [12], as in Zerocash. In 2018, with Zcash Sapling release, the proof system was switched to Groth16 [75]. Like BCTV14, Groth16 is a QAP-based system, and its improvements include e.g. smaller proof size of 192 bytes. Another change in the Sapling upgrade was the partial replacement of the SHA-256 hash function with other solutions that can be

represented by much smaller arithmetic circuits. Consequently, proving has become easier. The proof generation time and its memory requirements have been reduced from ∼40 seconds/3 GB to ∼2 seconds/40 MB according to Zcash site*. This has made shielded payments, i.e., operations that require proof generation easier, thus facilitating better support for third-party applications and mobile devices.

In 2018, it was revealed [53] that the BCTV14 proof system has a severe security flaw. The CRS exposed some reduntant elements that the adversary could utilize in an attack. Basically, attacker could create a proof of knowledge for any statement, given a valid proof for some statement. When the vulnerability was revealed, Zcash was already using Groth16 proof system. Zcash representatives stated their belief that no one else had been aware of the vulnerability. In the report [53] it was stated that the vulnerability was noticed during reviewing a proof of security for BCTV14 [25], and that the proof itself was found to be erroneous.

5.2 Ethereum

Blockhain has been used as a foundation for cryptocurrencies. After the Bitcoin introduced the concept of blockchain, it was quickly found out that also other things, in addition to cryptocurrencies, could be decentralized by blockchain. One of the early examples was Namecoin [81] whose purpose was e.g. act as a decentralized domain name registry (DNS). Namecoin functions as a cryptocurrency, but it also supports storing of records, that are essentially key-value pairs, in blockchain. Purchasing of record costs some namecoins, these are included in the record creating transaction. Record owner can modify record, or give its ownership to someone else by making a transaction. The protocol maintains the uniqueness of stored keys.

Namecoin has similar proof-of-work based consensus mechanism like Bitcoin, so it needs a reasonable community support, i.e., amount of miners to retain its security. If some party has a majority of mining power, then decentralization can be seen as lost. A term *altcoin* refers to "alternative cryptocurrency to Bitcoin". Creating an altcoin is relatively easy, but the harder part is to bootstrap a community support for it. Therefore, it does not seem reasonable to create a new altcoin for each narrow use case or application.

Ethereum† is a project that tries to tackle this issue. The functionality of the Namecoin

*https://z.cash/upgrade/sapling/

†https://ethereum.org/

described above could be implemented in Ethereum. But instead of hard-coding Namecoin's application logic to the Ethereum protocol, Namecoin could be implemented as a programmable smart contract. Ethereum has its own Turing-complete programming language, a bytecode language referred to as "Ethereum virtual machine code" (EVM code) [111]. The EVM is a stack based virtual machine, which means that is does not use registers.

Smart contracts

A *smart contract* is a code snippet that is described in EVM language and stored in Ethereum blockchain. Smart contract is created by a transaction, and it can be referred to in subsequent transactions. Smart contract has a state that is stored in blockchain, and it defines a set of public methods with their arguments. Calling a public method of smart contract in transaction changes the state of this smart contract according to its program logic. Contract can also call a public method of another contract. For writing smart contracts, Ethereum provides multiple high-level programming languages, that resemble some general use programming languages like JavaScript or Python. When creating a transaction, Ethereum client then compiles the smart contract into bytecode. A *decentralized application* (dApp) is an entire application that provides a front end for its users, and whose back end consists of smart contracts in blockchain.

Each node that validates Ethereum transactions runs the code of the smart contract defined in transaction. If smart contract is computationally expensive, or even has infinite loops, troubles may follow. Ethereum prevents these problems with a mechanism that revolves around the concept of *gas*. Each basic operation in smart contract costs a specific amount of gas that reflects its complexity. A smart contract is created by sending a transaction where contract's bytecode is defined. The sender of transaction must define the gas price in the Ethereum currency *ether* that they pay for one gas unit, and also the gas limit that the contract computation is allowed to cost at maximum. If gas limit is hit, like in the case of infinite loop, the validator of the transaction halts computation of the contract. In this case the sender of the transaction pays ethers according to the gas limit, otherwise they pay of the spent gas.

Privacy in Ethereum

Ethereum originates to year 2013 when Vitalik Buterin published Ethereum's white paper [32]. The initial release of Ethereum was in 2015. Privacy features of Ethereum protocol were essentially the same as those of Bitcoin, e.g., transparent data, pseudonymous identities. Actually, original privacy level of Ethereum was potentially worse than that of Bitcoin because the unencrypted smart contract data could potentially reveal much more than the mere amount of bitcoins that is disclosed in Bitcoin transaction.

Since the Byzantium version release in 2017, Ethereum has had support for zk-SNARKs. At that time precompiled contracts were added to execute certain mathematical operations, like operations on elliptic curves, that relate to the verification of zk-SNARKs. A precompiled contract is a smart contract hardcoded in the Ethereum protocol. The contract is not executed in EVM, but instead it is run on the machine that hosts the Ethereum client. Zk-SNARK verification-related operations benefit from this, because it would have been very inefficient to run them in the stack based EVM, and the gas cost would have been tremendous. Smart contracts can use precompiled contracts in order to verify zk-SNARKs. In essence, this provides a way for dApps to build a privacy layer on top of the Ethereum protocol. Sensitive input data for a smart contract can be kept out of the blockchain, and only a proof, zk-SNARK, of their validity for the smart contract is needed for the blockchain where it is verified. The creation of zk-SNARKs and the trusted setup are the responsibility of the dApp, outside the Ethereum.

Numerous third party solutions have been released for Ethereum that are focused on privacy. ZoKrates*[45] is a toolbox, that aims to help the implementation of dApps that utilize zk-SNARKs. First, the developer creates a file that describes a verification program in ZoKrates' domain specific language. Then the toolbox provides a chain of conversions to produce the needed components. Arithmetic circuit is generated, CRS is constructed in the setup phase, a smart contract described in Ethereum's high level language is outputted that includes the verification key from the CRS. The prover is provided with a utility to create a zk-SNARK from a witness and the previously described CRS and circuit representation. It should be noted that the setup mentioned here is really trusted only by the party executing it, the creator of the smart contract. A trusted setup, that could be trusted by anyone, is not at least yet available in ZoKrates.

Ethereum protocol-level support for zk-SNARKs is not in sight. Zcash needs two arith-

*https://zokrates.github.io/

metic circuits, one for Spend and other for Output operation, to manage all the needed zk-SNARKs. Because of the Turing completeness of the EVM language, Ethereum cannot cope just with fixed pre-created circuits. Basically every smart contract would need its own circuit that should be created automatically and stored in the blockchain. The trusted setup would also be very challenging.

Because the zk-SNARK enabled privacy is not in the Ethereum blockhain layer, but above it in smart contract layer, from the transaction it is visible that a specific account x, that is Ethereum's counterpart for address, is interacting with a smart contract y. This may not be a privacy issue if the interaction details are adequately hidden with zk-SNARKs.

5.3 Monero

Monero is a privacy focused cryptocurrency that was established in 2014. It utilizes various cryptographic primitives to obtain privacy, some of which use zero-knowledge proofs. Transaction structure in Monero is similar to that in Bitcoin: input addresses, signing with corresponding private keys and output addresses with values. Monero uses cryptographic techniques to hide input addresses, output addresses and transaction values, which are called ring signatures, one-time keys and Pedersen commitments associated with range proofs, respectively.

In a *ring signature scheme* [103] the signer s specifies a set S of possible signers, i.e., set of public keys, such that $s \in S$. As with regular signature scheme, the private key of signer s is used in signing. It can only be deduced from the signature that it was created by a member of S, without revealing the identity of the actual member. To validate such signature, it contains a non-interactive zero-knowledge argument on random oracle model. When sender creates transaction they specify a set of n "decoy" inputs from the blockchain that have equal value than sender's input and then sign a ring signature for this set that includes also sender's input. Monero has been using a variant of ring signature that is also *traceable* [51] to prevent double spending. One private key can be used to sign only one ring signature. If it is used second time, this is detectable in the scheme, and the corresponding transaction is deemed invalid. This ring signature primitive provides an anonymity set for a sender's address, and it is also enforced by the Monero protocol becauset minimum ring size is set to be 11 (in 2020).

Monero provides a mechanism of *one-time keys*. Let us denote the recipient's public address as $addr_r$, the related public key as pk_r and verifier's private key as sk_r. In transaction

the output address is not directly a recipient's public address. Instead, the sender is enforced according to the protocol to create a new public key $pk_r^{'}$ that is derived from the recipient's address and a random value provided by sender, and put that new public key as output address. For everyone else than the sender or recipient the new public key is unlinkable to recipient's address, or to any other address. When examining transactions, the recipient notices that transaction output address $pk_r^{'}$ is derivable from the private key sk_r. The scheme also enables the recipient to recover from $pk_r^{'}$ and sk_r the new private key $sk_r^{'}$, with which they can spend this transaction output in new transaction. The underlying cryptographic primitive to this confidential delivery of secret key is *Diffie-Hellman key exchange* [40].

Ring signatures and one-time keys were Monero's original privacy-enhancing ingredients introduced in Monero's underlying CryptoNote protocol [105]. To further enhance privacy Monero launched in 2017 new "Ring Confidential Transactions" (RingCT) [97] scheme. Transaction values are hidden by *Pedersen commitments*. In transaction the sum of the input values must be equal than the sum of output values. Thanks to the additively homomorphic property of Pedersen commitment scheme, transaction balance check can be done with committed values instead of plaintext values. The sum of committed input values must be equal than the sum of committed output values. This description still needs adjustments to take into account the ring signatures and decoy inputs that come with them; also the ring signature scheme needs modifications [97].

A more detailed view also reveals that the scheme needs to be strengthened with *range proofs*. Negative values would cause inconsistencies because the balance check with committed values would hold, for example, with plaintext values $(1 + 2) = (-5 + 8)$, and the transaction would create 5 coins out of thin air*. Also large positive values are harmful because the commitment space in Pedersen commitment is a finite field, so as the values increase they wrap around according to modulo arithmetics. Each output value needs a range proof, which proves that the value is in certain positive range, let it be denoted as $[0, 2^n]$. In RingCT scheme a range proof for an output value consists of n ring signatures, which essentially prove that the value can be represented as a binary expansion $b = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_n \cdot 2^n$ where $b_i \in \{0, 1\}$.

Range proofs formed a kind of bottleneck in the RingCT scheme, because a vast majority of transaction size consisted of range proofs. In 2018 Monero introduced Bulletproofs scheme [30], that brought efficiency to range proofs. Bulletproofs is a NIZK argument

*The negative output value could be subsequently just ignored.

protocol where the size of a range proof for statement $x \in [0, 2^n]$ is logarithmic to n. Comparing to RingCT where the the range proof size is linear to n, this is a substantial saving, and more savings follow from aggregating range proofs of all transaction outputs to a single range proof. Even 80% drops in transaction sizes have been reported since Bulletproofs deployment*.

5.4 Discussion

We have now gone through three examples of blockchain applications that utilize zero-knowledge proofs. In relation to these examples, the aspects of privacy, security and viability will be considered next.

Privacy

In can be said that privacy is the raison d'être for zero-knowledge proofs in blockchains. Next we assess the privacy aspects of Monero and Zcash.

In Monero, the transaction graph is severely obfuscated by the ring signatures, one time keys and transaction value hiding. Some analysis can still be made of it. For example, Möser et. al [93] studied traceability of Monero transactions, by which they meant how likely is that the real input of the transaction could be deduced. They pointed out that 0-mixin transactions, i.e., transactions whose inputs have no decoy inputs defined in ring signatures, are not only themselves traceable, but also increase traceability to some extent in subsequent transactions. The possibility to omit decoy inputs can be seen as a design flaw, that already had been patched in November 2017, when above research was conducted. Another weakness observed in [93] was the use of a suboptimal distribution from which the decoy inputs were sampled. Usually the real input was the newest input in the ring signature. A *spend-time* of an input means the time elapsed since the transaction whose output this input spends. Currently the input sampling distribution in Morero reflects more the usual spend times of real inputs.

In Zcash the shielded transactions, i.e., transactions that are verified using the embedded zk-SNARKs, do not disclose the transaction graph at all. In a shielded transaction, only its timestamp and transaction fee can be seen by an outside observer. If this would be the whole picture, complete privacy would be very close, assuming the NIZK scheme is se-

*https://web.getmonero.org/resources/moneropedia/bulletproofs.html

cure. However, a vast majority of Zcash transactions are not shielded. These transactions have either only inputs or outputs in the shielded pool, or they are totally transparent. Transparent transactions have privacy level of Bitcoin. To a some extent, transparent transactions also illuminate shielded pool activities, as studies have shown [101, 82, 14]. These studies describe certain identifiable patterns of transactions to and from the shielded pool.

The dominance of transparent transactions in Zcash severely impairs the privacy achieved by the use of zk-SNARKs. Practical reasons probably largely explain the popularity of transparent activity. Creation of a shielded transaction, its zk-SNARK in particular, is computationally expensive, although it has become substantially easier with the Sapling upgrade. Initially there were no cryptocurrency exchanges or wallets that would support sending zcash to shielded addresses. Currently the shielded transaction accessibility is better, but according to statistics*, the portion of shielded transactions has not essentially grown.

It can be concluded that the privacy issues of Monero and Zcash are caused by other factors than the use of zero-knowledge proofs. In Monero zero-knowledge proofs are only a part of the cryptographic scheme, in Zcash zero-knowledge proof scheme is not widely used.

Security

Privacy and not only privacy can be lost, if the cryptographic scheme is not secure. Recall that a proof of security for a cryptographic scheme means that some security properties of the scheme are proved mathematically based on assumptions. Typically, general purpose cryptographic schemes have security proofs. If the implementation deviates from the original scheme or the implementation is a composition of multiple schemes, it may need its own security proof. In practice, absence of security proof has not prevented the use of such implementation.

The original article [97] of Monero's RingCT scheme did not include security proof. Its security properties were formalized later [109], when RingCT had already been deployed in Monero. Zk-SNARK schemes in most cases have rigid security proofs, as in the QAP-based scheme GGPR13 [55]. Zcash's original zk-SNARK implementation, BCTV14 [12], was based on GGPR13 with some tweaks, which would necessitate a separate security

*https://explorer.zcha.in/statistics/transactions

proof. The security proof was provided later [25], when Zcash had already been released. That security proof itself was found to be erroneous [53] and a severe vulnerability was found from BCTV14 system. The following zk-SNARK construction of Zcash [27], that is a modification of the Groth16 proof system [75] was introduced with a security proof.

Security proof is typically based on assumptions. From a security point of view, "strong" is not a favorable qualifier for an assumption, as it may mean that the assumption has not been well studied, or it is more prone to invalidation compared to another assumption. For Monero, the strongest assumption can be considered to be the random oracle model whose issues are discussed in Sections 2.2 and 4.4. Zcash with its zk-SNARK scheme has assumptions that are considered strong. For GGPR13 the strongest is the "knowledge of exponent" assumption, for Groth16 the assumption of generic group model (GGM). The GGM resembles random oracle model in the sense that it is also an idealized construction that helps in formulating security proofs. The GGM also has an analogous example with random oracle model of a cryptographic scheme that is secure in the idealized model, but insecure with any concrete instantiation of that model [39]. With QAP-based zk-SNARKs, which are used in Zcash, an unavoidable encumbrance is also the CRS with its trusted setup that brings its own practical difficulties.

It may be that the above mentioned strong or uninstantiable assumptions are more of a headache for the theoretically inclined people. Implementing practical cryptographic schemes may also require a pragmatic stance to the assumptions.

Viability and recent trends

We have seen that CRS model based zk-SNARK schemes can have assumptions that are almost comparable to the random oracle model. Therefore, one may ask whether zk-SNARKs could be constructed without the CRS model, only with the random oracle model, which as such is a simple and efficient tool and does not need a trusted setup. Recall that the NIZK application of the random oracle model, the Fiat-Shamir transform, converts interactive zero-knowledge proof into non-interactive. One critical requirement for zk-SNARK is succinctness, and it should be noted that the Fiat-Shamir transform does not in itself provide succinctness, so the underlying interactive protocol must be succinct. In fact, in recent years NIZK constructions have been introduced in random oracle model, which can be classified as zk-SNARKs on the basis of succinctness and other properties.

Above discussed QAP-based zk-SNARK schemes, GGPR13 and Groth16 are optimal for

blockchains in the sense that they are so "unbalanced". The proof size $\mathcal{O}(1)$ and verifier's complexity $\mathcal{O}(1)$ are very small compared to prover's execution time $\mathcal{O}(n \log n)$. The proof is stored in the blockchain and the proof verification is done at each node in the network, so their lightness is critical. A recent example of zk-SNARK scheme where the prover's and verifier's complexities are more balanced is Libra [112]. In Libra the execution time for prover is $\mathcal{O}(n)$ and the proof size and verifier's execution time is $\mathcal{O}(d \log n)$, where n denotes the circuit size and d the circuit depth. The prover's work is easier than in Groth16, but especially the proof size may make this scheme still impractical for blockchains. In one example of the article [112], the proof size with Libra was 51 KB, while with Groth16 it was only 192 B.

There seems to be a trade-off between prover's complexity and verifier's complexity. In this respect, more balanced schemes may find use outside blockchains, for example in *verifiable computation* [54]. In verifiable computation, computing is outsourced to untrusted workers, which reside in cloud platforms, for example. In verifiable computation the worker returns the result of the computation along with a succinct proof that the result is valid. In this context the zero-knowledge property is not considered, and SNARK is referred to without "zk".

The burden of the trusted setup has not lightened. Much work has been done in order to reduce the need for trusted setup. Recall that in GGPR13 scheme the CRS, and therefore the trusted setup are circuit specific. The previously mentioned Libra has an *universal CRS* that can be reused with arbitrary circuits of limited size. The CRS construction described by Groth et. al [76] is not only universal, but also *updatable*. Being updatable means that the CRS can be updated after its creation, and if even one of the parties that contributed to the creation or updating of the CRS is honest, then the CRS is secure. Therefore, the security of the CRS does not only depend on the initial trusted setup, but its security can be enhanced at a later stage.

As was noted before, zk-SNARKs have been achieved also with the random oracle model. One example of such scheme is SuperSonic [31] where the proof size is logarithmic to the circuit size; it is around 10 KB with reasonable sized circuits. Currently, in 2020, the QAP-based schemes such as Groth16 are still in a class of their own in terms of small proof size and verifier's complexity. Schemes with other desired properties, such as smaller prover's complexity, an universal or updatable CRS, or no trusted setup, pay for these properties with a larger proof size and verifier's complexity. With recent developments in non-QAP zk-SNARK schemes, this proof size trade-off may already be acceptable for

some blockhain applications.

6 Conclusions

Zero-knowledge proofs have proven to be a usable way to increase privacy in blockchain applications. The cryptographic foundation is strong, and zero-knowledge proofs have become sufficiently efficient for practical use. Blockchains are a felicitous use case for zero-knowledge proofs because the transactions in a blockchain must be publicly verifiable. Depending on the extent of use, zero-knowledge proofs can reduce the information revealed from a transaction to an absolute minimum, so only its validity is revealed in the verification. Zcash reaches this highest obfuscation level, only the transaction fee is disclosed.

Zero-knowledge proofs can also be used as part of cryptographic obfuscation mechanism, as is done with Monero. Ethereum enables zero-knowledge proofs for smart contracts which form an abstraction layer on top of the blockchain protocol. When comparing Monero and Zcash, the amount of information revealed about transactions is lower in Zcash because everything is concealed with zero-knowledge proofs. Therefore, it could be argued that Zcash provides better privacy. On the other hand, the difference in privacy levels may not be relevant given the other obfuscation mechanisms in Monero.

The situation is also affected by other factors. If users have the possibility to opt-out of privacy enhancing features, it can erode the privacy of all users. This has been the case with Monero in the past, and still with Zcash at the time of writing. The lack of infrastructure support, i.e., lack of cryptocurrency wallets and exchanges that support the privacy preserving shielded transaction, has been a major explanatory factor for the privacy opt-out in Zcash. Third party cryptocurrency services have been struggling to digest the high computational time and memory requirements that are related to the construction of zero-knowledge proofs in Zcash. Still, a vast majority of Zcash transactions are transparent, so this may well make privacy worse in Zcash than in Monero.

The blockchain setting, where transactions must be publicly verifiable, requires non-interactive protocol for zero-knowledge proofs. The Fiat-Shamir transform provides a partial solution and is ready to be used as a sub-protocol, but it alone does not provide succinct enough proof for the entire transaction. Constructs called zk-SNARKs have been promising candidates, and when the QAP-based zk-SNARK scheme was introduded by Gennaro et. al they became a practical solution to create NIZK proof for the entire

transaction. This QAP-based scheme that provides constant sized proof and verification time is still the best in terms of those properties. A major drawback of it and other similar schemes is the dependence of trusted setup. Recent advances with zk-SNARKs have brought attractive alternatives that offer other favorable properties, such as shorter proving time or the freedom from trusted setup.

Cryptographic assumptions for Fiat-Shamir transform and zk-SNARKs are quite strong, at least compared to what cryptographers are generally accustomed to. It is difficult to get a clear picture of how great a threat these assumptions pose and which of them is the most vulnerable. On top of this there is the ongoing cryptography-wide preparation for the quantum era. Certain cryptographic assumptions are invalidated, if the adversary has access to a quantum computer with a sufficient number of qubits . It can be said that some currently used cryptographic schemes are "quantum resistant" [110], and some are not [108], including the schemes used within NIZK.

Technological advancements, such as cryptocurrencies, are not completely detached from societal issues. Already Bitcoin, with its supposed anonymity, has attracted users involved in illegal activity. Criminals have also found newer privacy-focused cryptocurrencies. Articles related to cryptocurrency traceability sometimes discuss these topics as a motivating factor for research [36], and for example, mixing services are sometimes referred to as "[money] laundering services". Cryptocurrencies, by definition, do not have user identification or supervising intermediaries. From the point of view of the cryptocurrency protocol, "good" and "bad" activities are indistinguishable. However, this applies also to cryptographic protocols in general. Problems related to illegal activities may be unavoidable, but it is generally believed that the benefits of blockchain applications outweigh them.

Bibliography

[1] W. Aiello and J. Hastad. "Statistical Zero-Knowledge Languages Can Be Recognized in Two Rounds". In: *Journal of Computer and System Sciences* 42 (1991), pp. 327–345.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (May 1998), pp. 501–555. ISSN: 0004-5411. DOI: 10.1145/278298.278306. URL: https://doi.org/10.1145/278298.278306.

[3] S. Arora and S. Safra. "Probabilistic Checking of Proofs: A New Characterization of NP". In: *J. ACM* 45.1 (Jan. 1998), pp. 70–122. ISSN: 0004-5411. DOI: 10.1145/273865.273901. URL: https://doi.org/10.1145/273865.273901.

[4] L. Babai. "Trading Group Theory for Randomness". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 421–429. ISBN: 0897911512. DOI: 10.1145/22145.22192. URL: https://doi.org/10.1145/22145.22192.

[5] A. Back. *Hashcash - A Denial of Service Counter-Measure*. Sept. 2002. URL: http://www.hashcash.org/hashcash.pdf.

[6] D. Bayer, S. Haber, and W. S. Stornetta. "Improving the Efficiency and Reliability of Digital Time-Stamping". In: *Sequences II*. Ed. by R. Capocelli, A. De Santis, and U. Vaccaro. New York, NY: Springer New York, 1993, pp. 329–334. ISBN: 978-1-4613-9323-8.

[7] Bellare. "A Note on Negligible Functions". In: *J. Cryptol.* 15.4 (Sept. 2002), pp. 271–284. ISSN: 0933-2790. DOI: 10.1007/s00145-002-0116-x.

[8] M. Bellare, G. Fuchsbauer, and A. Scafuro. "NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by J. H. Cheon and T. Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 777–804. ISBN: 978-3-662-53890-6.

[9] M. Bellare and O. Goldreich. "On Defining Proofs of Knowledge". In: *Advances in Cryptology — CRYPTO' 92*. Ed. by E. F. Brickell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 390–420. ISBN: 978-3-540-48071-6.

[10] M. Bellare and P. Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. Fairfax, Virginia, USA: Association for Computing Machinery, 1993, pp. 62–73. ISBN: 0897916298. DOI: 10.1145/168588.168596. URL: https://doi.org/10.1145/168588.168596.

[11] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs". In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 287–304. DOI: 10.1109/SP.2015.25.

[12] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. *Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture*. Cryptology ePrint Archive, Report 2013/879. 2013. URL: https://eprint.iacr.org/2013/879.

[13] D. Bernhard, O. Pereira, and B. Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by X. Wang and K. Sako. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 626–643. ISBN: 978-3-642-34961-4.

[14] A. Biryukov and D. Feher. "Privacy and linkability of mining in zcash". In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2019, pp. 118–123.

[15] A. Biryukov and D. Khovratovich. "Equihash: Asymmetric proof-of-work based on the generalized birthday problem". In: *Ledger* 2 (2017), pp. 1–30.

[16] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "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. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 326–349. ISBN: 9781450311151. DOI: 10.1145/2090236.2090263. URL: https://doi.org/10.1145/2090236.2090263.

[17] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. "Succinct Non-interactive Arguments via Linear Interactive Proofs". In: *Theory of Cryptography*. Ed. by A. Sahai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 315–333. ISBN: 978-3-642-36594-2.

[18] *Bitcoin community Wiki*. Accessed 15.11.2020. URL: https://en.bitcoin.it/wiki/.

[19] *Bitcoin community Wiki, CoinJoin*. Accessed 15.11.2020. URL: https://en.bitcoin.it/wiki/CoinJoin.

[20] M. Blum, A. De Santis, S. Micali, and G. Persiano. "Noninteractive Zero-Knowledge". In: *SIAM Journal on Computing* 20.6 (1991), pp. 1084–1118. DOI: 10.1137/0220068. eprint: https://doi.org/10.1137/0220068. URL: https://doi.org/10.1137/0220068.

[21] M. Blum, P. Feldman, and S. Micali. "Non-Interactive Zero-Knowledge and Its Applications". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 103–112. ISBN: 0897912640. DOI: 10.1145/62212.62222. URL: https://doi.org/10.1145/62212.62222.

[22] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath. "Dandelion: Redesigning the Bitcoin Network for Anonymity". In: *Proc. ACM Meas. Anal. Comput. Syst.* 1.1 (June 2017). DOI: 10.1145/3084459. URL: https://doi.org/10.1145/3084459.

[23] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: *Advances in Cryptology – CRYPTO 2019*. Ed. by A. Boldyreva and D. Micciancio. Cham: Springer International Publishing, 2019, pp. 67–97. ISBN: 978-3-030-26954-8.

[24] D. Boneh, E.-J. Goh, and K. Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography*. Ed. by J. Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 325–341. ISBN: 978-3-540-30576-7.

[25] S. Bowe, A. Gabizon, and M. D. Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2018, pp. 64–77.

[26] S. Bowe, A. Gabizon, and M. D. Green. "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK". In: *Financial Cryptography and Data Security*. Ed. by A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 64–77. ISBN: 978-3-662-58820-8.

[27] S. Bowe, A. Gabizon, and I. Miers. *Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model*. Cryptology ePrint Archive, Report 2017/1050. 2017. URL: https://eprint.iacr.org/2017/1050.

[28] J. Boyar, I. Damgå, and R. Peralta. "Short Non-Interactive Cryptographic Proofs". In: *Journal of Cryptology* 13.4 (2000), pp. 449–472.

[29] G. Brassard, D. Chaum, and C. Crépeau. "Minimum disclosure proofs of knowledge". In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 156–189. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(88)90005-0. URL: http://www.sciencedirect.com/science/article/pii/0022000088900050.

[30] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 315–334.

[31] B. Bünz, B. Fisch, and A. Szepieniec. *Transparent SNARKs from DARK Compilers*. Cryptology ePrint Archive, Report 2019/1229. 2019. URL: https://eprint.iacr.org/2019/1229.

[32] V. Buterin. "A next-generation smart contract and decentralized application platform". In: (2013). URL: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf.

[33] R. Canetti, O. Goldreich, and S. Halevi. "The Random Oracle Methodology, Revisited". In: *J. ACM* 51.4 (July 2004), pp. 557–594. ISSN: 0004-5411. DOI: 10.1145/1008731.1008734. URL: https://doi.org/10.1145/1008731.1008734.

[34] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1880446391.

[35] *CoinJoins as a Percentage of All Bitcoin Payments*. Accessed 15.11.2020. URL: https://en.longhash.com/news/coinjoins-as-a-percentage-of-all-bitcoin-payments-have-tripled-to-409-over-the-past-year.

[36] J. Crawford and Y. Guan. "Knowing your Bitcoin Customer: Money Laundering in the Bitcoin Economy". In: *2020 13th International Conference on Systematic Approaches to Digital Forensic Engineering (SADFE)*. 2020, pp. 38–45.

[37] *Cryptocurrency, The Merriam-Webster.com Dictionary*. Accessed 15.11.2020. URL: https://www.merriam-webster.com/dictionary/cryptocurrency.

[38] I. Damgård. "Non-Interactive Circuit Based Proofs and Non-Interactive Perfect Zero-knowledge with Preprocessing". In: *Advances in Cryptology — EUROCRYPT' 92*. Ed. by R. A. Rueppel. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 341–355. ISBN: 978-3-540-47555-2.

[39] A. W. Dent. "Adapting the weaknesses of the random oracle model to the generic group model". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2002, pp. 100–109.

[40] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654. ISSN: 1557-9654. DOI: 10.1109/TIT.1976.1055638.

[41] I. Dinur. "The PCP Theorem by Gap Amplification". In: *J. ACM* 54.3 (June 2007), 12–es. ISSN: 0004-5411. DOI: 10.1145/1236457.1236459. URL: https://doi.org/10.1145/1236457.1236459.

[42] J. Don, S. Fehr, C. Majenz, and C. Schaffner. "Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model". In: *Advances in Cryptology – CRYPTO 2019*. Ed. by A. Boldyreva and D. Micciancio. Cham: Springer International Publishing, 2019, pp. 356–383. ISBN: 978-3-030-26951-7.

[43] C. Dwork and M. Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology — CRYPTO' 92*. Ed. by E. F. Brickell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147. ISBN: 978-3-540-48071-6.

[44] C. Dwork, M. Naor, and A. Sahai. "Concurrent Zero-Knowledge". In: *J. ACM* 51.6 (Nov. 2004), pp. 851–898. ISSN: 0004-5411. DOI: 10.1145/1039488.1039489. URL: https://doi.org/10.1145/1039488.1039489.

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

[46] T. El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. Santa Barbara, California, USA: Springer-Verlag, 1985, pp. 10–18. ISBN: 0387156585.

[47] I. Eyal and E. G. Sirer. "Majority Is Not Enough: Bitcoin Mining Is Vulnerable". In: *Financial Cryptography and Data Security*. Ed. by N. Christin and R. Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 436–454. ISBN: 978-3-662-45472-5.

[48] U. Feige, D. Lapidot, and A. Shamir. "Multiple non-interactive zero knowledge proofs based on a single random string". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. Oct. 1990, 308–317 vol.1. DOI: 10.1109/FSCS.1990.89549.

[49] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194. ISBN: 978-3-540-47721-1.

[50] L. Fortnow. "The Complexity of Perfect Zero-Knowledge". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 204–209. ISBN: 0897912217. DOI: 10.1145/28395.28418. URL: https://doi.org/10.1145/28395.28418.

[51] E. Fujisaki and K. Suzuki. *Traceable Ring Signature*. Cryptology ePrint Archive, Report 2006/389. 2006. URL: https://eprint.iacr.org/2006/389.

[52] M. Furer, O. Goldreich, and Y. Mansour. *On Completeness and Soundness in Interactive Proof Systems*. 1989.

[53] A. Gabizon. *On the security of the BCTV Pinocchio zk-SNARK variant*. Cryptology ePrint Archive, Report 2019/119. 2019. URL: https://eprint.iacr.org/2019/119.

[54] R. Gennaro, C. Gentry, and B. Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers". In: *Annual Cryptology Conference*. Springer. 2010, pp. 465–482.

[55] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. *Quadratic Span Programs and Succinct NIZKs without PCPs*. Cryptology ePrint Archive, Report 2012/215. 2012. URL: https://eprint.iacr.org/2012/215.

[56] C. Gentry. "A Fully Homomorphic Encryption Scheme". PhD thesis. Stanford, CA, USA, 2009. ISBN: 9781109444506.

[57] C. Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC ’09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. ISBN: 9781605585062. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[58] C. Gentry, J. Groth, Y. Ishai, C. Peikert, A. Sahai, and A. Smith. “Using Fully Homomorphic Hybrid Encryption to Minimize Non-interative Zero-Knowledge Proofs”. In: *Journal of cryptology* 28.4 (2015), pp. 820–843.

[59] S. Ghosh. *Distributed Systems: An Algorithmic Approach, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466552972.

[60] O. Goldreich. “Concurrent Zero-Knowledge with Timing, Revisited”. In: *Theoretical Computer Science: Essays in Memory of Shimon Even*. Ed. by O. Goldreich, A. L. Rosenberg, and A. L. Selman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 27–87. ISBN: 978-3-540-32881-0. DOI: 10.1007/11685654_2. URL: https://doi.org/10.1007/11685654_2.

[61] O. Goldreich. *Foundations of Cryptography: Basic Tools*. USA: Cambridge University Press, 2000. ISBN: 0521791723.

[62] O. Goldreich. “Modern Cryptography, Probabilistic Proofs and Pseudorandomness”. In: (2000). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.485&rep=rep1&type=pdf.

[63] O. Goldreich and J. Håstad. “On the Complexity of Interactive Proofs with Bounded Communication”. In: *Inf. Process. Lett.* 67.4 (Aug. 1998), pp. 205–214. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(98)00116-1. URL: https://doi.org/10.1016/S0020-0190(98)00116-1.

[64] O. Goldreich and A. Kahan. “How to Construct Constant-Round Zero-Knowledge Proof Systems for NP”. In: *Journal of Cryptology* 9 (June 1997). DOI: 10.1007/s001459900010.

[65] O. Goldreich and H. Krawczyk. “On the Composition of Zero-Knowledge Proof Systems”. In: *SIAM Journal on Computing* 25 (Jan. 1996). DOI: 10.1007/BFb0032038.

[66] O. Goldreich, S. Micali, and A. Wigderson. “Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems”. In: *J. ACM* 38.3 (July 1991), pp. 690–728. ISSN: 0004-5411. DOI: 10.1145/116825.116852. URL: https://doi.org/10.1145/116825.116852.

[67] O. Goldreich and Y. Oren. "Definitions and Properties of Zero-Knowledge Proof Systems". In: *J. Cryptol.* 7.1 (Dec. 1994), pp. 1–32. ISSN: 0933-2790. DOI: 10.1007/BF00195207. URL: https://doi.org/10.1007/BF00195207.

[68] O. Goldreich, S. Vadhan, and A. Wigderson. "On Interactive Proofs with a Laconic Prover". In: *Comput. Complex.* 11.1/2 (June 2002), pp. 1–53. ISSN: 1016-3328. DOI: 10.1007/s00037-002-0169-0. URL: https://doi.org/10.1007/s00037-002-0169-0.

[69] S. Goldwasser and M. Sipser. "Private Coins versus Public Coins in Interactive Proof Systems". In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. STOC '86. Berkeley, California, USA: Association for Computing Machinery, 1986, pp. 59–68. ISBN: 0897911938. DOI: 10.1145/12130.12137. URL: https://doi.org/10.1145/12130.12137.

[70] S. Goldwasser and Y. T. Kalai. "On the (In)security of the Fiat-Shamir paradigm". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 2003, pp. 102–113.

[71] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012. eprint: https://doi.org/10.1137/0218012. URL: https://doi.org/10.1137/0218012.

[72] S. Goldwasser, S. Micali, and R. L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM Journal on Computing* 17.2 (1988), pp. 281–308. DOI: 10.1137/0217017.

[73] A. Greenberg. "Prosecutors Trace $13.4M in Bitcoins From the Silk Road to Ulbricht's Laptop". In: *Wired* (Jan. 2015). URL: https://www.wired.com/2015/01/prosecutors-trace-13-4-million-bitcoins-silk-road-ulbrichts-laptop/.

[74] J. Groth. "Non-interactive Zero-Knowledge Arguments for Voting". In: *Applied Cryptography and Network Security*. Ed. by J. Ioannidis, A. Keromytis, and M. Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 467–482. ISBN: 978-3-540-31542-1.

[75] J. Groth. "On the Size of Pairing-Based Non-Interactive Arguments". In: *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 305–326. ISBN: 9783662498958.

[76] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. Cham: Springer International Publishing, 2018, pp. 698–728. ISBN: 978-3-319-96878-0.

[77] J. Groth, R. Ostrovsky, and A. Sahai. "New Techniques for Noninteractive Zero-Knowledge". In: *J. ACM* 59.3 (June 2012). ISSN: 0004-5411. DOI: 10.1145/2220357.2220358. URL: https://doi.org/10.1145/2220357.2220358.

[78] J. Groth, R. Ostrovsky, and A. Sahai. "Perfect Non-interactive Zero Knowledge for NP". In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 339–358. ISBN: 978-3-540-34547-3.

[79] S. Haber and W. S. Stornetta. "How to Time-Stamp a Digital Document". In: *Advances in Cryptology-CRYPTO' 90*. Ed. by A. J. Menezes and S. A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 437–455. ISBN: 978-3-540-38424-3.

[80] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. "Zcash protocol specification". In: *GitHub: San Francisco, CA, USA* (2020). Version 2020.1.14. URL: https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf.

[81] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan. "An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design." In: *WEIS*. Citeseer. 2015.

[82] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. "An empirical analysis of anonymity in zcash". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 463–477.

[83] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2014. ISBN: 9781466570269. URL: https://books.google.fi/books?id=OWZYBQAAQBAJ.

[84] J. Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. STOC '92. Victoria, British Columbia, Canada: Association

for Computing Machinery, 1992, pp. 723–732. ISBN: 0897915119. DOI: 10.1145/129712.129782. URL: https://doi.org/10.1145/129712.129782.

[85] J. Kilian and E. Petrank. "An Efficient Noninteractive Zero-Knowledge Proof System for NP with General Assumptions". In: *J. Cryptol.* 11.1 (Jan. 1998), pp. 1–27. ISSN: 0933-2790. DOI: 10.1007/s001459900032. URL: https://doi.org/10.1007/s001459900032.

[86] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176.

[87] G. Maxwell. *CoinJoin: Bitcoin privacy for the real world*. Accessed 15.11.2020. URL: https://bitcointalk.org/index.php?topic=279249.0.

[88] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. "Simple schnorr multi-signatures with applications to bitcoin". In: *Designs, Codes and Cryptography* 87.9 (2019), pp. 2139–2164.

[89] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. "A Fistful of Bitcoins: Characterizing Payments among Men with No Names". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain: Association for Computing Machinery, 2013, pp. 127–140. ISBN: 9781450319539. DOI: 10.1145/2504730.2504747. URL: https://doi.org/10.1145/2504730.2504747.

[90] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. 1996.

[91] R. C. Merkle. "Protocols for Public Key Cryptosystems". In: *1980 IEEE Symposium on Security and Privacy*. Apr. 1980, pp. 122–122. DOI: 10.1109/SP.1980.10006.

[92] S. Micali. "Computationally sound proofs". In: *SIAM Journal on Computing* 30.4 (2000), pp. 1253–1298.

[93] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, et al. "An empirical analysis of traceability in the monero blockchain". In: *Proceedings on Privacy Enhancing Technologies* 2018.3 (2018), pp. 143–163.

[94] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Accessed 15.11.2020. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[95] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. USA: Princeton University Press, 2016. ISBN: 0691171696.

[96] A. Narayanan and J. Clark. "Bitcoin's academic pedigree". In: *Communications of the ACM* 60.12 (Dec. 2017), pp. 36–45. ISSN: 0001-0782. DOI: https://doi.org/10.1145/3132259.

[97] S. Noether, A. Mackenzie, and T. Lab. "Ring Confidential Transactions". In: *Ledger* 1 (Dec. 2016), pp. 1–18. DOI: 10.5195/LEDGER.2016.34.

[98] Y. Oren. "On the cunning power of cheating verifiers: Some observations about zero knowledge proofs". In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. Oct. 1987, pp. 462–471. DOI: 10.1109/SFCS.1987.43.

[99] B. Parno, J. Howell, C. Gentry, and M. Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 238–252.

[100] A. Pfitzmann and M. Köhntopp. "Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology". In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by H. Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–9. ISBN: 978-3-540-44702-3. DOI: 10.1007/3-540-44702-4_1. URL: https://doi.org/10.1007/3-540-44702-4_1.

[101] J. Quesnelle. *On the linkability of Zcash transactions*. 2017. arXiv: 1712.01210 [cs.CR].

[102] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[103] R. L. Rivest, A. Shamir, and Y. Tauman. "How to Leak a Secret". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by C. Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565. ISBN: 978-3-540-45682-7.

[104] D. Ron and A. Shamir. "Quantitative Analysis of the Full Bitcoin Transaction Graph". In: *Financial Cryptography and Data Security*. Ed. by A.-R. Sadeghi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 6–24. ISBN: 978-3-642-39884-1.

[105] N. van Saberhagen. *CryptoNote v 2.0*. 2013. URL: https://cryptonote.org/whitepaper.pdf.

[106] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized anonymous payments from bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 459–474.

[107] *Secure Hash Standard*. Federal Inf. Process. Stds. (NIST FIPS) 180-4. NIST. 2015.

[108] P. W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[109] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen. "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero". In: *European Symposium on Research in Computer Security*. Springer. 2017, pp. 456–474.

[110] D. Unruh. "Non-interactive zero-knowledge proofs in the quantum random oracle model". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 755–784.

[111] G. Wood. *Ethereum: A secure decentralised generalised transaction ledger. Petersburg version.* Accessed 12.10.2020. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[112] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. *Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation*. Cryptology ePrint Archive, Report 2019/317. 2019. URL: https://eprint.iacr.org/2019/317.

[113] X. Yi, R. Paulet, and E. Bertino. *Homomorphic Encryption and Applications*. Springer Publishing Company, Incorporated, 2014. ISBN: 3319122282.

[114] *Zcash FAQ*. Accessed 15.11.2020. URL: https://z.cash/support/faq/.